Handbook for the VFind Security Took Kit(s) 8th Edition

The Leader in Total Security Solutions For Linux, UNIX and all *NIX systems

* Anti-virus
* Baseline Control
* Pattern Analysis
* Computer Forensics
* Operational Monitoring
* Self-Healing and Repair
* Command Line or GUI
* Lights Out or Manned
* Identification and Rendering of Complex Files.

CYBERSOFT OPERATING CORPORATION

1958 BUTLER PIKE, SUITE 100
CONSHOHOCKEN, PA. 19428 U.S.A.
Telephone:+1 610-825-4748?
Fax:+1 610/825-6785
Website: www.cybersoft.com
General Email: sales@cyber.com
Providing Security Tools Since 1988

Handbook for the VFind Security Took Kit(s)

Edition 8, May 2018
By
Peter V. Radatti
radatti@cyber.com Edited by: Justin Honer

Portions of this book have been written by Dr. Rick Perry and Barbara Higgins.

Edition 8.00 MONTH 2019
Edition 7.00 May 2018
Edition 6.00 March 2017
Edition 5.00 May 2017
Edition 4.00 December 2014
Edition 3.00 May 3, 2005
Edition 2.00 April, 22, 2005
Edition 1.01 April 21, 2004
First Printed, Edition 1.00 April 12, 2004

Authors

This book was written by the founder of CyberSoft, Peter V. Radatti. The section on the CVDL language was written by Doctor Richard Perry with contributions by Peter. The Lexical Analysis section was written by Barbara Lynn Higgins. The 7h Edition of this manual was edited by Justin Honer with technical support provided by Alec Hussey and Michael Krakower.

ISBN-13: 978-1533465139
ISBN-10: 1533465134

Table of Contents

Document Opening

Simple Virus Scanning Protocol (SVSP)

Tools

[FINISH ME]

Organization of Handbook

This handbook is organized into sections. This handbook contains information that has never been published before in addiion to information that was previously published but collated into this handbook for easy reference. The goal is to make this handbook the go-to place for all documentation related the the CyberSoft VSTK family of products. If any errors are found, or if there are suggestions for improvement of the handbook, please email support@cyber.com

Thank you,
Pete Radatti

This Handbook covers the following VSTK Major Versions

VSTK 179 through VSTK 184

All Versions before VSTK 179 are unsupported, and so will no longer be covered in the handbook.

Preface

Definitions

In this handbook, Unix, Linux and Apple Macintosh OS-X are used interchangeably. When using the name Unix, the handbook is referring to all Unix like systems. The words Microsoft and Windows are used to mean Microsoft Windows unless referring to a non-Microsoft Windows system such as X-Windows.

The word Virus is used to mean all forms of attack software, examples of which are viruses, worms, Trojan horses, droppers, vectors, "bombs" and hacks. CyberSoft does not make a distinction between different types of attack code as to what VSTK products will detect or not detect. If it is attack code CyberSoft wants to detect it.

VSTK generically refers to the VFind Security Tool Kit in all its different versions. This is different from when referring to part numbers such as VSTK, VSTK-T, etc.

Legal Notices

The VSTK tools are copyrighted programs with multiple patents issued and pending. This document is copyrighted, all rights reserved.

Activation Keys

VSTK/-T locks to a specific system using an activation key. The keys lock on nodename, not hostid or domain name. For example, a hostid for a system might be 214435 while the domain name associated with that system is www.cybersoft.com. The nodename for that same system could be webbox. Notice that there is no relationship between these forms of identification, other than they all refer to the same physical system. The nodename is what the operating system thinks it calls itself. The domain names are how a domain name servers resolves DNS requests into IP addresses. The hostid is hardware based and is supposed to be something like a serial number but can be easily changed by someone who knows how, typically using the boot monitor software of the system.

The disadvantage of using a hostid is that it is supposed to be hard to change. Using Domain Names is not wise, since they tend to be moved from platform to platform frequently. The advantage of using the nodename is that it is easy to change for the system administrator. If a server breaks and the hard disk is transferred to another compatible hardware platform, then the nodename of the system remains the same and the activation key will operate.

To create Activation Keys an account must be created on my.cybersoft.com. This account can be gained from CyberSoft Operating Corporation by requesting one along with your purchase order number or other identification that associates you with the purchased product.

To contact CyberSoft by phone call +1-610-825-4748, fax at +1-610-825-6785 or email at support@cyber.com.

A Short History of CyberSoft and VFind

Prior to starting CyberSoft, Peter Radatti (CyberSoft Chairman/CEO) spent over 15 years working in the aerospace and military industries for companies like Control Data Corporation (CDC) at the Naval Air Development Center, General Electric (GE) at the Valley Forge Space Center and Systems Research Applications (SRA) in Moorestown New Jersey. The idea for CyberSoft's first commercial product was crafted while at General Electric. GE had no interest in the product which was an anti-virus scanner for Sun Microsystem's Unix computers.

Determined, he purchased a Sun workstation and completed the project at his residence outside of work hours. When the program was completed, Peter initially planned to enter it into the public domain, until General Electric took an interest in the program and asked Peter to wait.

GE's interest in the program, called "VFind," caused Peter to realize that it might have commercial value. At the same time, Bell Atlantic also showed interest in the program. With two major corporations showing an interest in the program, Peter decided to invest his personal savings to demonstrate the program at the Unix Expo International in October 1991 at the Jacob Javits Center in Manhattan. The investment paid off, and enough profit was generated to launch CyberSoft, Inc.

CyberSoft, Inc is a corporation registered in Pennsylvania. It was founded July 29, 1988 and incorporated January 1, 1993. The first product was shown in 1990 but publicly announced at the Unix Expo International in New York City on October 30, 1991. On July 1, 2002 CyberSoft, Inc (CS). was reorganized as an intellectual property holding company, and all operations were spun off to a new corporation called CyberSoft Operating Corporation (COC).

Basic Computer Security

There are three basic ways to secure computers. When combined these three methods are called a Ring Security System. Almost everything else is an extension of one of these methods. Everything in computer security is a compromise that complies with the laws of diminishing returns. Simply put, the more secure a system is the less useful it is. A good security system makes a balance between these competing needs.

The Three Components of a Ring Security System

1. Compartmentalization
2. Fortressing
3. Reactive

The first method is **compartmentalization**. Compartmentalization basically means that the system is isolated and locked up. Everything else is a compromise with various strong and weak points but compartmentalization is the strongest method. It also provides the most restrictions thereby having the most impact on use of the system. A compartmentalized system will generally not have access to an outside network and be located in a specially prepared and locked room. This is sometimes referred to as air gap.

The second method is called **fortressing**. This process attempts to make a fort or castle out of the computer by hardening the security. It tries to keep the bad guys out. Modern computers are network connected and are very complex. Even with the best fortressing technology known, someone, somewhere will penetrate the fortress and gain entrance. Fortressing itself is generally implemented in layers. The first layer is the network configuration, network firewalls and anti-virus, operating system configuration for security, automatic update systems, local firewall, local virus scanners and extremely important, enforced security policies.

The third method is **reactive security**. Reactive security assumes that unauthorized people are going to get in, or authorized people are going to make unauthorized modification to the baseline, or worse yet, unauthorized users will make unauthorized modification. It then deals with that issue. The most important part of reactive security is baseline configuration control. In other words, keep the system baseline where it belongs, keep the system under control and keep it performing its function as long as possible in a hostile environment. Of course if a security breach is found it must be identified and fixed. Maintaining the system configuration with a security hole is of reduced value.

Each of the three methods has drawbacks. The best overall solution is to combine all three. This is common. The outermost ring provides compartmentalization to the extent possible. On a network this may be provided by configuration of the system, router and firewall and by physical security. The second ring is fortressing.

Remove all known exploits from the system and harden the configuration to make it as hard as possible to gain entry or control while allowing enough functionality to allow the system to continue performing its job. The innermost ring is the reactive ring. Maintain the baseline configuration at all costs. Correct unauthorized changes to the baseline. Remove all attacks that gain entry. Automate the process so that discipline is maintained.

There is also a "fourth" method, which is really a modification of the fortressing and reactive methods. CyberSoft considers this a fourth method because it is very important, and yet the most complex and expensive to implement. This fourth method is Update Management. Update Management is the process in which all manufacturer and organization mandated updates are installed. On a single system with a automated or semi automated update system, with a trusted user, this is not a problem.

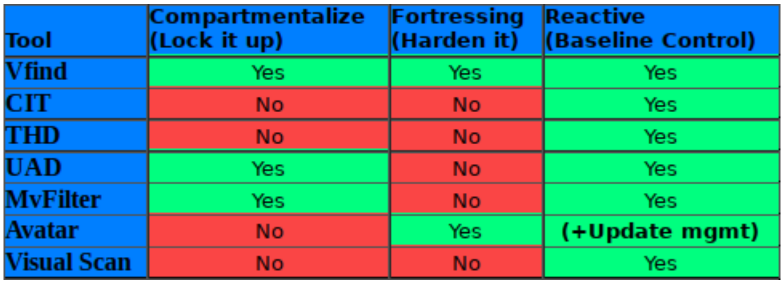
On networks of thousands of systems with users that are both trusted, untrusted, trained, untrained, capable and not capable of using the update system this becomes one of the most expensive and difficult computer security tasks. In addition it is also an area of large risk. If an update causes a critical process to fail and an automated update system has installed that update on thousands of systems the result can be a major catastrophe. Above and beyond this, every manufacturer has their own update management system. A typical computer may have dozens of software packages installed each with a different update system.

Even with all of these problems, update management becomes a critical problem because the most common reason for an update to be released is reactive. Someone found a security hole and it must be blocked. One of the most common reasons that computer viruses and worms spread so quickly on the Internet is because there are millions of systems that have not been updated or a new hole has been discovered. The Internet is susceptible to the agricultural problems of blight. Once a new attack comes along, it can wipe out an entire crop.

Update Management is a critical computer security tool. In fact, large organizations will not be able to wait for a manufacturer update to a problem and needs complete and proactive control over their update processes. Lightning fast in-house response may mean the difference between victory and defeat in an endeavor.

In addition to demonstrating ways to use the VFind Security ToolKits to solve the first three problems this handbook will also show methods of using these same tools to solve the Update Management problem.

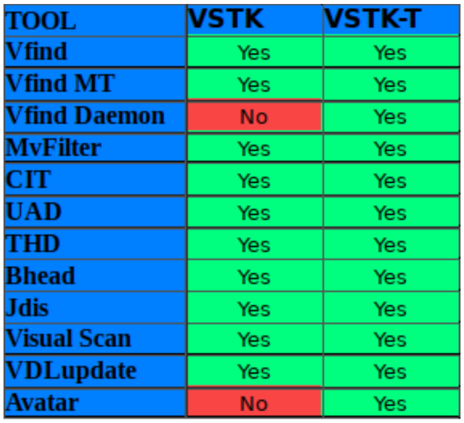
CyberSoft Tools and their use in the Tertiary Security Ring Model



Tool	Compartmentalize (Lock it up)	Fortressing (Harden it)	Reactive (Baseline Control)
Vfind	Yes	Yes	Yes
CIT	No	No	Yes
THD	No	No	Yes
UAD	Yes	No	Yes
MvFilter	Yes	No	Yes
Avatar	No	Yes	**(+Update mgmt)**
Visual Scan	No	No	Yes

Here is a table that explains what is contained in the VFind packages as of VSTK Package Version Number 179-184, except Visual Scan which was introduced in Version 180.

Table of Tools in Tool Kits (VSTK 180-184)



TOOL	VSTK	VSTK-T
Vfind	Yes	Yes
Vfind MT	Yes	Yes
Vfind Daemon	No	Yes
MvFilter	Yes	Yes
CIT	Yes	Yes
UAD	Yes	Yes
THD	Yes	Yes
Bhead	Yes	Yes
Jdis	Yes	Yes
Visual Scan	Yes	Yes
VDLupdate	Yes	Yes
Avatar	No	Yes

"Immunity" to Software Attacks

Many intelligent people seem to think that for one reason or another their systems are immune to software attacks such as viruses, trojans or worms. Not as often there are people who think their systems are immune to "hacker" attacks, either inside or outside. Often these people refer to firewalls, gateways and specific versions of an operating system in support of their argument.

There is no such thing as a safe or immune system and a system cannot be made totally safe. Even a system that has no connection to the outside world can be attacked. What a user can do is make a system as safe as possible. Generally this means hitting the law of diminishing returns, where making the system any safer also makes it non-functional.

In the past, people have asserted that Unix and other computers that utilize hardware instructions to provide a Supervisor mode of operation were immune to attack. The use of Supervisor mode is not necessary for viral infection or even any other form of attack; therefore, the argument is invalid. This argument is also supported by the fact that there are viruses in existence for Unix and other systems that utilize Supervisor mode. A white paper written by Tom Duff and M. Douglas McIlroy of AT&T Bell Laboratories contained in the USENIX 1989 Volume 2 journal not only attests to the existence of Unix viruses, but provides source code for an example.

White papers by Dr. Fred Cohen, a pioneer in the field of computer viruses, demonstrated that a computer virus could fully penetrate super user access on any computer, even properly configured security rated computers in a short period of time. Dr. Cohen performed his tests on multiple Unix systems. UPDATE: As of 2014 most major virus/trojan programs used to steal that were originally developed for Microsoft Windows have been ported to Unix/Linux systems.

Other people state falsely that dedicated single purpose computers such as process control systems, cell phone systems and embedded computers are immune. There are already cell phone viruses (Spain was struck with the first outbreak) and single purpose embedded computers have been infected. UPDATE: As of 2014, this is now common knowledge but at the time this was first written, the knowledge was considered hearsay.

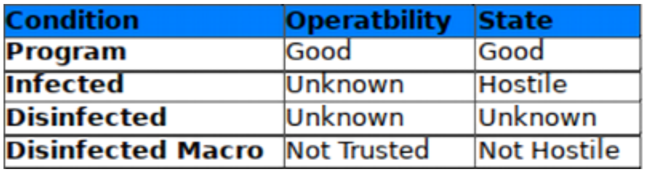
Some people even went so far as to design, build and field "immune" computers using ROM (read only memory) based operating systems. The theory was that if there is no way to modify the operating system, then it can not be infected or hacked. In the paper "From Little Acorns Mighty Viruses Grow" by Alan Glover of Pineapple Software, it was disclosed that the Acorn Archimedes computer (a British educational computer system), which holds all of its operating system and windowing systems locked in hardware based Read Only Memory, had been successfully infected by computer viruses.

This is an extreme case of hardware based protection, and yet it failed. As of January 1994, there were 52 virus families totaling 84 viruses affecting the system. If the Acorn, Unix, Macintosh, Microsoft Windows, cell phone and process controller computers can be infected or hacked, then any system can be. In 1993, Peter Radatti stated, "Those components of an operating system that are deemed necessary for practical use, such as copy, append, change permission settings and hundreds of other basic functions are the only necessary building blocks for viral code." and that, "Many simple and normal functions that may pass a security screen, when combined, implement a virus" or any other form of attack.

Disinfection of Viruses

All software is flawed. It is the nature of software. Programs which have run for 20 years without error suddenly, and without notice, fault. If a software executable which is flawed to begin with but at least tested (we hope) to operate properly under normal conditions is infected by a virus or modified by a Trojan or other software attack then the operability of the program is unknown and the state of the program is hostile. In addition, many viruses are poorly written and the author has no idea what effect it will have on the many different targets it may come across. Algebraically the equation would be fv(p) = p'. Function virus upon target program p yields p-prime. A disinfection program is of necessity a different program than the program that infected the file to begin with. Viruses do not run in reverse and undo the damage done. Again, the authors of disinfection program cannot test them against all combinations of viruses and target programs. The effect is a program that should never have been modified being modified twice. Algebraically the equation is fd(p') = p''. That is, function disinfection upon target program p-prime yields p-prime-prime. The result p-prime-prime is not equal to p. Even if by some rare chance the result of p'' is p, the state of the program is unknown unless a cryptographic hash of p prior to the first transformation exists and can be verified.

Table of Program Conditions after Disinfection



Condition	Operatbility	State
Program	Good	Good
Infected	Unknown	Hostile
Disinfected	Unknown	Unknown
Disinfected Macro	Not Trusted	Not Hostile

Programs which were infected by a virus and later disinfected by even the best disinfection program in existence, are still damaged and should be at best considered untrustworthy and at worst, should be considered seriously damaged.

There is one type of disinfection which while it will not yield the original file (p) it can yield a non-hostile result (p''). This is the case where a data file is infected. A perfect example of this is Microsoft OLE or XML files. OLE files have been extensively used for Microsoft Word documents and other types of documents.

Viruses which infect OLE documents are macros. They are interpreted by whatever is processing the file. Removal of all macros from an OLE document will yield a result that is still untrustworthy, but is at least known to be in a state which is not hostile. This distinguishment is made because many macros change the contents of the file. In a word processing document this may or may not be noticed. In a spreadsheet this may not be noticed. If a macro virus changed a document directing someone to do something, it may be serious but may be detectable by common sense. In a spreadsheet, a number or formula could be modified in such a way as to not be noticeable. There are real life examples of viruses which do this.

Disinfection of viruses has several drawbacks besides what has already been mentioned. First, they may leave a ghost* of the virus which can be detected by other anti-virus program. This is common. Secondly, the industry practice is to remove all macros. If a macro is essential to the proper use of the document, then the document is not recovered correctly. If a backup of the document is not available, then disinfection of these types of files are an acceptable starting place provided that the end user understands the limits of the disinfection program.

The only trustable method of disinfection of an executable program is to remove the program and replace it with a known good copy. Even if backup copies were not made, this is often an easy solution to implement by using the installation media. Disinfection should be considered an undesirable and temporary bandage until a fully trustable solution can be implemented.

*Some anti-virus products will delete a macro virus by flipping the delete flag. This leaves the macro virus intact and it will be detected by other products. The same is true of binary infectors, some anti-virus products will loop around the infection and change just enough of the code that they do not detect it themselves but other manufacturers products may still detect the attack since the majority of the attack code is present. This is called a ghost virus. CyberSoft does not do this, instead the MVFilter program will zero out the entire virus in addition to flipping the delete flag. There is no ghost virus for anyone to detect.

Computer Viruses

Here are the facts:

1. Computer viruses are real. They exist. Some people believe they do not exist.
2. If a computer is infected, it may not be apparent just be using it. Often, it requires an anti virus or integrity product to know if a system is infected.
3. Viruses travel in vectors. An open vector is an invitation and may result in a system infected with several different viruses simultaneously.
4. Computer viruses are a threat. The threat is proportional to the value of the system and the attached network.
5. The direct cost of a virus infection is proportional to the cost of restoring the system and the number of systems that must be restored. The indirect costs may be higher.
6. Computer viruses can bring down a network, including the Internet, by clogging it or by attacking the routers that operate the net.
7. Computer viruses are not created by the anti-virus industry.
8. One infected computer can damage the bandwidth of a network if the network is not configured with security in mind.
9. There are numerous viruses that were created over the course of many years, but in reality there are significantly less than the industry tries to make people believe there are. There are several reasons for this including the fact that many of the older viruses will not run on modern computers or the viruses require a transport format that is essentially obsolete, such as boot sector viruses and floppy diskettes. There is also the fact that some anti-virus companies will search for viruses that are moot. This is done solely for marketing reasons.

Super Critical Systems and Anti-Virus

In May 1995 Peter Radatti published a paper entitled Anti-Virus for Multimedia Publishers. This paper was likely the first time anyone proposed using more than one brand of anti-virus program on the same system. In fact, the paper suggested using three. Today this is a common practice on super critical systems.

The paper explained that each manufacture of anti-virus makes different trade-offs when designing their products. These trade-offs are necessary in order to make the product usable. Reference the law of diminishing returns. Peter suggested that super critical systems required a higher set point for diminishing returns than a normal system. Consequently, he also suggested using more than one manufacturers anti-virus and using multiple methods. An example is to use three virus scanners by different companies, one integrity system and one baseline control system. The more critical the system, the more tools and methods should be implemented. All phases of this concept, except for the multi-vendor idea, can be implemented using the standard VSTK products. Be careful that some manufacturers tools may not work well together.

In addition to the idea of multi-vendor products protecting the same system this paper also presented the CyberSoft Half Dozen Rules of Anti-Virus Common Sense. Here is an updated version:

CyberSoft Half Dozen Rules of Anti-Virus Common Sense

1. Always virus scan new software prior to installation. Especially true of Internet downloads.
2. Scan the system with a cryptographic hashing tool. Do this prior to an infection so changes can be detected.
3. Scan the system and use the integrity system often, at a minimum daily. Update the virus definitions daily or more often.
4. Use the keyboard or screen lock to keep people out of unattended systems.
5. Scan files copied from other systems prior to use.
6. Backup systems often. Destroyed data can be retrieved if it has been archived.

Different Versions of the VFind Security Toolkit and their Components

The VFind Security Tool Kits are general purpose computer security oriented products. Consider them as tool boxes filled with various tools that can be used to solve problems or build things. While these tools were all created with computer security as the primary problem set, many customers have used them to solve problems that are unrelated to computer security. As familiarity is developed with the individual tools and the tool concept, ways in which to use these tools to solve unique problems will begin to surface.

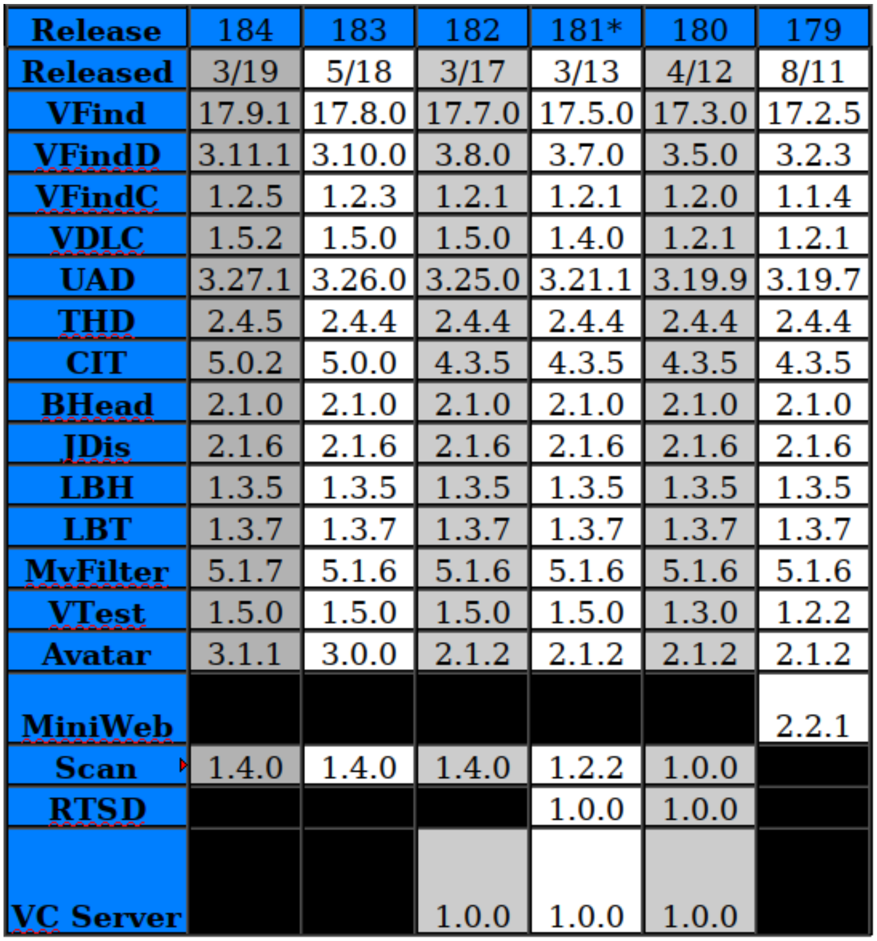
There are currently two packaged flavors of the VFind Security Toolkit(s). They are:

VSTK VFind Security Tool Kit (Standard Edition)
VSTK-T VFind Security Tool Kit Turbo

Version Numbers

It is necessary to explain the version number system used by the VSTK product line. Version numbers are simple to understand. Each tool has its own version number. Version numbers are increased every time a change is made to a tool. The version numbers of every tool is different because the tools have been introduced at different times and have different development schedules. Older tools will have higher version numbers because of enhancements made over the course of years. VFind was at Version 1.00 in 1990 and is currently at Version 17.7.0 in March 2017. Tools are not released individually. They are only released as a toolkit; consequently, each toolkit needs a version number. Tool Kits have two part release numbers. They are release number followed by the letter "u" and the update package number. If there has not been any updates, then an update number will not be provided. As of March 2017 the latest version is VSTK 182. Please note that this is Version 182 (One Hundred Eighty Two) and not Version 1.8.2. The product has been in continuous development and sale since 1990 and so has a high version number. Older versions of the products will continue to use the old release number but the update number will be incremented each time a patch is provided.

Table of VSTK Version Numbers and Release Dates

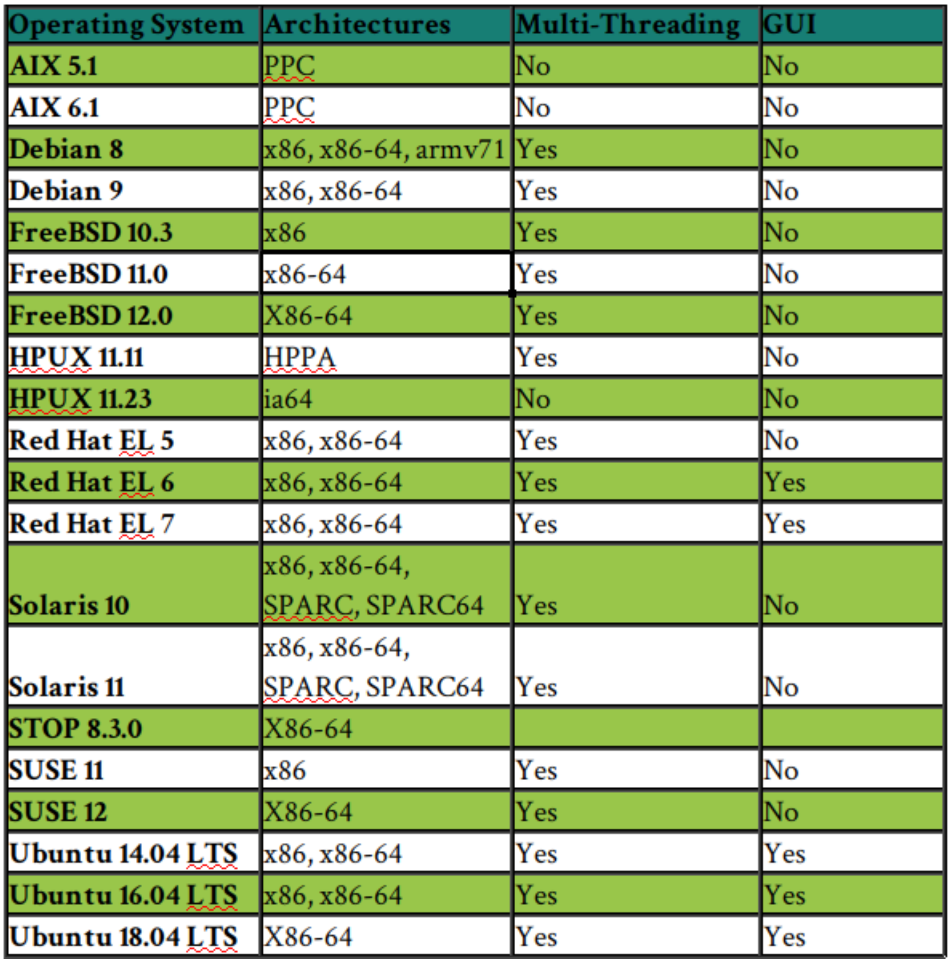


Release	184	183	182	181*	180	179
Released	3/19	5/18	3/17	3/13	4/12	8/11
VFind	17.9.1	17.8.0	17.7.0	17.5.0	17.3.0	17.2.5
VFindD	3.11.1	3.10.0	3.8.0	3.7.0	3.5.0	3.2.3
VFindC	1.2.5	1.2.3	1.2.1	1.2.1	1.2.0	1.1.4
VDLC	1.5.2	1.5.0	1.5.0	1.4.0	1.2.1	1.2.1
UAD	3.27.1	3.26.0	3.25.0	3.21.1	3.19.9	3.19.7
THD	2.4.5	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4
CIT	5.0.2	5.0.0	4.3.5	4.3.5	4.3.5	4.3.5
BHead	2.1.0	2.1.0	2.1.0	2.1.0	2.1.0	2.1.0
JDis	2.1.6	2.1.6	2.1.6	2.1.6	2.1.6	2.1.6
LBH	1.3.5	1.3.5	1.3.5	1.3.5	1.3.5	1.3.5
LBT	1.3.7	1.3.7	1.3.7	1.3.7	1.3.7	1.3.7
MvFilter	5.1.7	5.1.6	5.1.6	5.1.6	5.1.6	5.1.6
VTest	1.5.0	1.5.0	1.5.0	1.5.0	1.3.0	1.2.2
Avatar	3.1.1	3.0.0	2.1.2	2.1.2	2.1.2	2.1.2
MiniWeb						2.2.1
Scan	1.4.0	1.4.0	1.4.0	1.2.2	1.0.0	
RTSD				1.0.0	1.0.0	
VC Server			1.0.0	1.0.0	1.0.0	

After Release 181, CyberSoft switched to implementing an incremental update system in response to customers' requests. This means that while new major releases were not pushed as frequently, incremental updates were instead pushed to more easily allow customers to operate within the confines of their given policies.

Please note that the minor updates only contain bug fixes, patches, and other incremental changes. Major releases contain new features and breaking changes.

Table of Supported Operating System Compatibility, Version 184 Subset



Operating System	Architectures	Multi-Threading	GUI
AIX 5.1	PPC	No	No
AIX 6.1	PPC	No	No
Debian 8	x86, x86-64, armv71	Yes	No
Debian 9	x86, x86-64	Yes	No
FreeBSD 10.3	x86	Yes	No
FreeBSD 11.0	x86-64	Yes	No
FreeBSD 12.0	X86-64	Yes	No
HPUX 11.11	HPPA	Yes	No
HPUX 11.23	ia64	No	No
Red Hat EL 5	x86, x86-64	Yes	No
Red Hat EL 6	x86, x86-64	Yes	Yes
Red Hat EL 7	x86, x86-64	Yes	Yes
Solaris 10	x86, x86-64, SPARC, SPARC64	Yes	No
Solaris 11	x86, x86-64, SPARC, SPARC64	Yes	No
STOP 8.3.0	X86-64		
SUSE 11	x86	Yes	No
SUSE 12	X86-64	Yes	No
Ubuntu 14.04 LTS	x86, x86-64	Yes	Yes
Ubuntu 16.04 LTS	x86, x86-64	Yes	Yes
Ubuntu 18.04 LTS	X86-64	Yes	Yes

CyberSoft continually updates this information as new technologies are released and others are deprecated. CyberSoft will port to any system for the cost of the system. If the OS software is free and will run under a virtual machine, then there is no charge. There are more systems supported than is shown on this chart. If there is a question about system support please call.

www.cybersoft.com

The primary support page for CyberSoft Operating Corporation and the VSTK family of products is www.cybersoft.com. FAQs, training manuals and videos, pricing and example scripts can be found here.

my.cybersoft.com

The my.cybersoft.com website is restricted to customers by userid and password. To apply for a my.cybersoft account contact CyberSoft by phone or email and request an account. Proof of purchase will be required at the time of the request.

This portal will allow the user to:

- Manage and generate VSTK family of products licenses.
- Obtain VSTK daily virus definitions manually. (An automated script is delivered.)
- Download current and past versions of the VSTK software suites.
- Download additional scripts and support downloads to help the user keep CyberSoft products running as smoothly and efficiently as possible.

Please note that all screen shots are as of April 2017 and are subject to change without notice.

New User Overview

When products are purchased from CyberSoft either directly or via a reseller, an invoice is created. The invoice number becomes the name of a **Maintenance Account**. The Maintenance Account owns the activation **Keys**, product downloads and product update downloads. To access the Maintenance Account a **User Account** must first be applied for and approved. At the request of an authorized user a User Account is associated with a Maintenance Account. An authorized user can do Initial Program Load (IPL), key generation, software upgrade downloads, and daily update downloads.

A User Account can be requested at https://my.cybersoft.com/signup/. A name, company email address, company name, **Invoice Number**, and phone number will be required to apply. The invoice number will be given upon purchase of a CyberSoft product. A user account is not automatically granted, and users will be verified as appropriately authorized for access to an account. This process is conducted as a countermeasure to fake or erroneous user account requests. If an error is made during the application process, the request will likely be rejected by the system. In the event of a rejected request, please contact CyberSoft support directly to resolve the issue either by phone (610-825-6785) or email (support@cyber.com).

Either the purchaser, a point of contact listed on the purchase order, or another previously authorized user must approve User and Maintenance Accounts. Once a User Account is approved and associated with one or more Maintenance Accounts, software activation keys can be created, software package(s) can be downloaded, and **Daily Updates** can be downloaded.

In the CyberSoft Portal System, Maintenance Accounts own **Licenses** (keys), which have an **Expiration Date** that determines at what time the Maintenance Account is no longer accessible. A Maintenance Account is always named after an Invoice Number which ties it into the accounting system. The Maintenance Account also owns the ability to download daily updates and **Software Upgrades**. The portal allows for automated download of daily updates via a script that is delivered with the product.

Permanent keys mean that the software does not shut off, however they cannot access maintenance after the first year unless additional maintenance is purchased. Although the software will continue to function after maintenance has expired, it will no longer have access to daily updates, which jeopardizes the security of the system, with decreased risk to detect hostile programs resulting in increased security vulnerability. The daily updates provide detection for the most recent threats. The pace in the real world of hostile code moves at a steep incline, and daily updates are the means to keep pace with the incline of hostile code.

Dictionary of CyberSoft Terms

User Account - A role assigned to individual(s). Customers may have as many user accounts as desired. A User Account may be associated with multiple Maintenance Accounts. User Accounts never expire but can be deleted. Users have to apply for an account at the portal.

Maintenance Account - A Maintenance Account uses a CyberSoft Invoice Number as the identifier, which means that Maintenance Account names will be a series of numbers. This provides traceability to the purchase and therefore confirms ownership. Maintenance Accounts own licenses (keys) and are associated with User Accounts. Maintenance Accounts expire since they are purchased for a specific period of time. Some customers purchase more than one year of maintenance in advance. An authorized user, the initial buyer, or other Point Of Contact(s) listed on the Purchase Order must approve association of a User Account to a Maintenance Account. Who authorized what is logged.

Invoice Number - An identifier that associates a Maintenance Account to the accounting system. This is always a number and is used as the Maintenance Account name. Having the Invoice Number does not provide access to the invoice. Only the purchaser is able to see the invoice. In the case of a reseller, the purchaser is the reseller, not the end user.

Key/License - A software activation key is generated by a Maintenance Account at the request of an authorized User. A Key activates software at the end user site. Most keys never expire.

Daily Update - This is usually text data written in the CVDL language that is used by VFind to enhance detection of real world hostile programs.

Software Upgrade - The entire package of software used either to upgrade an installed product or as initial program load.

Expiration Date - Maintenance Accounts expire. Software Keys rarely expire unless specifically purchased that way. Maintenance Access is purchased for a specific period of time. Before the account expires it will automatically generate warning messages to the authorized users for that Maintenance Account. Maintenance Accounts are extended via a purchase of additional time. Time can be purchased in months or years.

How to Get Help

There are multiple ways to get help from CyberSoft.

Phone: +1 (610)825-4748 (9-5 Eastern Time, weekdays - voicemail off hours)
Fax: +1 (610)825-6785

General Email: support@cyber.com
Documentation: www.cybersoft.com
Help Ticket System: www.cybersoft.com/support/ (click on “Create A Help Ticket”)

VSTK user documentation is available online free and professionally printed for a cost from CyberSoft. Contact us for details.

When requesting support from CyberSoft, the maintenance account number is not necessary, but does speed up the assistance process.

VSTK Install Process

This information is subject to change without notice. The latest version of this information can be found in the installation media in the file install.txt.

===

UNIX Installation Instructions for VFind Security ToolKit

===

1. Insert the CD-ROM into the CD-ROM drive or unpack the downloaded installation package.

2. For CD-ROM, if the system does not have an "auto-mounting" daemon, then use the 'mount' command to mount the CD. (please see mount in the UNIX system manual) for example:

```
$ mount -t iso9660 /dev/cdrom /mnt/cdrom
```

(Solaris example)

3. Once the CD is mounted, use the 'cd' command to change the current directory to that of the cdrom at its top level. For example:

```
$ cd /mnt/cdrom
```

(Solaris example)

4. Use the 'ls' command to list the contents of the directory. Something similar to this should be seen:

```
INSTALL.txt
archinstall.sh
install_toolsindex.html
noarch
README.txt
```

5. Read the file noarch/EULA.txt. This is a copy of the the user's right to use the license.

6. Proceeding with this step means that the user agrees to the terms in noarch/EULA.txt. Making sure that the user is in the root directory of the CD-ROM, run the ‘install.sh' script. It may be necessary to explicitly invoke sh when running install.sh: /bin/sh install.sh

7. Follow the instructions in the install script.

8. After installation, the only thing remaining is to setup a valid license key for the system. If supplied with a Permanent Activation Key, or a Demo Key, then the user must create a file called LICENSE, copy and paste the Permanent/Temporary Key into the file, and copy that file into the VSTK base installation directory that was specified during the install process.

9. If not supplied with a Permanent or Temporary Activation Key, please contact CyberSoft. Contact information is provided at the end of this document.

10. Congratulations! VSTK is now installed!

11. CyberSoft also recommends signing up for a support and maintenance contract, which allows the user to keep their protection up-to-date through daily downloads of updated virus definition patterns. Contact information is provided at the end of this document.

12. For information on how to use the VSTK product and for recommended uses of the VSTK tools, please look at the example scripts included with the product and/or view the Support and White Papers sections on the CyberSoft web site.

For further assistance, please contact CyberSoft. Contact information is provided at the end of this document.

Contents of the Distribution

Man pages for these are installed in the man directory below the VSTK_HOME directory specified during installation. This directory should be added to the MAN_PATH environment variable. They then can typically be read by doing, for example, 'man VFind'.

In the VSTK_HOME directory specified during installation there are a number of additional files. Important ones include:

bin/Directory containing wrapper shell scripts. These shell scripts set up needed environment variables for executing the binary executables.

doc/html/html - versions of all man pages
doc/txt/text - versions of all man pages
doc/postscript/postscript - versions of all man pages
man/man1/troff - versions of application man pages
man/man5/troff - versions of protocol man pages (Turbo only)
LICENSE - an example license file
EULA.txt - Right to Use, License Agreement
programs/ - Directory containing the binary executables.
example_scripts/ - Directory containing example scripts

In order to access CyberSoft's virus definition updates (VDLs), the user will need to sign up for the new VDL update system at https://my.cybersoft.com/signup

This service will provide the user with:

Easy-to-find support date: The new VDL update system shows the date when the user's maintenance will expire. The user's maintenance contract must be current to access the VDLs.

Easy-to-find help: The user's new personalized VDL update page has links to training manuals and white papers.

Automatic updates: If configured, the vdlupdate cron job (part of the installation procedure), will keep the user's VDLs always up-to-date.

Thank You!

VSTK Supported Platforms and License Information

The VSTK ToolKits are supported on almost every version of Unix and Linux in current use and many that are no longer in common use. CyberSoft will port to and support any platform for the cost of the system. Many customers provide specialized systems to CyberSoft for this purpose.

VSTK licenses are per system. The cost of the product is flat and the optional yearly maintenance and upgrade is also a flat cost per system. The first year of maintenance and upgrades via the Internet is free.

The deliverable VSTK media contains almost all supported platforms. The customer can install on any system for which they have a license from a single deliverable. Activation keys are text based and are obtained from CyberSoft.

1. Per system pricing. No "per user" costs. Great for servers and email systems. The price is predictable and fixed.

2. No CPU or size costs. Add several CPUs, faster CPUs, no difference in price.

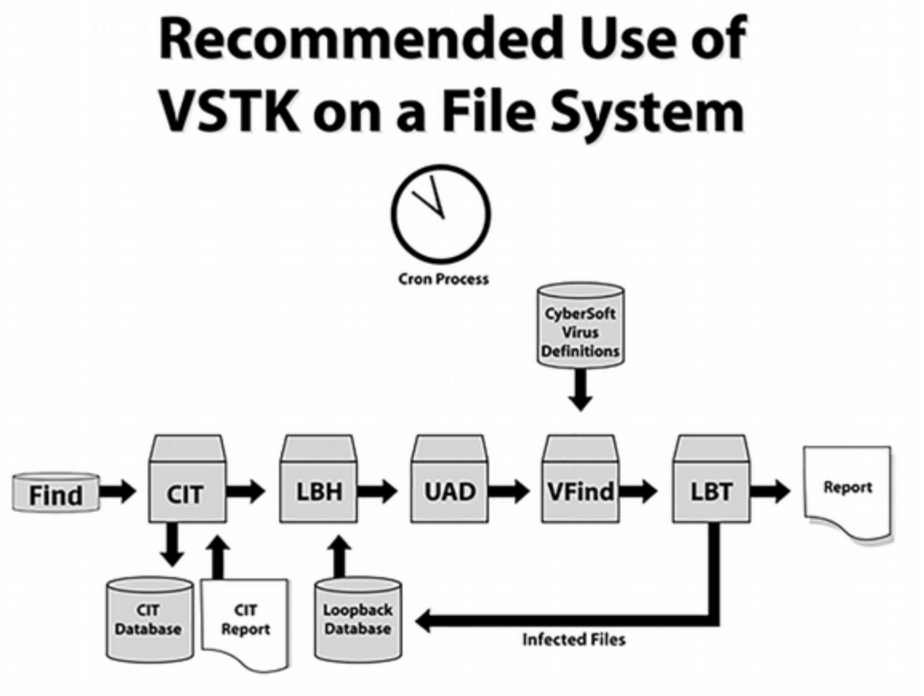
3. No system upgrade charges. Move from a small system to a big system. No difference in cost.

4. Platform independence. CyberSoft supports just about every Unix system and several Microsoft systems. CyberSoft will support any system for the cost of the system. As long as the nodename is kept the same and the license is not violated, the license can be moved from any platform to any platform without permission from CyberSoft. Examples: from/to Sun Micro, HP, IBM, SGI, Linux, BSD, etc.

5. VSTK products are Perpetual license, while VSTK-L products are only for the specified time. Most VSTK licenses never expire with the exception of Special products which are time based licenses. Maintenance can be purchased every year for VSTK licenses.

6. Maintenance support is all inclusive, and includes maintenance, upgrades and telephone support. Maintenance via Internet is free for the first year.

Graphic - Recommended Use of VSTK on a File System



VDLUpdate

VDLUpdate is a Unix shell script supplied with all VSTK ToolKits. It's purpose is to give the user an extremely easy manual or automatic means to update their virus definitions. When used from the Unix command line, the user can manually update their definitions. When used from the Unix cron system, the script runs automatically at the frequency and time set by the user. A file locking mechanism is employed to prevent updating of the VDL database while it is currently being used by VFind.

VDLUpdate uses port 443 which is the web HTTPS SSL port. This is the port that all Internet browsers use when communicating with web servers that provide SSL encryption. If the system can access an SSL encrypted website, it can use VdlUpdate without changing any firewall settings.

VDLUpdate only downloads VDLs if there has been a VDL change since the user's last run. This is done by supplying md5 hash codes for the VDL database. If the system's local hash code does not match the hash code on the CyberSoft server, an update is required. After the VDLs are downloaded, they are run against VFind as a final verification check. If all goes well, the new VDLs are accepted. As part of the installation of the VSTK toolkit, the user is given an option of configuring VDLUpdate for automatic execution. By choosing this option, the user is assured that their VDL database is up-to-date as of the time of execution. Users are urged to automate the execution of VDLUpdate and to run it often.

Example Script Notes

The following is a list of the example shell scripts included with the VSTK product line and a brief description of what function each script performs. These scripts will enable the user to quickly become familiar with the programs and tools included in the VFind Security ToolKit. The scripts require customization to the user's local standards prior to implementation. Aids are provided for that purpose. Subject to change without notice.

These scripts are fully functional and have been tested on a Solaris SPARC machine. The scripts should be reviewed and edited prior to using to determine platform and system specific requirements. Ensure that there is sufficient temporary disk space when using UAD as UAD uses the temporary space for expanding archive files. The UAD option '-t' enables the user to specify a temporary directory if the default, /tmp, does not have sufficient available space. UAD also supports the '--recursion' option to limit the levels of recursion when expanding archive files. In the example scripts, the recursion limit is set to 25, which may not be suitable for the user's system and resident archive files. CIT requires disk space for the CIT database (cit.db), report (cit.rpt), and interim work files. The CIT option '--dbpath' enables the user to specify the directory to store CIT files and to locate the cit_danger.db file. The default location for the CIT files is the current directory.

Script Description

cit.sh
Runs the Cryptographic Integrity Tool (CIT) to generate an MD5 signature string for every regular file on the system and stores the signatures in the CIT database file (cit.db). CIT generates a report after each run listing the files that have been added, deleted, or modified since the previous run.

uad.sh
Runs the Universal Atomic Disintegrator (UAD) utility to identify the file type of each regular file on the system and to expand compressed archive files and identify and process its sub-files.

uad_do.sh
Uses the UAD utility to expand a specified archive file. The resultant files are written to the current directory.

vfind.sh
Runs the VFind virus scanner against all the regular files on system to identify infected files. VFind output lists all the files that have been scanned and identifies which files are infected and which virus infected the file.

cit_uad_vfind.sh
Pipes all the regular files on the system through the CIT, UAD, and VFind utilities. CIT generates a database of signatures, UAD expands and types the files, and VFind scans for viruses. On subsequent runs, cit uses the data of signatures to filter unchanged files and passes only new or modified files to UAD and find. The SmartScan protocol is used by UAD to pass file type and file contents to VFind.

cit_uad_vfind_lb.sh
Pipes all the regular files on the system through CIT, LoopBack Head, UAD, VFind, LoopBack Tail utilities. The LoopBack Head and LoopBack Tail processes keep track of files that VFind has tagged as infected, and ensures that they continue to be scanned until they are replaced or removed from the system.

uad_Vfind_grep.sh
Pipes all the regular files on the system through uad and VFind. uad uses the Smartscan protocol to pass file type and file contents to VFind. The VFind output is filtered so that only the data concerning infected files is presented.

vfchmod.sh
Isolates all virus infected files on a UNIX system by setting their permission bits to zero and sending a report to the "root" user. See the man page for the command "chmod" for details.

vfmv.sh
Moves files suspected of being infected with a computer virus to a directory named ./QUARANTINE.

vfrm.sh

Removes from the system files that have been tagged as infected by VFind. The user is prompted before a file is removed. This is not recommended for whole system scans.

vfstab.sh

Scans the files that are referenced in the /etc/exports. i.e., those local file systems that are available to be mounted on remote systems.

boot.sh

Uses the BHead utility to read a boot sector on a PC-based Unix system and passing the data to VFind to be scanned for viruses. BHead allows the user to check the boot sector without having to process the entire disk drive. The user must edit the script and provide the name of the raw device and the blocking factor before running the script.

thd.sh

Uses Trojan Horse Detector (THD) to check every regular file on the system. THD output identifies potential problem files.

mvfilter.sh

Uses the MVFilter utility to strip VBA Macros from the specified file.

cron_VFind.sh

These scripts create cron entries for running VFind cron_vdlupdate.sh and for updating the VDL files.

uad_external

This directory of files comprises the interface for incorporating an external file expander into the UAD utility.

wget.sh

Uses wget to download files and UAD|VFind to scan the downloaded files. Infected files are saved and the others are removed.

cs_support.sh

This script can be used to collect data that would aid CyberSoft engineering in troubleshooting reported problems.

boot_CIT.sh

This script shows how user can use CIT to detect changes in boot sector.

cit_uad_report_VFind.sh

This script shows a simple tip how user can get UAD's output as a separated file when CIT, UAD, VFind are connected with SmartScan protocol.

uad_vfind_noscan.sh

This script is an upgraded version of existing uad_vfind.sh example-script especially for excluding a specific mount point from scanning. Unlike the uad_vfind.sh, this new script uses _config_noscan.sh instead of _config.sh. This script is also helpful if existing scripts (excluder, Pl, noscans.sh) do not properly work on the system's specific shell.

Example Script For VDL Update

```
#!/bin/sh
#
# Name: cron_vdlupdate.sh
#
# Copyright 2003, 2004, 2005, 2006 CyberSoft Operating Corporation, All Rights Reserved.
#
umask 022
PATH="/bin:/usr/bin:/sbin:/usr/sbin:$PATH"
export PATH

# every 4 hours, randomly
# random minute, 0-59
#
x=`expr "$$" % 4`
h1=`expr "$x" + 0`
h2=`expr "$x" + 4`
h3=`expr "$x" + 8`
h4=`expr "$x" + 12`
h5=`expr "$x" + 16`
h6=`expr "$x" + 20`
h="$h1,$h2,$h3,$h4,$h5,$h6"
m=`expr "$$" % 60`

VSTK_HOME=
vdlupdate="$m $h * * * $VSTK_HOME/bin/vdlupdate >/dev/null"

tmp="/tmp/cron.tmp.$$"
rm -f "$tmp"
crontab -l 2>/dev/null | grep -v "$VSTK_HOME/bin/vdlupdate" > "$tmp"
echo "$vdlupdate" >> "$tmp"

echo "setting cron vdlupdate run:"
echo "$vdlupdate"

crontab "$tmp"
rm -f "$tmp"
```

Arguments

The example scripts provided only use the most common options with each of the tools, so please read the documentation provided, consult the CyberSoft website www.cybersoft.com or use the -h option with each of the tools to receive a listing of the available options. More options can be passed to the tools from the command line with the following options:

For those scripts that use find:
-p <path> (where to start the find from - default is /)

For those scripts that use VFind:
-m <number of threads> (use multi-threaded version of VFind with specified number of threads)
-v <VFind options> (other arguments to pass to VFind)

For those scripts that use CIT:
-c <cit options> (other arguments to pass to cit)

ipts that use UAD:
-u <uad options> (other arguments to pass to uad)

Also, the VSTK_HOME environment variable in the example scripts is set to the directory identified as the VSTK home directory during installation.

Disclaimer

These scripts, programs, and patches are only examples, and may need to be modified to suit the user's requirements before they are used.

EICAR - Virus String Test System

European Institute for Computer Antivirus Research

The EICAR Antivirus test file is used for determining if an Antivirus product will sufficiently detect viruses. This test file is not a real virus and it is only used for testing the effectiveness of Antivirus products. When testing please use UAD with VFind.

EICAR Version 2 (Current Version)

The EICAR Version 2 test suite was made necessary when some viruses started to use the EICAR test string as camouflage. Many antivirus products would detect the EICAR string and not report the virus. This was never a problem with VFind which reported both the virus and the EICAR test string. The problem was corrected when the EICAR changed the test standard so that the test would only be positive if the string was exact and did not contain any additional bytes of data either before or after the test string. According to the standard, any antivirus product which detects the EICAR test string as a subset of a byte stream of data would fail the test - this test suite conforms to the standard. Please read the Version 2 EICAR like test suite below.

eicar.com

```
X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
```

eicar.vdl

```
:old-style EICAR TEST -- Fake Virus -- IGNORE, "\x58\x35\x4f\x21\x50\x25\x40\x41\x50\x5b
\x34\x5c\x50\x5a\x58\x35\x34\x28\x50\x5e\x29\x37\x43\x43\x29\x37\x7d\x24\x45\x49\x43\x41
\x52\x2d\x53\x54\x41\x4e\x44\x41\x52\x44\x2d\x41\x4e\x54\x49\x56\x49\x52\x55\x53\x2d\x54
\x45\x53\x54\x2d\x46\x49\x4c\x45\x21\x24\x48\x2b\x48\x2a" #

:new-style EICAR TEST -- Fake Virus -- IGNORE, ABS 0, "\x58\x35\x4f\x21\x50\x25\x40\x41
\x50\x5b\x34\x5c\x50\x5a\x58\x35\x34\x28\x50\x5e\x29\x37\x43\x43\x29\x37\x7d\x24\x45\x49
\x43\x41\x52\x2d\x53\x54\x41\x4e\x44\x41\x52\x44\x2d\x41\x4e\x54\x49\x56\x49\x52\x55\x53
\x2d\x54\x45\x53\x54\x2d\x46\x49\x4c\x45\x21\x24\x48\x2b\x48\x2a", EOD #
```

EICAR Version 2 (Non Standard)

This version does not conform to the EICAR test standard. It will detect the test string even as a subset of a larger byte stream. This test suite is the same as the obsolete Version 1 test suite.

eicar2 nonstd.vdl

```
;; $Id: eicar.vdl,v 1.1.1.1 2002/11/15 03:34:41 perry Exp $
;; COPYRIGHT 2002 BY CYBERSOFT, INC.  ALL RIGHTS RESERVED.
:EICAR TEST Fake Virus - IGNORE,"\x58\x35\x4f\x21\x50\x25\x40\x41\x50\x5b\x34\x5c\x50
\x5a\x58\x35\x34\x28\x50\x5e\x29\x37\x43\x43\x29\x37\x7d\x24\x45\x49\x43\x41\x52\x2d\x53
\x54\x41\x4e\x44\x41\x52\x44\x2d\x41\x4e\x54\x49\x56\x49\x52\x55\x53\x2d\x54\x45\x53\x54
\x2d\x46\x49\x4c\x45\x21\x24\x48\x2b\x48\x2a"#
```

VSTK FAMILY OF PRODUCTS
End User License Agreement

CyberSoft Operating Corporation
End User License Agreement

IMPORTANT - - READ CAREFULLY:

This End User License Agreement ("License Agreement") is a legally enforceable agreement between you (either an individual or a single entity) ("you" or "LICENSEE") and CyberSoft Operating Corporation ("CyberSoft"), regarding the use of the Licensed Software (as defined below) included or supplied with or on, or incorporated into, this media and/or downloaded by or otherwise provided to you in electronic form or otherwise. By installing or otherwise using the Licensed Software, you agree to be bound by the terms of this License Agreement. If you do not agree to the terms of this License Agreement, do not install or use the Licensed Software; however, you may return it to CyberSoft or the third party distributor, dealer or supplier ("Distributor") from whom you purchased the Licensed Software for a full refund.

DEFINITIONS:

"Embedded Programs" means all third party software, modules, products, interfaces, data files and other files and programs provided by CyberSoft as part of or in connection with its proprietary software.

"Invoice" means the purchase order, invoice or other documentation pursuant to which you licensed the Licensed Software.

"Licensed Software" means (i) CyberSoft's proprietary software, which may include but is not limited to VFind Security Tool Kit Standard Edition (VSTK), VFind Security Tool Kit Turbo (VSTK-T), VFind Security Tool Kit Lite (VSTK-L), VSTK for Windows (VSTK-W), Avatar (Avatar), VFind Security Tool Kit Standard Special (VSTK-SS), VFind Security Tool Kit Turbo Special (VSTK-TS), VFind Security Tool Kit Lite Special (VSTK-LS), VSTK for Windows Special (VSTK-WS), Avatar Special (Avatar-S), VFind VStick SysAdmin (VSTICK-S), VFind VStick Financial (VSTICK-F) and/or other CyberSoft-branded products, as applicable, (ii) the Embedded Programs, (iii) the Updates, and (iv) all documentation, instructions, manuals, diagrams and other materials, in whatever medium or format, pertaining to the foregoing.

"System" means a single computer system with one or more cpu processors sharing a common motherboard or backplane (but in no event shall a network constitute a motherboard or backplane hereunder).

"Updates" means any virus definition updates, hash codes and/or product updates/upgrades included in the Support Services as solely determined by CyberSoft.

"Users" means all persons or entities gaining access to the Licensed Software by or through you and/or any of your Systems.

"Maintenance" means Upgrades that are included in the Support Services as solely determined by CyberSoft.

"Support" means a reasonable amount of contact by email, postal mail, courier delivery, in-person, fax, telephone or other communications medium. Reasonable shall be determined solely by CyberSoft. CyberSoft reserves the right to discontinue support to any individual, without cause or notice. In addition, CyberSoft solely reserves the right to limit the number of individuals who may contact CyberSoft for Support.

"Modified Support" means a reasonable, limited and restricted amount of contact by email, postal mail, courier delivery, in-person, fax, telephone or other communications medium. Reasonable shall be determined solely by CyberSoft. Limited is defined as 30 calendar days from the date of the CyberSoft Invoice. After the 30 calendar day period Modified-Support is only available by means of email and only for the term of the contact as determined by the CyberSoft Invoice. Verbal communications is not available after the initial 30-day period. Restricted is defined as topic restrictions as determined solely by CyberSoft on a case-by-case basis. Determination of Restricted on one case does not influence the determination of Restricted on any other case and can be modified without notice by CyberSoft. CyberSoft reserves the right to discontinue support to any individual, without cause or notice. In addition, CyberSoft solely reserves the right to limit the number of individuals who may contact CyberSoft for Modified-Support.

LICENSE GRANTED:

Subject to the conditions herein, CyberSoft hereby grants to you, as LICENSEE, a non-transferable, non-exclusive, limited license to use the Licensed Software in machine-readable form on one System (the "License") located at the site, if any, designated in the Invoice (the "Designated Site"). The License granted herein is subject to all further restrictions as set forth in the Invoice. You shall cause all Users to comply with the terms of this License Agreement and shall be liable for all acts and omissions of such Users. No license, right or interest in any trademark, trade name or service mark of CyberSoft is granted under this License.

You may use the Licensed Software solely for your own internal use. Any other use of the Licensed Software (including without limitation timesharing, rental, facility management or service bureau usage) is strictly prohibited.

Except as otherwise set forth in the Invoice or as agreed by CyberSoft in writing, you may not sell, lease, assign, sublicense or otherwise transfer, in whole or in part, this License Agreement, the License, the Licensed Software and other CyberSoft-provided materials, or any licenses or rights granted hereunder; provided, however, that (a) if the Licensed Software is licensed for the express purpose of integration into a larger system, Licensee is permitted to sublicense the Licensed Software solely as part of such system (and not as a stand-alone product) subject to all other terms and conditions hereof, (b) if Licensee is a dealer or reseller of software and is licensing the Licensed Software for the express purpose of resale, Licensee may sublicense the Licensed Software to its customers subject to all other terms and conditions hereof (including, without limitation, that such customers may not sublicense the Licensed Software), and (c) transfer to the United States Government as a designated end user when part of a turnkey system is expressly permitted, subject to Section 12 hereof on U.S. Government Restricted Rights.

The License granted hereunder is for the object code version of the Licensed Software only. You shall not, and shall not permit anyone under your direction or control to, reverse engineer, disassemble or de-compile the Licensed Software or attempt to do so. LICENSEE may not modify, adapt, translate or create derivative works of the Licensed Software without CyberSoft's express written consent; provided, however, that if the Licensed Software is licensed for the express purpose of integration into a larger system, Licensee is permitted to do so. The Licensed Software is licensed as a single product. Embedded Programs may be used only for purposes of (i) running the Licensed Software, and (ii) extracting data from the Licensed Software for use with other programs. Embedded Programs may not be used for purposes of application development, modification or customization, or running programs other than the Licensed Software.

LICENSEE may make a reasonable number of copies of the Licensed Software for backup and archival purposes only, and must include on all copies of the Licensed Software all copyright, government restricted rights and other proprietary notices or legends included on or in the Licensed Software as provided to LICENSEE.

Notwithstanding anything to the contrary herein, LICENSEE's access to and use of the Embedded Programs shall be and remain subject to all third party and/or open source licenses applicable to such Embedded Programs, and LICENSEE shall comply with same.

LICENSEE shall be solely responsible for ensuring that performance of its obligations and exercise of its rights (including without limitation its use of the Licensed Software) under this License Agreement comply with all applicable federal, state, local and international laws, rules, regulations and orders (collectively, "Laws"), including without limitation all present and future laws and regulations relating to protection of intellectual property and privacy.

DELIVERY, INSTALLATION, ACCEPTANCE AND RISK OF LOSS:

If CyberSoft delivers the Licensed Software on CD-ROM or other tangible media, (i) CyberSoft shall deliver the Licensed Software to a common carrier, (ii) LICENSEE assumes all risk of loss or damage upon delivery of the Licensed Software by CyberSoft to the common carrier, and (iii) acceptance shall occur upon delivery of the Licensed Software by CyberSoft to the common carrier. In all other cases, acceptance shall occur immediately upon installation of the Licensed Software by LICENSEE. LICENSEE shall be solely responsible for installation of the Licensed Software.

SUPPORT SERVICES:

Limited to the products identified with the part numbers VSTK, VSTK-T, VSTK-L, VSTK-W, and AVATAR, for a period of twelve (12) months following the Invoice date, CyberSoft will provide to you, free of charge, Maintenance and Support services related to the Licensed Software as further described at www.cybersoft.com/purchase/prices ("Support Services"). Licensees desiring extended Support Services may purchase same at the www.cybersoft.com. All terms and conditions of this License Agreement will apply.

Limited to the products identified with the part numbers VSTK-SS, VSTK-LS, VSTK-WS, VSTK-TS, AVATAR-S, VSTICK-S and VSTICK-F, for a period of time specified in the CyberSoft Invoice, CyberSoft will provide to you, free of charge, Maintenance and Modified-Support services related to the Licensed Software as further described at www.cybersoft.com/purchase/prices ("Support Services"). In no extent will free services exceed twelve (12) months following the Invoice date. Licensees desiring extended Support Services may purchase same at the www.cybersoft.com. All terms and conditions of this License Agreement will apply.

LICENSEE shall be solely responsible for providing all necessary hardware and software to run the Licensed Software and to obtain the Support Services. All telephone and connection arrangements and charges from LICENSEE's facility to CyberSoft shall be LICENSEE's responsibility.

Except as otherwise provided in Section 8.1 hereof with respect to the Updates, all Support Services are provided on an "as is", "where is" basis, and CyberSoft makes no warranties with respect to such Support Services.

CyberSoft may, in its sole discretion, require LICENSEE to purchase and/or install certain Updates within a specified time period after their release, and may modify, suspend or terminate the Support Services, in whole or in part, upon LICENSEE's failure to timely do so, without liability or obligation to LICENSEE. Nothing herein shall be construed as requiring CyberSoft to support more than the most recent release of the Licensed Software.

Any technical information you provide to CyberSoft as part of the Support Services may be used by CyberSoft for product support and development; provided, however, that CyberSoft will not use such technical information in a form that personally identifies you.

By providing utility or other programs created by or for you to operate in conjunction with the Licensed Software, you grant CyberSoft a perpetual, non-exclusive, irrevocable, worldwide, transferable, royal-free, fully paid-up license (with right to sublicense) to use, reproduce, distribute, display, perform, create derivative works of, and modify the source code and executable code versions of such programs.

TERM AND TERMINATION:

Limited to the products identified with the part numbers VSTK, VSTK-T, VSTK-L, VSTK-W, and AVATAR, the effective date of this License shall be the date of the CyberSoft Invoice date or the date of the initial use, whichever is earlier, of the Licensed Software and its term is perpetual, subject to the termination provisions of this Section 5. Limited to the products identified with the part numbers VSTK-SS, VSTK-LS, VSTK-WS, VSTK-TS, AVATAR-S, VSTICK-S and VSTICK-F, the effective date of this License shall be the CyberSoft Invoice date or the date of the initial use, whichever is earlier, of the Licensed Software and its term is as specified on the CyberSoft Invoice, subject to the termination provisions of this Section 5. If no term is specified in the CyberSoft Invoice then the term shall be twelve (12) months from the CyberSoft Invoice date.

This License will terminate automatically upon your failure to comply with any term of this License Agreement. Upon termination of this License, you shall discontinue all use of the Licensed Software. In such event, the License and rights granted hereunder shall expire and you shall have no further rights or access to the Licensed Software. CyberSoft may terminate provision of the Support Services at any time, without notice to you, if it ceases to provide such services to its licensees generally.

PAYMENTS:

In consideration of the License and other rights granted by CyberSoft hereunder, you agree to pay the license fee specified in the Invoice and/or otherwise imposed by CyberSoft or any Distributor in connection herewith ("License Fees"). Unless otherwise provided in the Invoice, LICENSEE shall pay all License Fees within thirty (30) days after the date of CyberSoft's or Distributor's invoice therefor.

All payments shall be made in U.S. currency. Any sum not paid by LICENSEE when due shall bear interest until paid at a rate of 1.5% per month (18% per annum), or the maximum rate permitted by law, whichever is less. LICENSEE shall be responsible for the costs, including without limitation, reasonable attorneys' fees and court costs, incurred by CyberSoft in connection with CyberSoft's collection of any past-due amounts under this License Agreement. If LICENSEE is the United States Government, FAR 52.232-25PROMPT PAYMENT (OCT.2003) applies and is incorporated herein by reference, in lieu of the forgoing payment provisions of this Section 6.2.

All payments required under Section 6.1 and/or Section 6.2 or otherwise under this License are exclusive of taxes and you agree to bear and be responsible for the payment of all such taxes (except for taxes based upon CyberSoft's income) including, but not limited to, all sales, use, rental receipt, personal property, import and value-added or other taxes which may be levied or assessed in connection with this License.

PROPRIETARY RIGHTS; CONFIDENTIALITY:

CyberSoft Operating Corporation is a licensee of CyberSoft, Inc., the owner of the Licensed Software. As between LICENSEE and CyberSoft, CyberSoft and its licensors (including but not limited to CyberSoft, Inc.) shall be the sole owners of all patent, copyright, trademark, trade secret and other proprietary rights in and to the Licensed Software, and all modifications thereto and derivative works thereof.

The Licensed Software is protected under the copyright laws of the United States and equivalent or similar international laws and treaties, including without limitation, the Berne Convention. CyberSoft and its licensors (including but not limited to CyberSoft, Inc.) may utilize all ideas, suggestions, feedback, improvements, data, reports or the like that LICENSEE provides to CyberSoft with respect to the Licensed Software without any obligation to LICENSEE.

Although copyrighted, the Licensed Software (and Embedded Programs) is unpublished and contains proprietary and confidential information of CyberSoft and its licensors(including but not limited to CyberSoft, Inc.). LICENSEE agrees to maintain the Licensed Software (including Embedded Programs) in confidence and to use a reasonable degree of care to protect the confidentiality of the Licensed Software (and the Embedded Programs).

LIMITED WARRANTY:

CyberSoft warrants the media on which the Licensed Software is provided, if any, to be free from defects in materials and workmanship for ninety (90) days after delivery. Defective media may be returned for replacement without charge during the ninety (90) day warranty period unless the media have been damaged by accident or misuse. In addition, CyberSoft warrants, for ninety (90) days after acceptance, that the unaltered Licensed Software substantially conforms to the documentation that accompanies it (CyberSoft expressly reserves the right to provide the documentation on the same media as the Licensed Software). Any implied warranties not effectively disclaimed pursuant to Section 8.3 hereof are limited to the duration of the express warranties stated in this Section 8.1 NOTWITHSTANDING ANYTHING TO THE CONTRARY HEREIN, THE WARRANTIES SET FORTH IN THIS SECTION 8.1 DO NOT APPLY WITH RESPECT TO FREE, DEMONSTRATION OR ILLEGALLY OBTAINED COPIES OF THE LICENSED SOFTWARE.

CyberSoft's entire liability and your exclusive remedy for breach of the foregoing warranties shall be, at the option of CyberSoft, repair or replacement of the Licensed Software (or the part thereof that does not meet the warranty), when returned to CyberSoft or, if repair or replacement is not reasonably practicable (as determined by CyberSoft in its sole discretion) return of the price paid for such nonconforming portion of the Licensed Software.

Your Invoice must accompany any warranty claim. This limited warranty is void if failure of the Licensed Software has resulted from accident, misuse, misapplication or breach of this License Agreement. Any replacement software will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer. Outside of the United States, neither these remedies nor any Support Services offered by CyberSoft are available without proof of purchase from an authorized international source.

THE EXPRESS LIMITED WARRANTIES SET FORTH IN SECTION 8.1 ARE IN LIEU OF AND, TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CYBERSOFT, ITS DISTRIBUTORS AND LICENSORS (including but not limited to CyberSoft, Inc.) SPECIFICALLY DISCLAIM, ANY AND ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF TITLE, NONINFRINGEMENT, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH REGARD TO THE LICENSED SOFTWARE AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES. Without limiting the generality of the foregoing, none of CyberSoft, its Distributors or licensors (including but not limited to CyberSoft, Inc.) warrant that:(i) operation of any of the Licensed Software shall be uninterrupted or error free, (ii) that functions contained in the Licensed Software shall operate in combinations which may be selected for use by LICENSEE or meet LICENSEE's requirements, or (iii) that the Licensed Software will detect all hacker attacks, viruses, Trojan horses, worms or other software routines or hardware components designed to permit unauthorized access to or to disable, erase or otherwise harm any software, hardware or data, if applicable.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT WILL CYBERSOFT OR ITS DISTRIBUTORS OR LICENSORS (INCLUDING BUT NOT LIMITED TO CYBERSOFT, INC.) BE LIABLE TO LICENSEE, USERS OR ANY THIRD PARTY FOR SPECIAL, INCIDENTAL, INDIRECT, EXEMPLARY, PUNITIVE OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF INCOME, PROFITS, USE OF INFORMATION OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE LICENSED SOFTWARE OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF CYBERSOFT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. CYBERSOFT'S AND ITS DISTRIBUTORS' AND LICENSORS' (including but not limited to CyberSoft, Inc.'s) ENTIRE LIABILITY WITH RESPECT TO THE LICENSED SOFTWARE, THE SUPPORT SERVICES AND THIS LICENSE AGREEMENT SHALL BE LIMITED, IN THE AGGREGATE, TO THE LICENSE FEES ACTUALLY PAID BY LICENSEE FOR THE LICENSED SOFTWARE.

COPYRIGHT AND TRADEMARK INDEMNIFICATION:

CyberSoft will defend, at its expense, any action brought against LICENSEE to the extent that it is based on a claim that the use of the Licensed Software within the scope of the License infringes any United States copyright or trademark. CyberSoft will indemnify LICENSEE from any costs, damages and fees incurred by LICENSEE which are attributable to such claim, provided that LICENSEE notifies CyberSoft promptly in writing of the claim.

LICENSEE shall permit CyberSoft to defend, compromise or settle the claim and provide all available information, assistance and authority to enable CyberSoft to do so, provided CyberSoft reimburses LICENSEE for such activity. LICENSEE shall have no authority to settle any claim on behalf of CyberSoft.

Should the Licensed Software become or in CyberSoft's opinion, be likely to become the subject of a claim of infringement of a U.S. copyright or trademark, CyberSoft may (i) procure for LICENSEE, at no cost to LICENSEE, the right to continue to use the Licensed Software, (ii) replace or modify the Licensed Software, at no cost to LICENSEE, to make same non-infringing, or (iii) if the right to continue to use cannot be procured or the Licensed Software cannot be replaced or modified, terminate the License to use such Licensed Software, remove the Licensed Software, and where a specified License Fee was paid by LICENSEE, grant LICENSEE credit thereon as depreciated on a straight-line three (3) year basis.

CyberSoft shall have no liability for any claim of copyright or trademark infringement based on the (i) use of other than the then latest release of the Licensed Software from CyberSoft, if such infringement could have been avoided by the use of the latest release of the Licensed Software and such latest version had been made available to LICENSEE, (ii) use or combination of the Licensed Software with software, hardware or other materials not provided by CyberSoft, or (iii) modifications to the Licensed Software made by persons other than CyberSoft.

THIS SECTION 9 STATES THE ENTIRE LIABILITY OF CYBERSOFT AND ITS DISTRIBUTORS AND LICENSORS (including but not limited to CyberSoft, Inc.) WITH RESPECT TO INFRINGEMENT BY THE LICENSED SOFTWARE OR ANY PART THEREOF AND NONE OF CYBERSOFT, ITS DISTRIBUTORS OR LICENSORS (including but not limited to CyberSoft, Inc.) SHALL HAVE ANY ADDITIONAL LIABILITY WITH RESPECT TO ANY ALLEGED OR PROVEN INFRINGEMENT.

INDEMNIFICATION BY LICENSEE:

LICENSEE shall defend, indemnify and hold CyberSoft, its Distributors and licensors (including but not limited to CyberSoft, Inc.) and their respective officers, directors, shareholders, employees, agents and representatives harmless from and against any losses, liabilities, damages, demands, penalties and expenses (including, without limitation, court costs and attorneys' fees) arising out of or in connection with (i) LICENSEE's or Users' use of the Licensed Software, or (ii) any breach by LICENSEE, its Users, employees, agents or contractors of any representation, warranty or covenant of LICENSEE hereunder. Provided, however, if LICENSEE is the United States Government, the provisions of this Section 10 shall not apply.

EXPORT:

LICENSEE shall comply with all export or re-export restrictions and regulations imposed by the government of the United States. Without limiting the generality of the foregoing and regardless of any disclosure made by LICENSEE to CyberSoft, its Distributors or licensors (including but not limited to CyberSoft, Inc.) of an ultimate destination of the Licensed Software, LICENSEE shall not re-export or transfer, whether directly or indirectly, the Licensed Software or any system containing such Licensed Software, to anyone outside the United States of America without first obtaining a license from the U.S. Department of Commerce or any other agency or department of the United States Government, as required. The VFind™ Security ToolKit has been approved for export under General License GTDU, ECCN 5D002C. A special license is required for export to certain countries.

U.S. GOVERNMENT RIGHTS:

All CyberSoft products and documentation are commercial in nature. The Licensed Software is "Commercial Computer Software," as that term is defined in FAR 2.101. Where Licensee is an agency of the United States Government other than the Department of Defense, the clause at FAR 52.227-19 COMMERCIAL COMPUTER SOFTWARE (DEC. 2007) applies and is incorporated herein by reference. Except as expressly provided in FAR 52.227-19, as applicable, the Licensed Software is licensed to United States Government end users with only those rights as granted to all other end users, according to the terms and conditions contained in this License Agreement. Manufacturer is CyberSoft Operating Corporation, 1958 Butler Pike, Suite 100, Conshohocken, Pennsylvania 19428, U.S.A.

FORCE MAJEURE:

CyberSoft and CyberSoft, Inc. shall not be liable to LICENSEE for any failure or delay caused by events beyond CyberSoft's reasonable control, including, without limitation, acts or omissions of LICENSEE or its Users, acts of God, acts of government, acts of war or terrorism, sabotage, outages suffered by electric utilities, breakdown or damage to machinery, equipment or software, malfunctioning of software, corruption of data, unavailability of or interruption or delay in the Internet, telecommunications or third party services, failure of third party software, labor disputes, accidents, shortages of labor, fuel, raw materials or equipment, or technical failures (including, without limitation, defects in any Embedded Software).

GOVERNING LAW:

This License Agreement is made under and shall be governed by and construed in accordance with the laws of the Commonwealth of Pennsylvania, United States of America. Any dispute arising out of or in connection with this License Agreement shall be adjudicated exclusively in the state or federal courts located in Montgomery County, Pennsylvania, and all parties irrevocably consent to personal jurisdiction and venue therein. Provided, however, if LICENSEE is the United States Government, in lieu of the foregoing, this License Agreement shall be governed by the laws of the United States of America and the cases decided thereunder; and any dispute arising out of or in connection with this License Agreement shall be adjudicated in the Court of Federal Claims or the appropriate Board of Contract Appeals.

AMENDMENT:

The terms of this License Agreement may not be amended except in a writing signed by you and an authorized representative of CyberSoft. For purposes of this Section 15, Peter V. Radatti, CEO/Chairman or a person expressly designated by either of them in writing shall be an authorized representative of CyberSoft.

ENTIRE AGREEMENT:

This License Agreement and the Invoice, if any, constitute the entire agreement between you and CyberSoft with respect to the subject matter hereof and supersede all prior agreements, oral or written, with respect to the matters contained herein or relating thereto. In the event of a conflict between the Invoice and this License Agreement, the terms of this License Agreement shall govern.

NO WAIVER:

No failure or delay of CyberSoft to exercise any right, power or remedy, or partial exercise of any right, power or remedy, will preclude any future exercise of any such right, power or remedy. No express waiver or assent by CyberSoft to any default in any term or condition of this License Agreement shall constitute a waiver or assent to any succeeding default in the same or any other term or condition hereof.

BINDING EFFECT:

This License Agreement shall be binding upon and shall inure to the benefit of the parties hereto and their successors and permitted assigns. Except as otherwise expressly provided herein, the provisions of this License Agreement will not be construed as conferring any rights on any other persons.

SEVERABILITY:

If any provision of this License Agreement will be held illegal, unenforceable, or in conflict with any law of a federal, state, local or international government having jurisdiction over this License Agreement, the validity of the remaining portions or provisions hereof will not be affected thereby. IT IS EXPRESSLY UNDERSTOOD AND AGREED THAT EACH PROVISION OF THIS LICENSE AGREEMENT WHICH PROVIDES FOR A LIMITATION OF LIABILITY, DISCLAIMER OF WARRANTIES, INDEMNIFICATION OR EXCLUSION OF DAMAGES OR OTHER REMEDY IS INTENDED TO BE ENFORCED AS SUCH. FURTHER, IT IS EXPRESSLY UNDERSTOOD AND AGREED THAT IN THE EVENT ANY REMEDY UNDER THIS LICENSE AGREEMENT IS DETERMINED TO HAVE FAILED OF ITS ESSENTIAL PURPOSE, ALL LIMITATIONS OF LIABILITY AND EXCLUSIONS OF DAMAGES OR OTHER REMEDIES SHALL REMAIN IN EFFECT.

NOTICES:

All notices from you to CyberSoft must be in writing and sent by registered or certified mail, return receipt requested, or international equivalent, or by facsimile or electronic mail (receipt confirmed).

SVSP Protocol

CyberSoft Operating Corporation Simple Virus Scanning Protocol (SVSP)

This document is based upon the Manual pages delivered with the VSTK product along with new sample code. Please note that some aspects of this protocol may not exist in older versions of the program. Enhancements to the protocol are made on a regular basis. It should be noted that VFindD supports multiple protocols in addition to SVSP.

Name

SVSP - Simple Virus Scanning Protocol

Description

SVSP allows a client to send data files to a server for virus scanning. The protocol is a simple text based query/response, with asynchronous responses allowing simultaneous queuing of multiple files.

Notes

Commands may be prefixed by a numerical identifier, if so, the responses to that command will be prefixed by the same identifier. Filenames in command should not contain any quoting; file names in responses are quoted with double quotes; internal double quotes and backslashes are escaped with a backslash, newlines as \n.

After a SCAN command, SCANNING and possibly INFECTED responses are sent for each part of an archive/directory; for archives, the filename is then indicated as "archive" -> "part".

A QUEUED response may be sent if the scanning is delayed.

A DONE response is sent when all the parts of the scan request are complete. A client should not send additional SCAN commands until it has received a QUEUED, DONE, or Exxxxx response for the request.

A RECURSE limit of 0 implies unlimited (this is the default, if enabled).

For the IGNORE option, the virusname is as the virus name appears in the INFECTED response, e.g., CVDL W32/Nimda.E

If the EXPAND option is used without the expander value, UAD is initiated with the default set of active expanders. The list of expanders and their default state are: tar (on), Z (on), gz (on), text (on), HTML (off), zip (on), TNEF (on), Hqx8 (on), RAR (on), CAB (on).

Commands

[id] = customer defined identifier

[id] ENABLE <option>
Set a flag

[id] ENABLE <option> <value>
Add an option

[id] DISABLE <option> <value>
Clear a flag, remove options

[id] SCAN /<how> /<what> <args>
Scan data for viruses

[id] QUIT
Close and disconnect

Options

Options control details of the scanning process.

IGNORE <virusname>
Don't report this virus.

EXCLUDE <filename>
Don't scan this file.

EXPAND <expander>
Use this expander in UAD (* means all).

RECURSE <limit>
Expand composite files and directories recursively to the limit.

ORIGINAL
Scan original collection or compressed file as well as its components.

HTML [1-3]
HTML to text conversion level.

MAIL
UAD email mode.

Data Source

The /<what> tells the source of data to scan.

/REQUEST <filename>
Client connects to server port.

/PORT <portnum> <filename>
Server connects to client port.

/FILE <filename>
Scan data from file/directory.

/FILE-SHA1 <filename>
Scan data from file (& need its SHA1 hash).

/DATA bytes
Data sent inline, after SEND response (none) Inline as above, until socket closing.

Note: Only /FILE is implemented by the first version of the VFindD daemon.

Data Format

The /<how> indicates the format of the data to scan.

(none)
Raw data to be scanned.

/SS
SmartScan.

/LIST
Newline separated list of filenames/directories.

Note: Only raw data is implemented by the first version of the VFindD daemon.

Responses

[id] = customer defined identifier

The client should recognize these responses:

[id] READY
Initial connection prompt.

[id] DENIED
The connection is denied/blocked.

[id] OK
Command accepted (option enabled, etc).

[id] QUEUED
File queued for later scanning.

[id] Exxxxx message
Command failed (errno name).

[id] SEND <portnum>
Server provided port for scanning (from /PORT).

[id] SEND
Server is ready to receive data inline (from /DATA).

[id] SCANNING <filename>
File scan started.

[id] INFECTED <virusname>: <filename> : <message>
File is infected.

[id] DONE <viruscount> : <filename>
File scan is complete.

[id] DONE <viruscount> : <filename> : <SHA1-Hash>
File scan is complete (for SCAN/FILE-SHA1).

Errors

A specific subset of errno names may be returned to indicate errors

EINVAL
Invalid argument.

ENOTSUP
Argument or option not implemented.

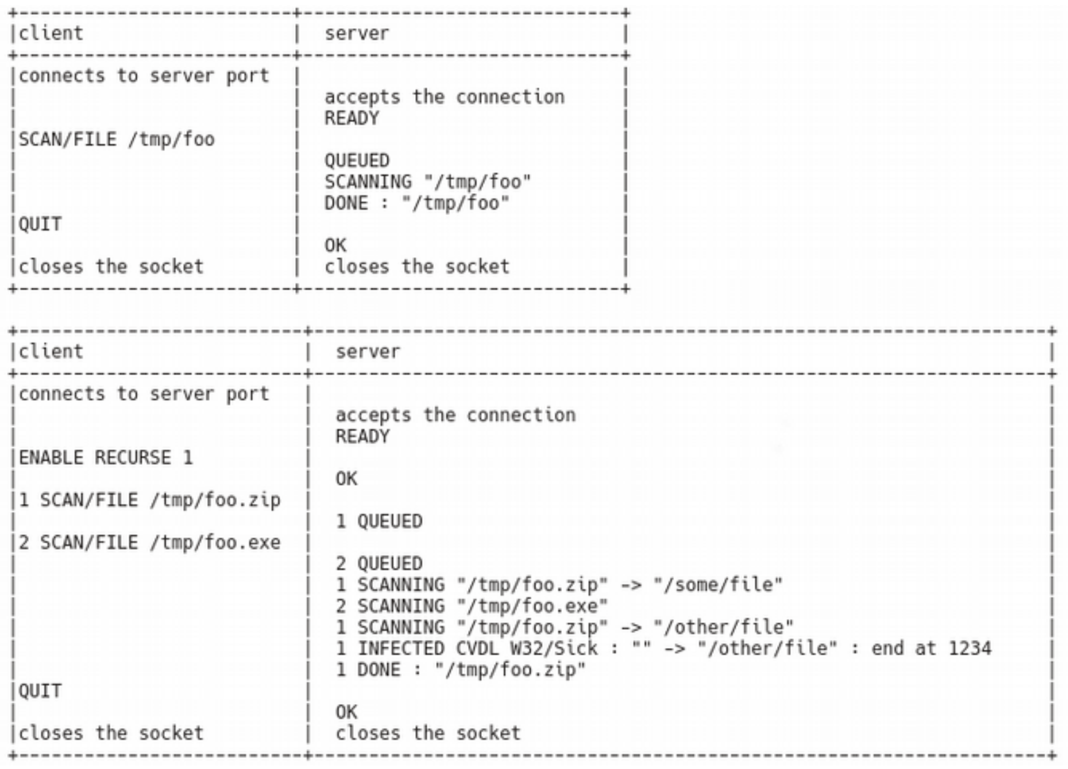
EACCES
File or directory could not be read.

ENOENT
File or directory does not exist; no connection to port.

EAGAIN
Server temporarily out of resources.

EFAULT
Server internal error.

Example



```
+------------------------+---------------------------+
|client                  |  server                   |
+------------------------+---------------------------+
|connects to server port |                           |
|                        |  accepts the connection   |
|                        |  READY                    |
|SCAN/FILE /tmp/foo      |                           |
|                        |  QUEUED                   |
|                        |  SCANNING "/tmp/foo"      |
|                        |  DONE : "/tmp/foo"        |
|QUIT                    |                           |
|                        |  OK                       |
|closes the socket       |  closes the socket        |
+------------------------+---------------------------+

+-------------------------+-------------------------------------------------------------+
|client                   |  server                                                     |
+-------------------------+-------------------------------------------------------------+
|connects to server port  |                                                             |
|                         |  accepts the connection                                     |
|                         |  READY                                                      |
|ENABLE RECURSE 1         |                                                             |
|                         |  OK                                                         |
|1 SCAN/FILE /tmp/foo.zip |                                                             |
|                         |  1 QUEUED                                                   |
|2 SCAN/FILE /tmp/foo.exe |                                                             |
|                         |  2 QUEUED                                                   |
|                         |  1 SCANNING "/tmp/foo.zip" -> "/some/file"                  |
|                         |  2 SCANNING "/tmp/foo.exe"                                  |
|                         |  1 SCANNING "/tmp/foo.zip" -> "/other/file"                 |
|                         |  1 INFECTED CVDL W32/Sick : "" -> "/other/file" : end at 1234 |
|                         |  1 DONE : "/tmp/foo.zip"                                    |
|QUIT                     |                                                             |
|                         |  OK                                                         |
|closes the socket        |  closes the socket                                          |
+-------------------------+-------------------------------------------------------------+
```

VDL Language

The CVDL language was first released to the public in VFind Version 4.0 in 1992. Using the CVDL fuzzy logic afforded by proximity, Boolean and case control logic it is possible to define one CVDL statement that can definitively locate all variations of a pattern, both known and unknown. In the terms of a computer virus this means that one VDL statement can be created that can detect more than one type of virus. In 2004 the CVDL language was expanded to include all of the "extended regular expressions (regex)".

One CVDL definition is called a "VDL statement" or just "VDL" for short. A collection of two or more statements is called a VDL program. VDL programs are sometimes referred to as VDL files.

A full explanation of the CVDL language is provided later in this manual.

Lexical Analysis using VFind's CVDL

The objective of this section is to introduce CyberSoft's approach of meeting the challenging lexical pattern analysis demands common in today's complex world. There is an additional manual which goes into full detail on this subject later in this handbook. All lexical analysis discussed in this chapter is done by VFind simultaneously to scanning for viruses.

Accommodating for dynamic patterns such as the example below cannot be met by conventional scanning methodologies. This is an important consideration when scanning messages for the medical field, for example, where the word Viagra can be found in legitimate communication. Using CVDL matching dynamic unsolicited messages becomes an attainable reality while bypassing messages that are most likely routine correspondence.

One feature of the CVDL language that contributes to the successful creation of dynamic pattern analysis is the ability to go beyond matching individual words and phrases. An analyst is able to expand upon Boolean expression logic utilizing an array of CVDL functions. Some of these functions may be seen in the example below.

Consider an e-mail message containing suggestive Viagra text such as:

```
"Click here to get your sample of ¥1ÄGR@"
```

The use of the CVDL language in the example below will find the word "click" regardless of case, and the word "Viagra" with any character substitution as defined in the macros for A, I, and V around "g" and "r", within two characters of one another. This allows the detection of the word Viagra even when it only appears to be "Viagra". False Matches are prevented by the NOT ~"viagra" (note the correct spelling) command at the outset of the VDL, and the WP1 (one or more white space or punctuation characters) surrounding the "viagra" alternative.

```
$define letterA "Aa@ÀÁÂÃÄÅàáâãäå"

$define letterI "Ii1;!¡¦ÌÍÎÏìîïlï|:"

$define letterV ({"Vv¥"} | ~"\/")

:SPAMVDL, NOT ~"viagra" AND~"click" AND WP1,$letterV,@-2,{$letterI},@-2,{$letterA},@-
2,~"g",@-2, ~"r",@-2,{$letterA},WP1 # ; This matches Viagra language with special
characters.
```

This sole Viagra example provides a glimpse of CVDL practicality in pattern matching. Its simple yet sophisticated approach provides a unique combination of tools to match the most dynamic of patterns while minimizing the possibility of false matches. The use of macros provides an additional edge not found in conventional scanning engines.

The CVDL language is a work in progress as it has been for over the last fifteen years. CyberSoft is constantly adding new constructs to the language in order to solve complex problems. This is because the problem of lexical analysis is becoming more and more difficult. The original driving force behind the lexical analysis using CVDL was dirty word checking used by the government and industry.

Think of words and phases such as "TOP SECRET" or "COMPANY PROPERTY". As an example of a lexical problem in the extreme, that is in itself an explanation of the problem, read the following paragraph:

Aoccdrnig to rscheearch at an Elingsh uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoetnt tihng is taht the frist and lsat ltteer is at the rghit pclae. The rset can be a toatl mses and you can sitll raed it wouthit a porbelm. Tihs is bcuseae we do not raed ervey lteter.

The prior paragraph has been circulating on the Internet as a joke. If the point of the paragraph was not understood, then go back and look at the spelling. A certain percentage of people will read that paragraph without noticing most of the spelling "errors". The ability of the mind to "fill in what is not there" very well defines some of the problem of lexical analysis. The CVDL language is very well suited to these kind of problems.

SVSP Sample Source Code

The following sample source code in the C language is intended to be a starting point for anyone who wants to implement the SVSP protocol with VFindd.

```
#include <time.h>
#include <stdio.h>
#include <errno.h>
#include <ctype.h>
#include <fcntl.h>
#include <string.h>
#include <limits.h>
#include <stdlib.h>
#include <unistd.h>
#include <alloca.h>
#include <sys/types.h>
#include <sys/stat.h>
...
...
...

#ifdef CONF_OS_WIN32
#define LOCALHOST "127.0.0.1"
#else
#define LOCALHOST "localhost"
#endif
...
...

int
vf_init(const char *const *argv)
{
        ...
        ...

        if (!servers_connected())
        connect_server(NULL, LOCALHOST);
        if (!servers_connected()) {
                printf("##==>>> No server connected -- exiting\n");
                fflush(stdout);
                exit(1);
        }

        ...
        ...

        return 1;
}

        ...
        ...
        ...

struct queue_data {
        struct queue_data *next;
        int pending_requests;
        const struct vfconf *config;
        int result;
        struct vfstat *vfstat;
        int sock;
        void *user;
};

struct per_request {
        const char *file name;
        struct queue_data *queue_data;
};

static int
handle_scan(int server,
        char *text,
        int size,
        int id,
        void *ptr)
{
        char buf[BUFSIZ];
        struct per_request *request = ptr;
        struct queue_data *q = request->queue_data;
        char *file name, *virus, *message;
        int bytes, error = 0;
        int waiting = 0;
        int file;
        char *args;
        char *end;
        int count;
        long port;
        int channel;

#ifdef DEBUG_CLIENT
        fprintf(stderr, "HANDLE %d: '%d %s'\n", size, id, text);
#endif

        args = nextword(text);
        text = trim(text);
        args = trim(args);

        if (q->pending_requests <= 0) {
                unexpected("SCAN", "No pending requests", NULL);
                goto done;
        }

        /* FIXME: break up in separate functions ? */

        if (strcasecmp(text, "QUEUED") == 0) {
        set_available(server);
        return 0;
        }

        if (strcasecmp(text, "SCANNING") == 0) {
        ++q->vfstat->total_file_count;
        announce_check(q->sock, args, q->user);
        return 0;
        }

        if (strcasecmp(text, "DONE") == 0) {

                if (split3(&virus, &file name, NULL, args)) {
                        count = (int) strtoul(virus, &end, 10);
                        if (count < 0 || *end != '\0') {
                                unexpected("SCAN", text, virus);
                                count = 0;
                        }
                        announce_done(q->sock, file name, count, q->user);

                        if (q->vfstat != NULL) {
                                q->vfstat->total_virus_count += count;
                        }
                }
                else {
                        unexpected("SCAN", text, args);
                }
                goto done;
        }

        if (strcasecmp(text, "INFECTED") == 0) {
                if (split3(&virus, &file name, &message, args))
                        announce_found(q->sock, file name, virus, message, q->user);
                else
                        unexpected("SCAN", text, args);

                return 0;
        }

        if (strcasecmp(text, "SEND") == 0) {
                if (args == NULL) {
                        unexpected("SCAN", "SEND: missing port (NYI)", args);
                        goto done;
                }
                port = strtoul(args, &end, 10);
                if (*end != '\0' || port < 0 || port != (unsigned short) port) {
                        unexpected("SCAN", "SEND: bad port", args);
                        goto done;
                }
#ifdef DEBUG_CLIENT
                fprintf(stderr, "Delivering '%s' to port %d\n",
                                request->file name,
                                (int) port);
#endif
                file = open(request->file name, O_RDONLY);
                if (file < 0) {
                        perror(request->file name);
                        goto done;
                }

                for (;;) {
                        error = channel = open_channel(server, (int) port);
                        if (channel >= 0)
                                break;

                        if (!is_temporary(error)) {
                                printf("##==>>> SCAN channel failed: %s\n",
strerror(-error));
                        }
                        if (communicate() == 0) {
                                if (option.quiet < 1 && !waiting++)
                                        printf("##==> Waiting for response from
server\n");
                                throttle();
                        }
                }
```

```
                for (;;) {
                        bytes = read(file, buf, sizeof buf);
                        if (bytes < 0) {
                                printf("##==>>> SCAN read failed: %s\n",
strerror(-error));
                                error = bytes;
                                break;
                        }
                        if (bytes == 0) {
#ifdef DEBUG_CLIENT
                                fprintf(stderr, "Delivered '%s' to port %d\n",
                                                request->file name,
                                                (int) port);
#endif
                                error = 0;
                                break;
                        }

                        error = send_command(channel, buf, bytes);
                        if (error < 0) {
                                printf("##==>>> SCAN channel SEND failed: %s\n",
                                        strerror(-error));
                                break;
                        }
                        if (error < bytes) {
                                unexpected("SCAN", "SEND", "not all data received");
                                break;
                        }
                        if (communicate() == 0)
                                break;
                }
                close(file);
                if (error < 0)
                        goto done;

                error = close_channel(channel);
                if (error < 0)
                        goto done;

                return 0;
        }

        error = errmsg(text);
        if (error < 0) {
                printf("##==>>> SCAN command failed: %s, %s\n",
                        strerror(-error),
                        args);
                goto done;
        }

        unexpected("SCAN", text, args);

 done:

...
...
...
```

```
        if (request != NULL) {
                if (request->file name != NULL)
                        free((char *) request->file name);
                free(request);
        }

        count = --q->pending_requests;
#ifdef DEBUG_CLIENT
        fprintf(stderr, "REQUEST COMPLETE, %d PENDING\n", count);
#endif

...
...

        return count >= 0;
}

static int
scan_request(const char *file name, int status, void *data)
{
        int use_local = option.localhost;
        struct queue_data *q = data;
        const struct vfconf *config = q->config;
        struct per_request *request = NULL;
        const char *failure = NULL;
        int id, server = -1;
        int waiting = 0;
        int error;

        if (filematch(file name, config->noscans)) {
                if (option.quiet < 1)
                        printf("##==> Not scanning: \"%s\"\n", file name);
                return 0;
        }

        if (status < 0) {
                error = status;
                failure = file name;
                goto fail;
        }

        if ((status & WALK_LINK) &&
                ((!config->follow_symlinks && (status & WALK_DIR)) ||
                (config->ignore_symlinks && (status & WALK_FILE)))) {

                if (option.quiet < 1)
                        printf("##==> Not scanning: \"%s\"\n", file name);
                return 0;
        }

        if (use_local && file name[0] != '/') {
                if (option.quiet < 1)
                        printf("##==>> Ignoring --localhost for relative file name\n");
                use_local = 0;
        }

        if ((status & WALK_DIR) && !use_local) {
                return 1;/* Nothing more to do, all handled inside walk() */
        }

        error = server = select_server();
        if (server < 0) {
                failure = "select_server()";
                goto fail;
        }
        set_busy(server);

        error = set_options(server, q->config);
        if (error < 0) {
                failure = "set_options()";
                goto fail;
        }

        /* free() called in handle_scan()
         */
        request = calloc(1, sizeof *request);
        if (request == NULL) {
                error = -ENOMEM;
                failure = "vf_scan()";
                goto fail;
        }
        request->file name = strdup(file name);
        if (file name == NULL) {
                error = -ENOMEM;
                failure = "vf_scan()";
                goto fail;
        }
        request->queue_data = q;

        for (;;) {
                id = request_id(server, request, handle_scan, &q->result, "SCAN");
                if (id < 0 && is_temporary(id)) {
                        if (communicate() == 0) {
                                if (option.quiet < 1 && !waiting++)
                                        printf("##==> Waiting for response from
server\n");
                                throttle();
                        }
                        continue;
                }
                if (id < 0) {
                        error = id;
                        failure = "request_id()";
                        goto fail;
                }
                break;
        }

        error = send_commands(server,
                                        "%d SCAN/%s %s\r\n",
                                        id,
                                        use_local ? "FILE" : "REQUEST",
                                        file name);
        if (error < 0)
                goto fail;

        ++q->pending_requests;
#ifdef DEBUG_CLIENT
        fprintf(stderr, "NEW REQUEST, %d PENDING\n", q->pending_requests);
#endif

        return !(status & WALK_DIR);/* SCAN/FILE recurses on the server */

 fail:
        if (server >= 0)
                set_available(server);

        if (request != NULL) {
                if (request->file name != NULL)
                        free((char *) request->file name);
                free(request);
        }
        if (failure != NULL) {
                errno = -error;
                perror(failure);
        }
        return error;
}

...
...
...

static struct queue_data *queue = NULL;

...
...
...

int
vf_queue(const struct vfconf *config,
                int sock,
                const char *file name,
                FILE *file,
                long size,
                int type,
                struct vf *vf,
                struct vfstat *vfstat,
                void *user)
{
        int error;
        struct queue_data *next;

        if (queue != NULL && queue->pending_requests <= 0) {
                while (queue->next != NULL && queue->next->pending_requests <= 0) {
                        next = queue->next;
                        queue->next = next->next;
                        free(next);
                }
        }
        else {
                next = queue;
                queue = calloc(1, sizeof *queue);
                if (queue == NULL)
                        return -ENOMEM;
                queue->next = next;
        }

        queue->config = config;
        queue->vfstat = vfstat;
        queue->sock = sock;
        queue->user = user;

        switch (type) {

        case VF_FILE_SCAN:
                error = process_file(file, file name, config, queue);
                break;

...
...
...
...

default:
        error = -EINVAL;
}

        if (file != NULL)
                fclose(file);

        return error;
}

...
...
```

SVSP Frequently Asked Questions
All answers tested in VSTK-179

1) User has verified that the data that they sent to the VFindd socket was read using truss, however, VFindd didn't seem to process all of the data. Sometimes sending more commands to VFindd would cause the unprocessed commands to process. A work around is to send a new line, which causes the commands to be processed.

CyberSoft recognized that the situation happens when a user uses telnet to send multiple scan-requests to VFindd at the same time, but it does not happen when users use VFindC (VFind-Client) which is provided as the default client. The source code example provided with this document is from the working VFindc program.

The cause of this situation is if the user tries to send multiple SVSP-commands on one request. Basically, VFindD handles the requests from a client using event (request)-driven-approach to optimize its performance for scanning and reduce some additional loads, which are not directly related to its scanning operation. The VFindD program uses a new-line ("\r\n" or "\n") as its parsing delimiter to recognize a valid SVSP-command in the stream. In other words, the scan-procedure (or any responses) on VFindD is invoked only when a new request from a client has arrived on the socket.

If a request arrives at VFindD through a socket, VFindD reads all the data on the socket and saves it to its queue. VFindD then picks-up one request from the queue to start scanning. This situation is good under the situation when one request has one SVSP-command. Of course, a client can send multiple requests without any problem if each request has one command but if a client tries to send multiple SVSP-commands in one request, the user may meet a "buffered scanning situation". Since one request will invoke one scan even though the remaining commands are still in the queue, the second command which was in the first request may need to wait for the next new request to be run in a FIFO manner.

We recommend that a client sends one command per request but the user can send multiple requests on the same socket. This approach was already implemented in VFindc (VFind-client) and it can be easily implemented by users in different programming languages. The only exceptional case in which this approach may not be easy is the situation in which a customer uses telnet to send files/directories to VFindD. That is because the "telnet" requests users to input all data manually and there is no room for a user to implement the logic/approach recommended above. In this case, CyberSoft recommends that a user gathers all the files in one directory and send the directory name as one request. Following are examples of two possible cases of a "buffered scanning situation". This example is for one request with five commands.

[1] Case #1

Added a new-line ("\r\n"-> Pressing "Enter" when editing a request) at the end of every command when he/she edits the ONE request as follows.

```
SCAN/FILE /dummy/file1.txt
SCAN/FILE /dummy/directory1
SCAN/FILE /dummy/file2.txt
SCAN/FILE /dummy/directory2
SCAN/FILE /dummy/file3.txt
```

Operations on VFindD

When the ONE request (including five commands) arrives at VFindD (socket), VFindD reads all the data in the socket. All five commands (as a ONE request) were saved in the daemon-queue, and the event-handler invokes scan-process for the request in the queue. After parsing the request using the new-line delimiter ("\r\n" or "\n"), the first command is recognized as a formal command from the client and started its scanning. The other four commands are still in the queue and wait for the next request from the client. The next request may be a formal command or just new-line ("Enter"), and the commands in the queue start scanning (eventually) on a FIFO-basis. The request to invoke the event-handler should be from a client.

[2] Case #2

No new-line ("\r\n") at the end of every command when the user edits the ONE request as follows.

SCAN/FILE /dummy/file1.txt SCAN/FILE /dummy/directory1 SCAN/FILE /dummy/file2.txt SCAN/FILE /dummy /directory2 SCAN/FILE /dummy/file3.txt

=> Operations on VFindd

VFindD sends the "ENOENT" error message to the client, which means "The file or directory does not exist", and waits for the next valid request on the socket. That is because there is no parsing delimiter (new-line) at the end of each command, (after reading and parsing the data) VFindd considers the data/stream in the request as follows.

- "SCAN/FILE" ==> The formal (and first) SVSP command, and
- All the remaining data after the "SCAN/FILE"==> file/directory name. It is because there is no newline at the end of one command. Of course, there is no file/directory-name like "/dummy/file1.txt … SCAN/FILE /dummy/file3.txt", and VFindd sends "ENOENT" message to the client, which means "File or Directaname="Part_02-Section_04-SS_01ory does not exist."

2) VFindD misses QUEUED responses.

Unlike the "SCANNING","DONE" or "Exx.. (Error names)", the "QUEUED" response from VFindd-mt may not be sent to the client especially if the user chose to use the multi-threading option with many threads and many files. So, do not use the "QUEUED" response as a trigger index for another action in the application.

The reason is that (under multi-thread environment) the thread for adding target files to the internal scan-queue and the thread(s) to pop-up the task from the queue (and start the scan a file) are different threads. After target files are added to the scan-queue by the main thread, other new created threads try to pop-up each task from the queue and start scanning. The order of adding the files to the scan-queue is independent of the order for pop-up (and scanning) the files by new thread by the operating systems kernel's thread scheduler.

The main thread takes the file names and adds them to the scan-queue. Then it does some internal logging procedure, and then tries to send the "QUEUED" response to the client if the task (file) in the scan-queue is still in the queue, in other words, it has not started scanning.

In most cases, especially if the number of target files is big (which means the number of newly created threads is big), the "QUEUED" message is sent to the client to inform that the task/file was queued for scan, however in some cases, the "QUEUED" message may not be sent to the client, because the queued task/file was popped quickly by another thread and started scanning BEFORE the main thread could check if the task is still in the queue. When the main thread checks the status, if the task is already popped and started scanning, the main thread does not send the "QUEUED" message because the file is not in QUEUE, it is in process.

3) Sending bad commands or random data causes VFindD to crash.

Problem does not exist. VFindD sends an "EINVAL" message to the client, which means "Unrecognized command", and waits for the next request. CyberSoft has not been able to cause it to crash using bad commands or random data. Example:

```
# telnet 127.0.0.1 8081(==> VFindd-mt on port 8081)
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
READY
wrongCommand (==> unimplemented command)
EINVAL Unrecognized command: wrongCommand(==> EINVAL from VFindd-mt)
SCAN/FILE /opt/vstk/README.txt
QUEUED
SCANNING "/opt/vstk/README.txt"
DONE 0 : "/opt/vstk/README.txt"
```

As can be observed from this test, a user is not able to cause VFindD to crash by sending it bad commands. Further tests show that sending random data also does not cause it to crash. CyberSoft is has not been able to create this problem with any version of VFindD, however if using a version prior to VSTK-T 179, an unsupported obsolete version is being used. Please upgrade.

4) If the VFindD process receives a request to scan a nonexistent file the client request to close the connection is not handled gracefully.

VFindD sends a "ENOENT" message to the client, which means "File or Directory does not exist", and waits for the next request. Example:

```
# telnet 127.0.0.1 8081
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
READY
SCAN/FILE nonexist
QUEUED
ENOENT "nonexist" (==> ENOENT from VFindd-mt)
```

As can be seen from this test, the user is not able to cause VFindD to fail in any way by sending it a request to scan a nonexistent file.

5) VFindD appears to create Zombie threads.

All the threads generated from the process (VFindD-MT) are terminated if the process (VFindD-MT) is terminated. Please check if another VFindD-MT was invoked on different port(s) and they are not terminated.

6) How many copies of the VFindD daemon can I run at the same time?

As many as the system has resources for. Multiple VfindD(s) can be run on different ports with "--SVSP-port=" option. Without this option, VFindD uses 8081 as its default port to listen the SVSP-connection.

7) Are there other APIs beyond SVSP?

Yes, but SVSP is the one that CyberSoft recommends. CyberSoft also supports APIs that are unpublished for specific customers and CyberSoft supports the ClamAV API for use of VFindD with open source programs that conform to the ClamAV API. The specific API CyberSoft uses for ClamAV is now obsolete but still works for those programs that use the older API. In the future this API will be brought up to date.

8) What is the difference between SCAN/REQUEST and SCAN/FILE?

The SCAN/REQUEST command was designed for a situation when the server (VFindD) is on a remote machine. In other words, the "SCAN REQUEST" command is for when the files to be scanned and the VFindd server are on different machines. While the "SCAN/REQUEST" command can be used for a client on the same machine it causes VFindd to perform additional work which is not necessary for "local' files. If the client is on the same machine with VFindD, CyberSoft recommends customers to use "SCAN/FILE" command, which does not consume additional hard-disk resources.

VFind

Name

vfind - Heterogeneous Anti-Virus Tool

Description

VFind is a heterogeneous virus scanner that simultaneously scans for a host of malware targeting Windows, UNIX, Linux and other systems, including denial of service attacks, back door attacks, hostile java applications and applets, OLE/VB5 macro viruses, and common hacks.

Notes

By default, CyberSoft does not install a path for execution. So a path will either need to be enabled, or use the full path when running the command.

Background

Many people think of VFind as an anti-virus program instead of a pattern analysis system. This is understandable since many customers need anti-virus programs and do not need a pattern analysis program. This perception is also not entirely wrong, since VFind started life as an anti-virus program and has multiple functions that are geared toward efficient processing for that purpose. In fact, all anti-virus programs are single purpose pattern analysis programs specialized to finding viruses. Since VFind is more than an anti-virus program, it has the ability to perform functions that are not normal to anti-virus programs. It has a different problem set to solve. VFind was the first anti-virus program to execute on something other than a Microsoft operating system.

VFind is unlike any other virus scanner in existence. It was the first anti-virus scanner for Unix and Linux, the first heterogeneous virus scanner, the first scanner to be part of a toolkit, the first scanner to include cryptographic hashing, the first scanner to deliver the scan database in source, the first scanner to positively identify the file being scanned and the first scanner to incorporate a full virus description language, CVDL. Unlike most virus scanners, it searches for attacks in a file based upon what the file contents reveals its type to be. Most virus scanners assume that the file name is a description of the file type. VFind determines the file type by direct examination of the file's contents. This makes VFind significantly more powerful than a virus scanner that only searches in files with the ".com" and ".exe" file name extensions for Microsoft executable viruses because it is not reliant upon a file name which, in a hostile environment (such as a virus attack), could be wrong. Without this additional functionality, a mere file name change can be used as a form of stealth attack. (example:Rename virus.exe to virus.dat.) In addition, this allows VFind to examine data in a byte stream in which file names may not exist. This can be a significant feature if a computer is network attached and invaluable in a firewall.

An example of an attack that VFind can detect but other anti-virus programs can not is a two body problem. A normal virus/trojan is a single body problem. Normally a virus infects an executable. The executable is named in such a manner as to allow it to be recognized as an executable by the system. When the virus infected executable runs, the payload is delivered. In a two body problem the payload is hidden in a file that is not normally scanned. An example of that is "payload.data".

This file will not be scanned by a normal virus scanner because it is not recognized by the operating system or the virus scanner as an executable. In this case the contents of the file are considered moot. The second body is an uninfected program which reads the payload file, loads it into memory and executes it.

A virus scanner may be able to clean up the results of the infection in recognized executables but the infection will reoccur every time the clean program executes since the payload file is never scanned. Since VFind evaluates every file and determines the contents by direct examination this problem can not occur because VFind would recognize the "payload.data" file as an executable. Special Note:There are viruses which will "infect" jpeg graphic files, flash files and other data type files that are normally not considered directly executable.

VFind is also heterogeneous. This is critical in a server or firewall environment. Often a server will contain executable programs for network attached client systems of a different architecture. A very common example of this is a Unix server providing network disk services to Microsoft Windows workstations. In this case, the Unix system can harbor viruses for the Windows system, even though it is itself immune to that attack. This problem was documented by Peter Radatti in a white paper published in 1991, "Heterogeneous Computer Viruses In A Networked Unix Environment" where the phrase "Typhoid Mary Syndrome" was coined to describe the problem. VFind solves this problem by simultaneously searching for Unix, Microsoft, Macintosh, Java and any other type of attack programs CyberSoft deems important.

An interesting aspect of VFind is that it simultaneously solves for all patterns. This makes it very fast when there are hundreds of thousands of patterns to search for. In addition, some versions of VFind are multi-threaded and can scan more than one file at a time. Multi-threading also allows VFind to take advantage of multiprocessor systems.

VFind includes the CVDL pattern analysis language. This system allows CyberSoft or the user to define new attacks or any other type of information that can be examined by advanced pattern analysis. CVDL is implemented in text so no binary modification is required. In addition to the proprietary CVDL operators, the language includes the entire Extended Regex language as a subset. CVDL is very powerful, includes macro capability, meta-vdl capabilities and is constantly being expanded.

A meta-vdl is a VDL that references other VDLs in order to decide if a pattern is solved. This is not a macro. Meta-vdl capability is unique and powerful. An example of where a meta-vdl is useful is if several VDLs are all solved for a dataset. Each of those VDLs would normally report, however the useful information that ALL of them solved would not be reported. A meta-vdl would allow the CVDL programmer to inform the user that a significant event took place. As an aid to the programmer VDLs can be set so as to not report to the user but will set a flag for the meta-vdl.

Another use for the CVDL system is to search for words or phrases that are not allowed on a system. These phrases could be proscribed as part of a organization policy such as sexual harassment, or could be part of a compartmentalization policy for handling classified information, or for restricting what programs may reside on a computer by direct examination. It also allows for reactive processing of espionage attacks in which data is being moved. Finally, CVDL allows for very fast updates of the VFind tool to search for new attacks without the need for replacement of the binary executable.

VFind has several engines. Each engine solves a different type of pattern analysis problem. These engines are under constant improvement and engines may be removed or added on a as-needed basis. The scanning engines are: "Basic Grunt," "Case Insensitive," "Jade", "Cryptographic Hash", "Bayes" and others. There are also support engines: "CVDL Compiler" and "Control System" but they are moot from a user standpoint.

The "Basic Grunt" engine is neither basic nor grunt, but is a very complex system developed over more than two decades of time. It is a fast parallel search engine. The search time in this engine is independent of the number of patterns being solved.

The "Case Insensitive" engine is the same as the "Basic Grunt" engine but was specifically designed to operate on case insensitive patterns. Since case only exists for textual data this engine is enhanced for that purpose. The engine views the letter "a" and "A" as the same letter. This engine is normally activated using the CVDL command "~". An example of which is ~"Pete".

The "JADE Engine" (Java Advanced Disassembly Engine) does a fast disassembly of Java Byte Code then searches for patterns in the disassembly. This is also available as a stand-alone command line tool for security analysts called JDIS (Java Disassembly). It is important to disassemble Java Byte Code prior to virus scanning because it is the only way to associate constant pool entries with opcodes. This association allows the virus scanner to scan for what is actually going on in the program instead of guessing. VFind is the only tool that does this. This is not a problem for JavaScript which is handled elsewhere in the system.

The "Cryptographic Hash" engine(s) is a subset of the Cryptographic Integrity Tool (CIT). It does not perform all of the functions of CIT but does process the danger-file database. The danger-file database is a flat file that contains a list of the names of known Trojan Horses and their MD5 cryptographic signatures. Since Trojan Horses and some other types of software attacks are rarely infectious and are generally not polymorphic, there is no need to search the entire file for a pattern when the entire file itself is the pattern. These attacks can not be made safe and should always be deleted upon detection. Consequently an MD5 hash value can be calculated for every file processed and then compare that hash value to the database of known hash values. If anything matches, it is an exact match.

Of course, this feature can also be used for the detection of objects that are not software attacks. Anything that is static can be digested into a hash value and placed in the database.

The "Bayes" engine was actually developed for the analysis of Unsolicited Bulk Email, UBE (sometimes referred to as Spam). CyberSoft's implementation of Bayes is different from, and in CyberSoft's belief, better than most other implementations. In addition, CyberSoft products allow it to be applied to any file desired. The applicability of the algorithm is dependent upon the Bayes database. In short, the Bayes engine allows objects to be statistically classified (file or data stream) as belonging to a class or family of objects without specifically having knowledge of the object under evaluation.

The determination of belonging to a family of objects is determined by whatever is programmed into the database. While not infallible, this engine allows for detection of needles in a haystack in a statistically meaningful way without knowing what kind of needle might be in the stack. Of course, this engine's value can be made either moot or valuable depending upon the quality of the database.

The "CVDL Compiler" changes text based programs written in the CVDL language to the internal structures required for identification. It supports both the CVDL proprietary language along with both the "POSIX Basic Regular Expressions" and "POSIX Extended Regular Expressions". It also interprets the vdl.list control file and manages all file type restrictions.

POSIX Extended Regular Expressions have been enhanced to PCRE Regular Expressions as of release 182.

The "Control" engine is basically the superstructure that manages all of the engines. This engine is available to end users via command line options.

1. SmartScan read (-ssr) from UAD or other sources (since SmartScan protocol is public)

2. quiet level and chevron output meaning (--quiet=n)

3. multi-threaded, --threads= option, plus -UAD=... option to have separate UAD/SmartScan input for each thread.

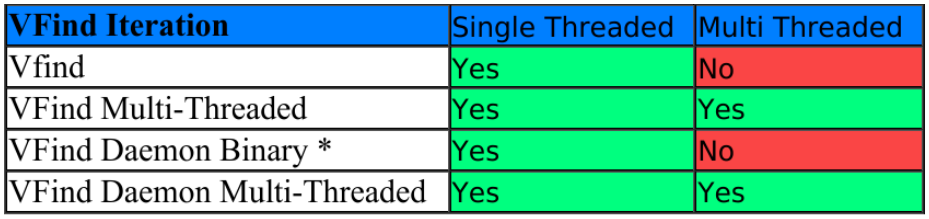
Customer Case Study

Customer case study number 2 demonstrates a typical use of VFind after a break in. This part of the document was written by the customer with edits for brevity and clarity.

> *"We installed VFind on four dedicated email firewalls. Two in the United Kingdom, one in the United States and one in Sweden. Another is planned for Australia. Each server is a Sun Micro 400 series with 4 processors. (We get a lot of mail). A few years ago we had external consultants draw up an anti-virus strategy document. They said, in summary:Use more than one anti-virus product, but definitely keep VFind on the firewalls. Most of the document was about selecting anti-virus for the desktops but essentially said that this decision could be made on costs grounds as long as VFind remained on the boundary.*
>
> *We merged with another company that had no firewall virus scanning. (They were not VFind customers.) When the Melissa virus hit the world, both halves of the company were infected and disconnected from the Internet. The day after the outbreak we have a signature installed from CyberSoft and isolated our infected PCs so we could reconnect our half of the company to the Internet. The other half of the company remained off-line for most of the week. While managing the desktops was difficult for both halves of the company we only had to update the three VFind servers to reconnect to the world. This allows us to update the desktops away from the crisis. The other half of the company installed VFind on its servers after this incident."*

VFind Iterations

There are different iterations of VFind. The first and oldest version is the standard single threaded command line version. Later, CyberSoft added the other versions to solve specific customer problems.



VFind Iteration	Single Threaded	Multi Threaded
Vfind	Yes	No
VFind Multi-Threaded	Yes	Yes
VFind Daemon Binary *	Yes	No
VFind Daemon Multi-Threaded	Yes	Yes

Multi-threaded programs allow the user to take advantage of multiprocessor systems on systems that support threading.

* - The VFind Daemon Binary Single Threaded is required for systems that do not support multi-threading.

VFind Speed - Comparing Apples to Oranges

While Windows based anti-virus only concerned itself with scanning selected parts of binary executable files, VFind did not have the same luxury. First, Windows anti-virus programs restrict what they scan by file name. Only programs ending in file extensions that are executable such as EXE, BAT, COM, etc. are scanned.

Secondly, with the exception of self extracting archives (which are executables), most Windows based anti-virus programs do not scan the entire file but restrict themselves to scanning selected parts of the file where viruses are known to reside. By restricting themselves to Windows they can take advantage of low level operating system directives to read only selected portions of each file, and not bother processing the bulk of each file. They do not run into a disk drive bandwidth bottleneck. This concept is not acceptable in a heterogeneous environment such as modern day servers and in-transit processing systems like firewalls. First, file names cannot be trusted to be accurate descriptions of the contents. Secondly, viruses that infect Unix and heterogeneous environments are more complex than Windows only viruses and may infect an executable program at any point in the file. There are also many more types of encapsulating technologies, of which the Microsoft based technologies are just a subset. When a complex system needs to be protected, halfway measures cannot be tolerated. Since VFind does not do things half way, people have the incorrect perception that VFind is slower than some Microsoft Windows based scanners. Consequently, the problem of this incorrect perception must be addressed.

To prove 100% coverage of every file, VFind has to scan every byte of every file. It also has to do the work of identifying everything by its contents instead of blind faith. This makes VFind appear slow when compared to windows based anti-virus programs but it allows it to do things other scanners can not do. The comparison is biased, since CyberSoft has clocked VFind as significantly faster than Windows based anti-virus programs at processing data. This means that VFind is faster, but handles a larger workload, which causes the overall time to be greater. There are now competing products on the Unix platform, however, they are slower than VFind. Another way to understand this problem is that a Windows based anti-virus scanner can scan a very large disk drive in a matter of minutes. If the user reviewed the total amount of data being scanned, and divided that number by the total number of minutes the scan took, the user would learn that the program appears to be processing data at a rate that exceeds the maximum data transfer rate of the disk drive. This is of course not possible, so logic dictates that the Windows based scanners are not processing all of the data.

This is no longer true of some of CyberSoft's competitors' programs that run on Unix and heterogeneous environments. It is also not true of Java based anti-virus programs which are painfully slow, but for reasons that have to do with Java and not what was already stated. Other competitors learned that doing things half way in these complex environments does not work.

In fact, when VFind was compared to CyberSoft's competitors' Unix offerings, VFind was significantly faster. This however, is no help since people's perceptions were created on the Windows platforms which cut corners in favor of speed at the cost of security and attention to detail.

The issue of speed is a perceptional problem and not a technical problem. CyberSoft's solution has been twofold. The first has been educational. This has been somewhat successful, but it is not feasible to educate the everyone. The more successful solution has been technical. CyberSoft has been making VFind faster. It took almost 15 years of constant effort, but CyberSoft's algorithms are now significantly faster than any other commercial product without any reduction in the features. VFind is doing more work than anyone else, at a faster pace. The next thing that CyberSoft implemented was context switching of VDLs based upon content. CyberSoft's product now has the option of restricting looking for OLE based viruses to OLE files. In the past, CyberSoft products searched every file for every type of virus. There is a small risk to this. Since this feature is optional, it is only used where the risk has been deemed to be low or nonexistent. One example where this strategy cannot be used is that OLE documents must now be searched for some types of binary executable viruses. Due to the flexibility of the CVDL language, it is trivial to direct VFind to perform this task and to do so without changes to the VFind binary executable.

There are still ways in which speed can be compromised and which the end user has direct control. The first is the way in which the user will use VFind. When using VFind inside of another program, an example of which is a script, then VFind should not be restarted for every file scanned. VFind has a large startup overhead. Over the course of thousands of files that overhead is negligible but over the course of one file it is large. Use the daemon, or any other method, but do not restart VFind for every file.

Another way in which speed can be compromised is to thrash the disk drive. A thrashing disk drive will slow down everything on the systems, not just VFind. Basically, the disk drive is causing the heads to move excessively. Some ways of causing thrashing is to have too little RAM memory in the system. This will cause a large amount of read/writes to the swap partition of the drive. If the swap partition is on the same drive that other reads and writes are being done to, then that could slow the system down. In this case the solution is to add RAM to the system and to move the swap partition to a drive of its own. For maximum benefit, use an SSD drive for swap and it will benefit many programs and not just VFind. Finally, if scanning many compound files then the UAD program breaks these down into their components before processing by VFind. If the /tmp directory is on the same drive then both thrashing and throttling could occur. Throttling is when the amount of data being read/written to a drive exceeds the bandwidth of the drive. The solution here is to either use a RAM disk/ SSD disk or to move the tmp space UAD uses to someplace else in the system file structure that is on a different drive. The UAD command line to do this is -tmpdir=xxxx where xxxx is the directory to be used instead of /tmp. These are all basic system tuning issues that can be resolved by a System Administrator. A more difficult to resolve problem is caused by the fact that the CVDL language is full featured and CVDL statements can be written that are extremely slow. Some operators are slower than others. In addition, very complex scanning programs can be created. Just like any computer language VFind will do what it is told to do. If it is given a pattern analysis program that means that it must do millions of operations more than normal that will slow down the program. Pattern analysis programs can always be optimized in the CVDL language to avoid this problem.

The VFind command line option "--trig-count" will help to determine which CVDL statements are slowing down the CVDL program and fine tune them.

There are many considerations when tuning VFind for speed. Most of these considerations are in the user's control. Some of these issues will now be examined below.

Historically, virus scanning and pattern recognition were dependent on two things: disk I/O performance, and processing power. Essentially, how fast data can be read, and how fast data can `be processed. Because of this, tuning and optimizing VSTK for performance applications is very much hardware dependent. However, there is still opportunity to improve performance by optimizing how system resources are utilized, and VSTK gives the user the flexibility to adapt the product to many different environments. The key to increasing performance with VSTK (and specifically VSTK-Turbo) is parallelism. This means removing data flow bottlenecks and fully utilizing the CPU. The first rule of thumb is to avoid using pipes while invoking components of VSTK at all costs. This causes a situation where data can only be processed in a linear fashion, restricting the flow of data into VFind-MT. For example, using the find command in conjunction with UAD and VFind-MT connected via pipes, will cause VFind-MT to only be as fast as UAD can process, which is one file at a time. This situation can be easily avoided with the --uad flag which is built into VFind.

When used in conjunction with the --threads flag, VFind-MT will invoke one UAD process for every VFind process, thus removing the bottleneck caused when using only one UAD process. Threading is the next consideration in the optimization process. It is a common misconception that there should be close to a one-to-one relationship of processes to processor cores. Most of the time, this rule does not apply and will not be effective because all system resources available will not be utilized. In many cases, it is actually much more beneficial to run many times more threads than physical cores on the processor. The number of threads used, directly equates to the number of files VFind is able to process at any given time. If 16 threads are being used, and there is still more than enough processing power left over, and the disk is not being thrashed, then there is the potential to process more data at once simply by increasing the thread count. The optimal thread count will differ from system to system.

Different types of processors and different types of disk setups will handle vastly different types of work loads. A system with many large files and archives will largely benefit from a lower thread count as a lot of processing power is required. A system with many small files will benefit from a higher number of threads as the files can be read and processed more quickly. Logically then, systems with a wide variety of data will find that its optimal thread count is somewhere in the middle. Eventually, limits on the current hardware may be reached, at which point the hardware configuration may need to be optimized. As previously mentioned, processing power and disk I/O are where most bottlenecks occur, so these should be the focus. Processors which cope well with many small tasks run in parallel, such as most modern multi-core desktop processors, will work best. Generally speaking, the more physical cores in a system, the better VSTK will be able to perform. Hard disk drives (HDDs) are the most common source of I/O bottlenecks. HDDs are notoriously slow because the disks are physically too slow to provide large amounts of data very rapidly.

This was a problem until recently when solid state disks (SSDs) became widely available at an affordable price point. Since SSDs are mostly random access, it is possible to read from them at speeds exceeding 500 MB/s, making them perfect candidates for increasing VSTK performance. CyberSoft has demonstrated and conclusively proven that, with relatively inexpensive off-the-shelf hardware, the ability to process data at a rate of up to 2.43 gigabytes per minute with room for improvement. It is important to note that results may vary, depending on the hardware used, as well as how heavy of a burden is already being placed on the system by other programs. During testing, CyberSoft used 8-core desktop processors with SSDs to achieve these speeds. A highly parallel system configuration and simple product optimization is what made these speeds possible. With enterprise level hardware, it would be theoretically possible to process orders of magnitude more information.

Usage

Input

VFind can be run in three ways.

1. Interactive mode:
Running VFind without any file arguments (or other input such as SmartScan and stdin) will result in a prompt asking what file to scan. Example:

```
$ vfind
```

2. Batch mode:
VFind can be invoked with a list of files (or other input such as SmartScan or stdin). In this mode, VFind will scan all of the targets and write a report to stdout. This mode is useful when scanning many files or directories.

Example:

```
$ vfind *.doc *.exe
```

3. Automated mode:
VFind can be run from a script, batch file, or other application and be scheduled using UNIX cron or a similar program. To run in this mode simply create a VFind command and place it in the appropriate place in the script, batch file, or application. When this mode is invoked, VFind will run unattended and generate a report to stdout. This report can be redirected to a file, emailed, or otherwise processed. This mode of operation is useful when scanning a large amount of data on a regular basis.

Output

VFind's output can be very verbose at times. In order to cut down the output CyberSoft recommends using the choke method. The choke method is as simple as piping the output from VFind into grep, or a similar tool.

Each line of output from VFind starts with a chevron as follows:

```
Chevron Meaning
-------------------------------------
##==>Informational Message
##==>>VFind Warning
##==>>>Serious VFind Condition
##==>>>>Possible Virus Detection
```

Example:

```
$ vfind / | grep '##==>>>>' REPORT
```

The above example would only show errors and virus detection messages.

Custom Messages

By default, VFind output is worded for virus scanning. When VFind is used for scanning for things other than viruses, it is possible to customize its messages thru the use of particularly formatted comments in the *.vdl files, or in the vdl.listfile for messages that are not specific to a particular VDL file. Comments of the form $name = sets the named variable to the given value. The following variables are recognized, and will change the default text (in parentheses below) to the specified value.

$pattern-type (VIRUS)

A short name used to identify the pattern.

$pattern-match (VIRUS POSSIBLE) Message presented when a pattern matches.

$pattern-description (possible virus infection) Message describing what the pattern matches. $pattern-not-found (No apparent virus infections) Message presented when no patterns matched.

Variable names and values are case independent; VFind adjusts the case to match the running text of the output. VFind adds -e or -es when needed to evaluate the pattern type or pattern description in plural; more complicated plurals can be given explicitly by specifying the variable value as singular/plural.

SmartScan

VFind is a SmartScan compliant tool. Specifying the -ssr option to VFind will cause it to read a SmartScan stream from stdin.

For example:

```
$ find /export/home -type f -print | uad -s -ssw | vfind -ssr > REPORT
```

Speed

Why would a user ever want to use less than the maximum speed? Most users will never have to worry about this; however, here are a couple of reasons some might.

One reason is that there is a space/speed trade-off. If the dynamic space required to run VFind with --speed=2 is prohibitive on the machine (i.e., VFind can not run or there is excessive paging), try --speed=1.)

Another reason involves the trade-off between start-up time and marginal scan time. With --speed=2 there may be a substantial start-up time as VFind initializes various internal structures. This might be on the order of, e.g., one second. When scanning a single small file, this might be a waste of time.

On the other hand, --speed=2 provides the fastest marginal scan time, that is, the time needed to scan each extra byte of data. Thus, when scanning large amounts of data with a single invocation of VFind (such as when handling SmartScan data from UAD or handling a large number of file names piped in via standard input), --speed=2 (if there is space for it) is a good idea despite the start-up time.

Locking

Note: This feature is provided for backwards compatibility only, and is scheduled to go away in a future release of VFind. To update VDL files on the fly, the files should be copied/untarred first with a different name, and then renamed to their correct name. This will assure that VFind can not read a file before it's complete.

VFind includes an internal locking mechanism to facilitate VDL updates. This is useful for systems where VFind processes are started continuously, for example a mail server which runs VFind automatically to process one or more newly arrived mail messages. If updated VDL files were installed at the same time that a VFind process was started, the VDL data read by VFind could be wrong or missing. This problem is avoided by using lvfind and lvfind links to VFind which use internal locking. A dummy file, $VSTK_HOME/data/LOCK by default, is used for the fcntl() locking. There is also a --lock= command-line option to specify an alternate lock file.

If VFind is invoked using a name starting with 'l', (e.g. an lvfind symlink to VFind), then it attempts to acquire a shared (read-only) lock on the LOCK file. If invoked using a name starting with 'l' (e.g. lvfind), then it attempts to acquire an exclusive (read-write) lock on the LOCK file.

Shared locks do not interfere with other shared locks, but will fail if there is an existing exclusive lock. An exclusive lock will fail if there are any other locks of either type. Shared locks require only read access to the LOCK file, but an exclusive lock requires read-write access.

lvfind will release the lock only after reading all VDL files, including those from the data/vfind/vdl.list VDL list file plus any others specified using --vdl=, --vdlc=, etc. command-line options.

lvfind simply waits up to 60 seconds (--sleep-time option) and then exits, it does not read any VDL files or scan any input data. It prints the process id to stderr (as though --pid was specified) to facilitate killing it.

idth="7.5in"> The default values for command-line options are equivalent to specifying:

```
lvfind --lock-retries=60
lvfind --lock-retries=60 --sleep-time=60
```

If locking fails due to interfering locks, it is retried up to 60 more times (--lock-retries option), with a one second delay between attempts. The locking could fail, for example, if a VDL update process is started while an lvfind process holds a shared lock. If the locking fails due to some reason other than interfering locks, that is a fatal locking error; lvfind will set the VFind error flag, give up on the locking, and continue; lvfind will just exit.

After 60 failed locking attempts (--lock-retries option), lvfind will set the VFind error flag and give up, continuing with the rest of the program; but lvfind will just exit.

Restarting

VFind will restart using execvp() when receiving signal SIGHUP. This is useful with the -i, ignore-eof option when running VFind as a daemon, to restart after updating VDLs. VFind should only be restarted when input is at EOF, otherwise stdio buffering can cause a loss of data and/or SmartScan desyncronization.

VFind Daemon

NAME

vfindd - Heterogeneous Antivirus Daemon

Description

VFindD is a heterogeneous virus scanning daemon that simultaneously scans for a host of malware targeting Windows, UNIX, Linux and other systems, including denial of service attacks, back door attacks, hostile java applications and applets, OLE/VB5 macro viruses, and common hacks. The daemon runs in the background and accepts SVSP, SMTP, and CLAMD connections.

Note: In the current version, the SVSP SCAN command is only implemented in its SCAN/FILE form.

VFind Daemon Benefits

VFind Daemon provides user applications virus scanning and detection services at a high level of performance. Running as a daemon process eliminates the need to re-initialize the scan engines on each request. Without the overhead of initialization, files are processed as they are received,improving response time and minimizing the effect of virus scanning on the main application. CyberSoft's tests indicate that VFind Daemon is able to scan a user's Internet accesses without introducing any noticeable delay. In addition to the initialization savings the VFind Daemon offers multi-threading for multiprocessor systems. This allows the application of additional processing power of a multiprocessor system to VFind. The VFind Daemon is part of the VFind Security Tool Kit Turbo product. (VSTK-T)

VFind Daemon makes VFind's scanning and virus detection services accessible to any application running on a user's system. An application connects to VFind Daemon using sockets. The easy to use message interface makes passing a file scan request and processing the result a straightforward task. VFind Daemon's multi-threading capability enables it to scan requests from multiple applications concurrently.

Applications can access VFind Daemon services through an easy-to-use message interface. The Simple Virus Scanning Protocol (SVSP) is a text-based, request/response interface that gives applications full access to VFind Daemon's services. SVSP includes commands that enable the program to set scanning options on a per-request basis and to specify the file to be scanned. Requests can be tagged so that the subsequent responses can be matched.

This allows the application to submit multiple scan requests and be able to match the asynchronous responses. See SVSP documentation, source code example and FAQ in Section 2 of this manual.

The VFind Daemon also supports the interfaces for other available virus scanning daemons, for example, ClamAV's clamd. This makes it possible to incorporate VFind Daemon into an existing system with minimal software changes and enables applications to migrate towards utilizing VFind's additional capabilities as required. NOTE: The version of the ClamAV interface supported is now out of date, as of May 2017, and will be updated sometime in the future.

VFind Daemon's multi-threading capability enables it to scale gracefully and take advantage of systems with multiple processors. The number of threads used by VFind Daemon is configurable and can be set to match the available computing power.

VFind Daemon Features

The VFind Daemon is a multi-threaded server process. It provides all the functionality of the UAD and VFind programs. VFind Daemon supports file the expansion of archive files and parses mail messages and HTML files. Multiple scan engines are supported: the parallel scanning engine, emulator engine, MD5 engine, Bayes engine, JPEG engine, and Java Disassembler. The VFind Daemon is highly configurable, enabling the user to specify whether to scan for viruses or spam or both. VFind Daemon can be configured to scan for information that is of concern to a specific business or site.

Applications connect to the VFind Daemon using sockets and using the Simple Virus Scanning Protocol (SVSP) interface. SVSP is a text-based, request/response interface. SVSP commands allow an application to set scan options and to send a request to scan a file. The scan options can be enabled/disabled on a per request basis. The results of a scan are returned in an easy to parse message. If a virus is detected in a file, a response is returned to the application that identifies the infected file and identifies the virus. SVSP supports message tags, that is, if a request is tagged, the resultant responses are returned with the same tag. This feature allows the application to send multiple scan requests and to use the tags to match the responses. As of release 183, VFind Daemon now supports the SVSP SCAN/DATA command, and all SVSP commands are now case-sensitive.

The VFind Daemon supports multiple interfaces for scanning daemons, for example, the ClamAV clamd interface, in addition to propriety interfaces and CyberSoft's SVSP interface. VFind Daemon accepts the message formats used to submit requests to clamd and returns the results in the expected format. NOTE: As of 2017, the version of the ClamAV interface supported is out of date and will be updated sometime in the future.

Usage

Input

VFindD runs as a background daemon, accepting connections by TCP and Unix sockets.

Output

VFindD doesn't produce much output; output is instead expected from any client connecting to the daemon. When running in the background (normally, without --foreground), any output from VFindD can be found in:

$VSTK_HOME/var/vfindd.log.

Locking

VFindD does not provide any locks against multiple access. To update VDL files on the fly, the files should be copied/untarred first with a different name, and then renamed to their correct name. This will assure that VFindD can not read a file before it's complete.

Restarting

VFindD will restart using execvp() when receiving signal SIGHUP. This is useful to restart after updating VDLs. VFindD should only be restarted when it is idle, otherwise stdio buffering can cause a loss of data and/or SmartScan desyncronization.

ClamAV & ClamD Compatability

The VFind Daemon supports the interface for the ClamAV's CLAMD daemon, it recognizes SCAN, CONTSCAN, RAWSCAN, STREAM, SESSION, and END requests. The CLAMD interface usually listens on a UNIX socket at /tmp/clamd.

```
VFind Example Script: cit_uad_VFind_lb.Sh
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#!/bin/sh
#
# Name: cit_uad_VFind_lb.sh
#
# Copyright: 2003, 2004, 2005, 2006 CyberSoft Operating Corporation, All Rights
Reserved.
#
# Description: This is a general script that will demonstrate how to run the CIT, UAD
# and VFind programs using the SmartScan protocol and the Loop-back tools.
# The find command options may need to be changed to properly scan your system.
# The uad --recursion option is set to 25 to limit the levels of recursion for expanding
# archive files.  This may need to be altered for your system.
#
VSTK_HOME=<vstk_home>
export VSTK_HOME

VFind=VFind
VPATH=/
VOPTS=
COPTS=
UOPTS=
mflag=
pflag=
vflag=
uflag=
cflag=
while getopts m:p:v:u:c: name
do
case $name in
    m)
        mflag=1
        mval=$OPTARG
        ;;
    p)
        pflag=1
        val=$OPTARG
        ;;
    v)
        vflag=1
        vval=$OPTARG
        ;;
    u)
        uflag=1
        uval=$OPTARG
        ;;
    c)
        cflag=1
    cval=$OPTARG
        ;;
    ?) printf "Usage: %s [-m ] [-p ] [-v 'VFind options'] [-c 'cit options'] [-u 'uad
options']\n" $0
        exit 1
        ;;
    esac
done

if [ ! -z "${mflag}"-a-x ${VSTK_HOME}/programs/VFind-mt ]; then
    VFind=VFind-mt
    NUM_THREADS=${mval}
fi
if [ ! -z "${pflag}" ]; then
    VPATH=${pval}
fi
if [ ! -z "${vval}" ]; then
    VOPTS=${vval}
fi
if [ ! -z "${uflag}" ]; then
    UOPTS=${uval}
fi
if [ ! -z "${cflag}" ]; then
    COPTS=${cval}
fi

PATH=/usr/bin:/usr/sbin:/bin:/sbin:$PATH
export PATH
```

```
.$VSTK_HOME/bin/_config.sh

if [ ${VFind} = "VFind-mt" ]; then
find ${VPATH} ${FIND_FLAGS} -type f -print | \
    $VSTK_HOME/bin/cit${COPTS} | \
    $VSTK_HOME/bin/lbh -db=lb.db | \
    $VSTK_HOME/bin/${VFind} --threads=${NUM_THREADS} --uad="--recursion 25 ${UOPTS}"
--exit-on-vdl-error ${VOPTS} | \
    $VSTK_HOME/bin/lbt -db=lb.db -r=4
else
find ${VPATH} ${FIND_FLAGS} -type f -print | \
    $VSTK_HOME/bin/cit ${COPTS} | \
    $VSTK_HOME/bin/lbh -db=lb.db | \
    $VSTK_HOME/bin/uad --recursion 25 -s -ssw ${UOPTS} | \
    $VSTK_HOME/bin/${VFind} -ssr --exit-on-vdl-error ${VOPTS} | \
    $VSTK_HOME/bin/lbt -db=lb.db -r=4
fi
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

VFind Client

Name

vfindc - Heterogeneous Antivirus Client

Description

VFindC is a client that attaches to VFindD, and scans for a host of malware targeting Windows, UNIX, Linux and other systems, including denial of service attacks, back door attacks, hostile java applications and applets, OLE/VB5 macro viruses, and common hacks.

Usage

Input

VFindC can be run in three ways.

1. Interactive mode:
Running VFindC without any file arguments (or other input such as SmartScan and stdin) will result in a prompt asking what file to scan. Example:

```
$ vfindc
```

2. Batch mode:
VFindC can be invoked with a list of files (or other input such as SmartScan or stdin). In this mode, VFindC will scan all of the targets and write a report to stdout. This mode is useful when scanning many files or directories. Example:

```
$ vfindc *.doc *.exe
```

3. Automated mode:
VFindC can be run from a script, batch file, or other application and be scheduled using UNIX cron or a similar program. To run in this mode simply create a VFindC command and place it in the appropriate place in the script, batch file, or application. When this mode is invoked, VFindC will run unattended and generate a report to stdout. This report can be redirected to a file, emailed, or otherwise processed. This mode of operation is useful when scanning a large amount of data on a regular basis.

Output

VFindC's output can be very verbose at times. In order to cut down the output CyberSoft recommends using the choke method.

The choke method is as simple as piping the output from VFindC into grep, or a similar tool. Each line of output from VFindC starts with a chevron as follows:

```
Chevron Meaning
-------------------------------------
##==>Informational Message
##==>> VFindc Warning
##==>>> Serious VFindc Condition
##==>>>> Possible Virus Detection
```

Example:

```
$ vfindc / | grep '##==>>>>' REPORT
```

The above example would only show errors and virus detection messages.

Custom Messages

The current version of VFindC does not support Custom Messages (see VFind); VFindC output is worded for virus scanning only. Support for custom messages in VFindC may be added in a future release.

SmartScan

The current version of VFindC does not support SmartScan streams; file typing and extraction using UAD can instead be achieved using the --uad= flag.

Locking

VFindC does not provide any locks against multiple access. To update files while VFindD is scanning them, the files should be copied/untarred first with a different name, and then renamed to their correct name. This will assure that VFindC can not read a file before it's complete.

Restarting

VFindC does not support restarting using SIGHUP.

Command Line Options – VFIND

Version – [184]

```
   --force-prompt [New to 184]
   Option which allows the user to force the 'enter filename'
```

Version - [181-184]

--license=path
Specify an alternate path to the LICENSE file.

-c, --copyright
Display copyright information and then exit. All other options will be ignored.

-h, --help
Display usage message and then exit. All other options will be ignored.

-v, --version
Display version information and then exit. All other options will be ignored.

-d, --dup-check
Tells vfind to check for duplicate VDL names and definitions, and other potential problems. With this option enabled, duplicates will be reported as parser errors. Also, any VDL segment which starts with an offset range or .* operator will be reported as a parser error. An offset at the beginning of a VDL segment is allowed by the CVDL syntax, but does not make sense to use, and may cause the VDL to run very slow.

--trig-count
Lists the names of any VDLs that could not be indexed for speed, and also the trigger and run counts for all VDLs. In general, only simple VDL constructs can be indexed, and only constructs containing strings of four or more bytes. Having many non-indexed VDLs, or VDLs with excessive trigger hit counts, will make VFind run significantly slower.

-e, --exit-on-error
Tells VFind to exit immediately after encountering any kind of error or warning condition. Normally, VFind prints a warning message and attempts to continue processing after encountering a non-fatal error, such as a syntax error in a VDL description.

-ev, --exit-on-vdl-error
Tells VFind to exit immediately after encountering any kind of error related to processing of vdl files. Normally, vfind prints a warning message and attempts to continue processing after encountering a non-fatal error, such as a syntax error in a VDL description.

-i, --ignore-eof
Tells VFind to ignore end-of-file and keep trying to read input files names or SmartScan input. --end and --quit may still be used to exit Interactive mode.

--jadevdl=file
Tells VFind to load additional virus signatures from file. File contains VDL models for hostile java applets and applications.

--libon=library, --liboff=library
Turn on/off library. VFind will list the available libraries upon startup. Amiga and eicar libraries are turned off by default. Use --libon='*' to turn on all libraries.

--md5=file
Tells vfind to read additional MD5 virus signatures from file.

--cbayes=file
Tells vfind to read Read additional cbayes data from file.

--noload=virus
This option provides a way to disable loading of individual VDLs. This may be useful if your site gets a lot of false positives for some particular virus due to the type of data you have. Virus is the name of the virus as it appears in the VDL file, for example: --noload="W32/Sircam.a"

--noloads=file
This provides a way to specify multiple noload parameters in a file. File is a file that contains valid virus parameters as described in the "--noload" option. For each line of the file, leading and trailing whitespace is stripped, then lines which are empty or start with '#' (i.e. comments) are skipped.

--noscan=filename
This option provides a way to turn off scanning of particular files or directories. This may be useful when scanning disks containing e.g. quarantined viruses or virus scanner configuratuion files. If the name contains a '/' (slash) character, if is matched agains the full name of the scanning target, otherwise, it is matched only agains the final component, after the final slash. As of release 183, file exclusion patterns now support wildcards and new match rules. Pattern matches if: input equals pattern, absolute paths are equal, basname of filename matches wildcard pattern, or absolute path of filename matches wildcard pattern. Wildcards must be used with absolute paths only, not relative paths.

--noscans=file
This option provides a way to specify multiple noscan parameters in a file. File is a file that contains file or directory names as described in the noscan option.

--notell=virus
This option provides a way to turn off reporting of individual viruses. This may be useful if your site gets a lot of false positives for some particular virus due to the type of data you have. Virus is the name of the virus as it appears after "VIRUS ID: " in vfind's output, for example: --notell="CVDL W32/Sircam.a"

--notells=file
This option provides a way to specify multiple notell parameters in a file. File is a file that contains valid virus parameters as described in the notell option.

--force-scan
Force scanning of the compiled vdl data file. This file is normally not scanned, as it would be very slow, and produce a large amount of false hits. The file is instead protected by a cryptographic checksum; any modifications to the file are reported as hits on the "Forgery.1" virus id.

--recursion-limit=number
Maximum depth to recursively scan directories, negative for unlimited (the default), or zero to not scan directories at all.

--follow-symlinks
If this flag is given, VFind will follow symbolic links pointing to directories; otherwise, such links are ignored. This option should be used with precaution, as loops in the directory structure can make VFind unable to scan all files.

--ignore-symlinks
If ths flag is given, VFind will not follow symbolic links pointing to regular files, otherwise, such links are followed and the file scanned as usual. Setting this may be useful e.g. to limit scanning to a locally mounted file system.

--keep-tmpfiles
The temporary files containing constituent files created during expansion are retained (normally they are deleted). This option is forwarded to UAD when UAD is enabled.

-p, --per-file
Per-file: display the number of possible virus infections for each file.

--pid
Print process id to stderr. See also --pidfile.

--quiet=num
This command provides a way of suppressing some of vfind's verbosity.

--quiet=0
The default behavior.

--quiet=1
Suppresses the "Enter the name of the file to be checked:" prompt and its two trailing newlines.

--quiet=2
Suppresses the "Checking file: filename" and its two trailing newlines.

--quiet=3
Suppresses all per-file output, including virus detections. Available only for applications linked with vfind as a library using callback functions to handle detections.

Thus, with --quiet=2, you can pipe a list of file names to vfind and there will be no per-file output unless a possible virus is found. There will always, however, be the final report of the number of files scanned and the number of possible infections found.

--quit
Used to exit vfind while in Interactive mode.

--rcf=file
Run Control File. Tells VFind to read additional command-line arguments from file.

--pidfile=file
Save process id to file.

--savepid=file
This option is available for backwards compatibility only, and is scheduled to be removed in a future release.

-ssr, --smartscan-read
This option will tell vfind to read a SmartScan input stream. There must be a process writing a SmartScan stream to vfind’s stdin.

-sst, --smartscan-types
SmartScan Types: Displays file types and any VDL’s skipped due to file type restrictions.

-nosst, --no-smartscan-types
No SmartScan Types: Disables skipping any VDLs due to file type restrictions. VDL file type restrictions will be ignored and all VDLs will be applied to all file types.

--stdin
Use the data on standard input as the file to scan. This will be treated as a file called "-".

--threads=num
Specify maximum number of threads (default=1). Only the multithreaded VFind executables vfind-mt and lvfind-mt support use of multiple threads.

--tmpdir=dir
Set the directory used for temporary files to dir. Without this option, the default temporary directory will be used. This option is forwarded to UAD.

-u, --unbuf
Make SmartScan Read unbuffered. Use of this option may cause a performance penalty, so it should not be used unless your application requires it.

--uad=opts
Note: this option is now on by default. Only use to override UAD command line options.

Tells vfind to run uad as a subprocess with the specified command-line opts which will be passed to uad in addition to -ssw and -s. Thus, uad will read file names from standard input and write smartscan output to vfind. Examples:

```
1. vfind --uad=
2. vfind --uad="-M -H3"
```

are equivalent to:

```
1. uad -s -ssw | vfind -ssr
2. uad -s -ssw -M -H3 | vfind -ssr
```

The --uad= option is mainly useful for multithreaded vfind-mt which will run a separate uad process for each thread. In this case there is no equivalent command using pipes. Example:

```
vfind-mt --threads=4 --uad=
```

will run 4 instances of uad, with each instance connected to a separate thread, and the file names from standard input will be distributed among the threads running in parallel.

--uad-disable
Disable advanced filetype detection and disintegration of files during scanning.

--vdl-list=file
Tells vfind to read the VDL library list from file instead of $VSTK_HOME/data/vfind /vdl.list. Must be the first command-line option if used because it must be processed before other options like --libon= which require the VDL library list file to already be read. Note that the VDL files specified in the VDL library list must be in the $VSTK_HOME/data/vfind/ directory.

--vdl-library-path=path
Use an alternate path to a directory where the VDL library is stored. All the VDL files described in vdl.list will be loaded from this directory.

--vdl=file
Tells vfind to read additional virus description codes from file.

--vdl0=file
Tells VFind to read additional virus descriptions from file, but without compiling them into the parallel search engine.

The parallel engine provides fast scanning but no control over the order in which the VDL patterns are applied. With vdl0, VDL rules are placed in a first-in-last-out queue, so the last rule specified is the first one executed, and vdl0 rules are always executed before the parallel search engine. So the --vdl0 option is useful when you have some set of VDL rules which you want executed in a guaranteed order, and this would usually be used in conjunction with the --#=1 option to stop scanning after finding one match.

--vdlc=file
Tells VFind to read additional case-insensitive virus descriptions from file.

Case-insensitive VDL constructs (i.e. ~"..." strings) are not compiled into the regular parallel search engine. But VDL files specified using the --vdlc option are compiled into a separate case-insensitive parallel search engine for faster processing.

--vdle=file
Tells VFind to read additional decrypted polymorphic virus descriptions from file. Used in conjunction with the --emu option. This option is still under development and its usage will be documented further in a future release.

--vdlE=file
Tells VFind to read additional Entry point virus descriptions from file. Used in conjunction with the --emu option. This option is still under development and its usage will be documented further in a future release.

--vdlm=file
Tells VFind to read additional meta virus descriptions from file. Note that meta VDLs match on the names of other VDL hits, not on the data being scanned. See the CVDL documentation for more information.

--vdl-data-file=file
Name of the file holding the compiled VDL data between VFind invocations. If not set, $VSTK_HOME/var/vdl.dat is used.

--max-vdl-data-size=bytes
Maximum size in Mbytes of the compiled VDL data file. If not set, VFind will make an estimate based on the total size of VDL files.

--rebuild-vdl-data
This option causes VFind to always rebuild the VDL data file, even when it is up to date.

--vexit
This option causes vfind to return a known value on exit. With this option vfind will return 0 if no viruses were detected. In the event that a virus has been detected, vfind will return 23. This functionality is useful when integrating vfind in a script or other program. The return values cannot be changed from the defaults (23 and 0).

--vlist
This option causes vfind to print to stdout a list of all viruses for which it currently scans.

--md5-disable
This option disables the md5-engine initialization procedure when vfind starts.

--#=num
Stop scanning a file after finding num viruses. Default is 1. Use 0 for no limit.

Note that # starts a comment in the Unix Bourne shell, so you may have to specify this option in quotes: '--#=0'

--End of Options: Signals to VFind that all remaining arguments are to be treated as filenames, even if they start with '-'.

Command Line Options – VFINDC

Version – [184]

```
--force-prompt
Option which allows the user to force the 'enter filename'
```

Version – [181-184]

--license=path
Specify an alternate path to the LICENSE file.

-c, --copyright
Display copyright information and then exit. All other options will be ignored.

-h, --help
Display usage message and then exit. All other options will be ignored.

-v, --version
Display version information and then exit. All other options will be ignored.

-e, --exit-on-error
Tells VFindc to exit immediately after encountering any kind of error or warning condition. Normally, VFindc prints a warning message and attempts to continue processing after encountering a non-fatal error, such as a syntax error in a VDL description.

--end
Used to exit VFindc while in Interactive mode.

--noscan=filename
This option provides a way to turn off scanning of particular files or directories. This may be useful when scanning disks containing e.g. quarantined viruses or virus scanner configuratuion files. If the name contains a '/' (slash) character, if is matched agains the full name of the scanning target, otherwise, it is matched only agains the final component, after the final slash. As of release 183, file exclusion patterns now support wildcards and new match rules. Pattern matches if: input equals pattern, absolute paths are equal, basename of filename matches wildcard pattern, or absolute path of filename matches wildcard pattern. Wildcards must be used with absolute paths only, not relative paths.

--noscans=file
This option provides a way to specify multiple noscan parameters in a file. File is a file that contains file or directory names as described in the noscan option.

--notell=virus
This option provides a way to turn off reporting of individual viruses. This may be useful if your site gets a lot of false positives for some particular virus due to the type of data you have. Virus is the name of the virus as it appears after "VIRUS ID: " in vfindc's output, for example: --notell="CVDL W32/Sircam.a"

--notells=file
This option provides a way to specify multiple notell parameters in a file. File is a file that contains valid virus parameters as described in the notell option.

--force-scan
Force scanning of the compiled vdl data file. This file is normally not scanned, as it would be very slow, and produce a large amount of false hits. The file is instead protected by a cryptographic checksum; any modifications to the file are reported as hits on the "Forgery.1" virus id.

--recursion-limit=number
Maximum depth to recursively scan directories, negative for unlimited (the default), or zero to not scan directories at all.

--follow-symlinks
If this flag is given, VFindc will follow symbolic links pointing to directories; otherwise, such links are ignored. This option should be used with precaution, as loops in the directory structure can make VFindc unable to scan all files.

--ignore-symlinks
If ths flag is given, VFindc will not follow symbolic links pointing to regular files, otherwise, such links are followed and the file scanned as usual. Setting this may be useful e.g. to limit scanning to a locally mounted file system.

-l, --localhost
Assume that the VFind Daemon has access to the same file system as the client. This allows the client to pass files by name to the daemon, rather than copying them over the network, which may affect performance somewhat.

-p, --per-file
Display the number of possible virus infections for each file.

--pid
Print process id to stderr. See also --pidfile.
--quiet=num
This command provides a way of suppressing some of vfindc's verbosity.

--quiet=0
The default behavior.

--quiet=1
Suppresses the "Enter the name of the file to be checked:" prompt and its two trailing newlines.

--quiet=2
Suppresses the "Checking file: filename" and its two trailing newlines.

--quiet=3
Suppresses all per-file output, including virus detections. Available only for applications linked with vfindc as a library using callback functions to handle detections.

Thus, with --quiet=2, you can pipe a list of file names to vfindc and there will be no per-file output unless a possible virus is found. There will always, however, be the final report of the number of files scanned and the number of possible infections found.

--quit
Used to exit vfindc while in Interactive mode.

--rcf=file
Run Control File. Tells VFindc to read additional command-line arguments from file.

--pidfile=file
Save process id to file.

--savepid=file
This option is available for backwards compatibility only, and is scheduled to be removed in a future release.

--stdin
Use the data on standard input as the file to scan. This will be treated as a file called "-".

--uad=opts
Note: this option is now on by default. Only use to override UAD command line options.

Instructs VFindD to run UAD as a subprocess with the specified command-line opts which will be passed to uad in addition to -ssw and -s. Thus, uad will read file names from standard input and write smartscan output to VFindd.

Note: UAD runs on the server with VFindD and not on the client.

--server=host[:port]
Connect to the VFind Daemon on the specified host and port. It is possible to use this option more than once, to spread the scanning load over multiple hosts. By default, VFindc connects to the local host on port 8081.

-4, --IPv4
Connect to the server using internet protocol version 4. If not specified, the client will use both IPv4 and IPv6, if available.

-6, --IPv6
Connect to the server using internet protocol version 6. If not specified, the client will use both IPv4 and IPv6, if available.

--vexit
This option causes vfindc to return a known value on exit. With this option vfindc will return 0 if no viruses were detected. In the event that a virus has been detected, vfindc will return 23. This functionality is useful when integrating vfindc in a script or other program. The return values cannot be changed from the defaults (23 and 0).

--vlist
This option causes vfindc to print to stdout a list of all viruses for which it currently scans.

--#=num
Stop scanning a file after finding num viruses, e.g. --#=1 will stop after finding 1 virus.

Note that # starts a comment in the Unix Bourne shell, so you may have to specify this option in quotes: '--#=1'

--End of Options: Signals to VFindc that all remaining arguments are to be treated as filenames, even if they start with '-'.

Command Line Options – VFINDD

Version – [181-184]

--license=path
Specify an alternate path to the LICENSE file.

-c, --copyright
Display copyright information and then exit. All other options will be ignored.

-h, --help
Display usage message and then exit. All other options will be ignored.

-v, --version
Display version information and then exit. All other options will be ignored.

-d, --dup-check
Tells vfindd to check for duplicate VDL names and definitions, and other potential problems. With this option enabled, duplicates will be reported as parser errors. Also, any VDL segment which starts with an offset range or .* operator will be reported as a parser error. An offset at the beginning of a VDL segment is allowed by the CVDL syntax, but does not make sense to use, and may cause the VDL to run very slow.

--trig-count
Lists the names of any VDLs that could not be indexed for speed, and also the trigger and run counts for all VDLs. In general, only simple VDL constructs can be indexed, and only constructs containing strings of four or more bytes. Having many non-indexed VDLs, or VDLs with excessive trigger hit counts, will make VFind run significantly slower.

-e, --exit-on-error
Tells VFindd to exit immediately after encountering any kind of error or warning condition. Normally, VFindd prints a warning message and attempts to continue processing after encountering a non-fatal error, such as a syntax error in a VDL description.

-ev, --exit-on-vdl-error
Tells VFind to exit immediately after encountering any kind of error related to processing of vdl files. Normally, vfind prints a warning message and attempts to continue processing after encountering a non-fatal error, such as a syntax error in a VDL description.

--emu-help
List options for polymorphic virus emulation. This option is still under development and its usage will be documented further in a future release.

--emu=options
Set options for polymorphic virus emulation. This option is still under development and its usage will be documented further in a future release.

--emu-config=file
Specify emulation configuration file. This option is still under development and its usage will be documented further in a future release.

-f, --foreground
Stay connected to the controlling terminal, do not fork a background process.

--user=username
Run vfindd as the specified user. This option is ignored on the MS Windows platform, and otherwise only available when vfindd is started by the superuser.

-i, --ignore-eof
Tells VFindd to ignore end-of-file and keep trying to read input files names or SmartScan input.

-4, --IPv4
Listen for connections using internet protocol version 4. If not specified, the server will use both IPv4 and IPv6, if available.

-6, --IPv6
Listen for connections using internet protocol version 6. If not specified, the server will use both IPv4 and IPv6, if available.

--jadevdl=file
Tells VFindd to load additional virus signatures from file. File contains VDL models for hostile java applets and applications.

--tmpdir=dir
Set the directory used for temporary files to dir. Without this option, the default temporary directory will be used. This option is forwarded to UAD.

--keep-tmpfiles
The temporary files containing constituent files created during expansion are retained

(normally they are deleted). This option is forwarded to UAD when UAD is enabled.

--libon=library, --liboff=library
Turn on/off library. VFindd will list the available libraries upon startup. Amiga and eicar libraries are turned off by default. Use --libon='*' to turn on all libraries.

-l, --localhost
Listen for connections on the local host loopback interface only. Without this, the server will listen on all interfaces.

--md5=file
Tells vfindd to read additional MD5 virus signatures from file.
--cbayes=file
Tells vfind to read Read additional cbayes data from file.

--noload=virus
This option provides a way to disable loading of individual VDLs. This may be useful if your site gets a lot of false positives for some particular virus due to the type of data you have. Virus is the name of the virus as it appears in the VDL file, for example: --noload="W95/Sircam.Worm"

--noloads=file
This provides a way to specify multiple noload parameters in a file. File is a file that contains valid virus parameters as described in the '--noload' option. For each line of the file, leading and trailing whitespace is stripped, then lines which are empty or start with '#' (i.e. comments) are skipped.

--pid
Print process id to stderr. See also --pidfile.

--quiet=num
This flag is available for backwards compatibility only, and may go away in a future release.

--rcf=file
Run Control File. Tells VFindd to read additional command-line arguments from file.

--pidfile=file
Save process id to file.

--savepid=file
This option is available for backwards compatibility only, and is scheduled to be removed in a future release.

-sst, --smartscan-types
SmartScan Types: Displays file types and any VDL's skipped due to file type restrictions.

-nosst, --no-smartscan-types
No SmartScan Types: Disables skipping any VDLs due to file type restrictions. VDL file type restrictions will be ignored and all VDLs will be applied to all file types.

--threads=num
Specify maximum number of threads (default=1). Only the multithreaded VFindd executable vfindd-mt support use of multiple threads.

--vdl-list=file
Tells vfindd to read the VDL library list from file instead of $VSTK_HOME/data/vfindd/vdl.list. Must be the first command-line option if used because it must be processed before other options like --libon= which require the VDL library list file to already be read. Note that the VDL files specified in the VDL library list must be in the $VSTK_HOME/data/vfindd/ directory.

--vdl-library-path=path
Use an alternate path to a directory where the VDL library is stored. All the VDL files described in vdl.list will be loaded from this directory.

--vdl=file
Tells vfindd to read additional virus description codes from file.

--vdlc=file
Tells VFindd to read additional case-insensitive virus descriptions from file.

Case-insensitive VDL constructs (i.e. ~"..." strings) are not compiled into the regular parallel search engine. But VDL files specified using the --vdlc option are compiled into a separate case-insensitive parallel search engine for faster processing.

--vdle=file

Tells VFindd to read additional decrypted polymorphic virus descriptions from file. Used in conjunction with the --emu option. This option is still under development and its usage will be documented further in a future release.

--vdlE=file
Tells VFindd to read additional Entry point virus descriptions from file. Used in conjunction with the --emu option. This option is still under development and its usage will be documented further in a future release.

--vdlm=file
Tells VFindd to read additional meta virus descriptions from file. Note that meta VDLs match on the names of other VDL hits, not on the data being scanned. See the CVDL documentation for more information.

--vdl-data-file=file
Name of the file holding the compiled VDL data between VFindd invocations. If not set, $VSTK_HOME/var/vdl.dat is used.

--max-vdl-data-size=bytes
Maximum size in Mbytes of the compiled VDL data file. If not set, VFindd will make an estimate based on the total size of VDL files.

--rebuild-vdl-data
This option causes VFindd to always rebuild the VDL data file on startup, even when the file is up to date.

--vlist
This option causes vfindd to print to stdout a list of all viruses for which it currently scans.

--md5-disable
This option disables the md5-engine initialization procedure when vfindd starts

--svsp-port=portnum
Specifies which port vfindd should listen to for SVSP connections. If not given, it will use TCP port 8081 by default. If the port is given as an empty string, vfindd will not accept SVSP connections.

--smtp-in-port=portnum
Specifies which port vfindd should listen to for SMTP connections. If not given, vfindd will not accept SMTP connections.

--smtp-out-port=portnum
Specifies which port vfindd should send SMTP results to. If not given, vfindd will use a port numbered one more that the one specified by --smtp-in-port.

--smtp-out-host=hostname
Specifies which host vfindd should send SMTP results to. If not given, vfindd will send data back to the originating host.

--vfdclam-port=portnum
Specifies which port vfindd should listen to for CLAMD connections. If not given, vfindd will not accept CLAMD connections.

--vfdclam-mailscan
Treat scanned files as email

MVFilter

The MVFilter program disinfects OLE documents (Microsoft Word, Excel and PowerPoint) from macro viruses (both VBA and Word Basic). It does this in the same way that all anti-virus programs disinfect macro viruses, by removal of all of the macros. The difference is that MVFilter was designed as a tool and most significantly, the way in which it removes the macro. Very often, anti-virus programs can detect a virus infection in a program that was disinfected by a different manufactures program.

This is because the virus is actually left intact but disabled. A few bytes of the virus code is changed so that the disinfecting anti-virus product will not detect the disabled virus. The most common way of disabling a virus is to flip the "erase" bit. The virus is still intact but will not execute since it was disabled. Since each manufacture determines which bytes to change these bytes may not overlap.

Consequently, Norton might detect a virus in a sample disinfected by McAfee. This problem is called a "ghost virus". MVFilter does not have this problem because it totally removes the virus by zeroing out the entire virus and virus macro name from the table of objects. There is nothing for another virus scanner to detect.

As such, MVFilter can be used for compartmentalization purposes in addition to its reactive disinfection role. As a compartmentalization tool, MVFilter can be used to proactively prevent all macro virus infections, including new unknown infections, by automatically stripping all macros from OLE documents as they enter a system.

This is a reasonable policy since it provides 100%, infallible protection in exchange for losing the ability to use macros. Since very few people use macros, this should not be troublesome. Users who need macros can use authorized methods to allow the macros they need while still preventing damage from unauthorized macros.

A second use for MVFilter is as part of a document warehouse archival and management system such as Documentum*. MVFilter has been successfully interfaced and operating with these systems for years. Generally, these systems manage hundreds of thousands to millions of critical documents. MVFilter automates the process of document baseline control by providing a consistent format, free of all macros, in addition to preventing attacks. MVFilter Command Line Options

Name

mvfilter - Microsoft Word (and other OLE2) Macro Stripper

Description

MVFilter examines its file argument and strips the VBA Macros or the Pre-VBA Macros from the file. The Original file is backed up in a file named with a .bak extension unless a different extension is specified with the -s argument. For example, foo.doc will be backed up as foo.bak.

Exit Values

MVFilter returns different error codes based on what actions have been performed or what errors have occurred. It returns:

30 if VBA Macros were disinfected.
31 if Pre-VBA Macros were disinfected.
40 if disinfection was not completed.
80 if Input File was not opened.
81 if File has an invalid header.
82 if Backup file could not be opened.
98 if an invalid command line option is entered.
99 if No valid license key was found.

In Addition to the codes listed above, the MVFilter returns the following values when the -t (test) option is supplied:

10 if VBA Macros were found.
11 if Pre-VBA Macros were found.
20 if No Macros were found.

MVFilter Command Line Options [180-184]

--license=path
Specify an alternate path to the LICENSE file.

-t
Tests the file to see what type of Macros are present.
NOTE: Disinfection is NOT performed using this option.

-v
This option displays the version.

-i
This option will display information.
NOTE: Disinfection is NOT performed using this option.

-s <extension>
Use extension instead of '.bak' for back-up files.

Example:

```
$ mvfilter -s aaa foo.doc
```

-n
This option will NOT write a backup file.

MVFilter Script Example

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#!/bin/sh
#
# Name: mvfilter.sh
#
# Copyright: 2003, 2004, 2005, 2006 CyberSoft Operating Corporation, All Rights
Reserved.
#
# Description: This is a general script that will demonstrate how to run the MVFilter
program.
#

PATH=/usr/bin:/usr/sbin:/bin:/sbin:$PATH
export PATH

VSTK_HOME=<vstk_home>
export VSTK_HOME

$VSTK_HOME/bin/mvfilter $1
```

UAD – Universal Atomic Disintegrator
[181-184]

Name

uad - Universal Atomic Disintegrator

Description

For each file argument, UAD (Universal Atomic Disintegrator) attempts to identify the file type. If the file is a composite file of a type it knows how to expand, uad attempts to expand the file and recursively process the sub-files.

Currently, uad expands BinHex, TNEF, tar, RAR, zip, zip2exe, compress and gzip files as well as extracting MIME and uuencoded enclosures in text files. Support for other archive and compression formats is available using external programs and the --external option.

Background

The UAD tool solves two difficult problems, identification and decomposition. Decomposition of a file to its smallest indivisible parts (universal atomic disintegration using classical Greek language meanings) is a difficult problem. First, the program must have infallible identification of the file in order to decompose it. This is not an issue for UAD, which identifies the file by direct examination of its contents. Most decomposition tools assume the contents of a file by its file name. If the file is named "xyz.zip" most decomposition tools will assume that the file is a "zip" compressed composite file. UAD does not make any assumptions. This also allows UAD to identify data in a byte stream where file name information may not exist. This is important in a network environment and increases the accuracy of anti-virus scanning by reducing stealth attacks and by adding additional known information for the scanner.

Secondly, decomposition is critical to proper pattern analysis. There is no value in virus scanning a compressed, composite or encoded file, since the encapsulating technology will hide the contents from examination. (There is an exception to this rule where the encapsulating technology is static and statically meaningful.) This is why UAD is able to decompose email, including attachments in uuencode and MIME formats. It is also able to decompose tar, gnu gzip, pkzip, zip2exe, 7zip, Unix compress and other formats. (See decompressor list from UAD.) UAD will continue decomposition recursively until every part of the file has been decomposed into a state that is known to be a terminus (atomic state), or has been decomposed into an unknown format. Most unknown formats are already in the atomic state or are moot.

A benefit of the UAD system besides its uses for virus scanning is its ability to decompose many formats of encapsulated files. This can save a lot of time when the file format is not directly compatible with the system on which it resides. The user just executes UAD with the file he wishes to decompose and UAD performs the rest. Unfortunately, many times when a user uses a tool other than UAD to decompose a file into its parts, the tool will place the decomposed files in multiple places on the system. UAD solves this problem by forcing the current working directory to be the top level directory for the purposes of decomposition. This allows a user or system administrator to have full control over the installation of a new program without "splatting" programs and data all over the system in an uncontrolled way.

As an intelligence gathering program, UAD can detect the actual identity of a file even though its file name may be a lie. This can be valuable information when doing an investigation. For example, xyzzy.txt is identified as a tar file with a dozen graphic files. This should be a red flag requiring additional investigation.

Newer versions of UAD also include automated decryption, which is important for the latest class of encrypted viruses which carry the encryption key as part of a text message. While it may not be conceivable to some that someone would decrypt a file, then execute the contents, these attacks have been very successful.

To view a partial list of expanders that UAD can decompose, use the command “uad --elist”. To see the list of file types that can be identified use the command “uad --tlist”.

UAD Expander List:

Please use the command-line option to see the current list of expanders. This list may be incomplete at any given time due to the frequent updates UAD receives.

UAD expanders: tar (on), Z (on), gz (on), bz2 (on), HTML (off), text (on), zip (on), TNEF (on), Hqx8 (on), SIT (on), ISO (on), cpio (on), RAR (on), CAB (on), 7zip (on), ext (off), XZ (on), LZMA (on), LZ4 (on), and Unix-style static libraries.

Usage

Input

UAD can take input in three ways.

1. UAD with a filename or list of filenames as arguments.
Example:

```
$ uad jdk1.1.tar.gz strerror.c
```

2. UAD reading a data stream from the standard input.
Example:

```
$ dd -bs=20 -if=/dev/rmt/0mn | uad -S
```

3. UAD reading a list of filenames from the standard input.
Example:

```
$ find /usr/local/bin -type f -print | uad -s
```

Output

Each input file is described with at least two lines written to standard output. Each line begins with a zero- based number indicating how many levels of expansion were needed to access the file. The first line reports the name, and the second reports the type. For example:

```
$ uad /bin/ls
```

might produce the following output:

```
0: Name: /bin/ls
0: Type: sticky 80386 COFF executable
```

If the file has sub-components, there will be a third line announcing the fact, followed by the components with a level number one greater than that of the file. For example:

```
$ uad archive.tar
```

might produce:

```
0: Name: archive.tar
0: Type: tar archive
0: Components...
1: Name: foo
1: Type: unknown
1: Name: bar
1: Type: unknown
```

Such expansion can proceed to any number of levels. For example:

```
$ uad mbox.tgz
```

might produce:

```
0: Name: mbox.tgz
0: Type: gzip compressed data
0: Components...
1: Name: unknown
1: Type: tar archive
1: Components...
2: Name: mbox
2: Type: text
2: Components...
3: Type: text fragment
3: Unnamed
3: Type: text (no enclosures found)
3: Unnamed
3: Type: text fragment
3: Unnamed
3: Type: text (8-bit)
3: Unnamed
3: Type: text fragment
3: Name: medal.gif
3: Type: GIF picture - version 87a 80 x 80, interlaced, 256 colors
```

In this case, UAD has extracted attachments from email messages. The components of type 'text fragment' are typically mail messages and/or MIME headers that delimit the other components. These are put in mini-files of their own. Attachments that are themselves text files are, of course, recursively scanned. If nothing is found in them, this is announced as type 'text (no enclosures found)'. If, in the above example, UAD had been invoked with the -w option (see above), it would have also announced:

```
uad: Wrote file `medal.gif'
```

SmartScan

UAD is a SmartScan compliant tool. Specifying the -ssw option to UAD will cause UAD to produce a SmartScan stream as output. This stream can then be redirected to another SmartScan compliant tool, such as VFind, and processed further. For example:

```
$ find /export/home -type f -print | uad -s -ssw | vfind -ssr > REPORT
```

UAD Command Line Options
[181-184]

--license=path

Specify an alternate path to the LICENSE file.

-c, --copyright

Display copyright information and then exit. All other options will be ignored.

-h, -?, --help

Display usage message and then exit. All other options will be ignored.

-v, --version

Display version information.

--elist

List expanders.

--tlist

List recognized file types. The actual file types reported will vary depending on the file data, with %c and %s replaced by a character or string from the data, and %o, %d, %ld, and %x replaced by an integer. If an archive appears to be corrupted the "malformed" modifier may be reported, e.g. "malformed zip archive file". The ‘magic’ types are capable of reporting additional sub-type information which is appended to the basic type and is listed with extra indentation in the uad --tlist output. Some types and sub-types are listed more than once because internally they are detected from different byte sequences in the data which represent the same type.konami buyout

--enable <expander>

Enable expander. Use --enable '*' to enable all expanders.

--disable <expander>

Disable expander. Use --disable '*' to disable all expanders.

-e <prog>, --external <prog>

Use external expander for unknown archive types. The user must provide the external expander program. An example uad-helper Unix sh script is provided to handle cpio, bzip2, and StuffIt formats.

--cpu-time-limit <value>

This option causes uad to limit the time it spends executing to <value> seconds. The default is 3600 (one hour).

--file-size-limit <value>

This option causes uad to limit the the maximum size of its temporary files to <value> KibiBytes (KiB). The default is 1048576 (one GibiByte, GiB).

-H, --HTML2text

HTML to text conversion: this option will cause UAD to produce a text version of any components which appear to be HTML documents.

-H1, -H2, -H3

HTML to text conversion, with higher detection level. Default is to only skip white-space before recognizing the first HTML tag. With -H1 any initial non-HTML is skipped. With -H2 unrecognized tags are also skipped. With -H3 <a and <img tag contents are also extracted as text.

-k, --keep-tmpfiles

The temporary files containing constituent files created during expansion are retained (normally they are deleted). These files are announced as each input file is scanned when this option is specified.

-l, --follow-links

Symbolic links are followed. First, the link file is described, then the file it points to is scanned.

-L, --transparent-links

Symbolic links are followed transparently. The data in the file linked to is used to describe the link file.

--magic-after <file>

Use the rules in file to recognize file formats, after all built-in or previously loaded rules have been exhausted. The rules file format is similar to that used by the file utility, as described in magic.

--magic-before <file>

Use the rules in file to recognize file formats, before attempting any builtin or previously loaded rules.

--magic-replace <file>
Use the rules in file to recognize file formats, instead of all builtin or previously loaded rules.

-m, --malformed
Keep malformed files. Normally UAD does not keep any partial output or extracted components if an archive or encoding is malformed. This option will cause UAD to keep such files, e.g. they will appear on SmartScan output.

For use in conjunction with SmartScan output piped into VFind, the user should also specify the uad -p option when using -m, otherwise trailing content in a malformed archive may not be processed.

-M, --Mail
Set Mail mode: This option makes UAD report type "top level mail header" for the first component of input files. Input files are treated as text and are assumed to be mail files including headers.

-n, --no-expansion
Do not expand composite files.

--no-recursion
Do not expand files recursively.

-p, --provide-all
Provide All Files: Normally UAD does not provide output files for archives which have been expanded into component files. This option will cause UAD to also provide the archive files.

-p0, --provide-top
Provide Top Files: Normally UAD does not provide output files which are simply copies of input file archives which have been expanded into component files. This option will cause UAD to also provide the top-level input archive files.

-pid, --pid
Print process id to stderr.

-q <string>, --quote <string>
Write a literal string on SmartScan output.

--recursion <value>
Recursion Limit: This option causes UAD to limit the level of recursively expanded files to <value>.

-s, --stdin-file-list
Read the names of files to scan from standard input, one per line.

-S, --stdin
Use the data on standard input as the file to scan.

-ssr, --smartscan-read
This option causes uad to read files from a SmartScan stream on standard input.

-sst, --smartscan-types
Used with -ssw, this option causes UAD to output expanded archive names and types. For use in a pipeline with VFind or other post-processing, this allows detection and blocking based on archive file names and/or types without having to use the uad -p option. With -p the unexpanded archive contents are provided; with -sst only the archive name and type is provided, and a single zero byte is written for the archive contents.

-ssw, --smartscan-write
Write a SmartScan Stream to the standard output. This option is useful when interfacing with other SmartScan compliant tools such as VFind.

-t dir, --tmpdir <dir>
Set the directory UAD uses for its temporary files to <dir>. Without this option, UAD will use the default temp directory appropriate to the operating system.

-T, --Text
Set Text mode. This option forces UAD to treat input files as text.
-w, --write-files
Constituent files for which meaningful names are available (from, e.g., an archive or MIME message) are saved with those names in the current directory. Sub-directories are created as needed.

-z, --zeroing-off
Do not zero out the temporary files before unlinking them. This speeds things up a little, but introduces a potential security hole.

--

End of Options: Signals to uad that all remaining arguments are to be treated as filenames, even if they start with '-'.

Cryptographic Integrity Tool (CIT) [181-184]

Description

CyberSoft, Inc.'s Cryptographic Integrity Tool is part of the VFind Security ToolKit. When fed a stream of files, a cryptographic hash code is generated for each file and compared to a previous signature for that file stored in a database (by default, cit.db; this can be overridden with the -db option). A list of new and modified files is written to standard output. A report of new, modified and deleted files is placed in cit.rpt (by default; again, see the -db option).

Primary Features

The Cryptographic Integrity Tool uses hashes (SHA3, MD5) to track files within a system and report changes, additions, deletions, and duplications within that system. It can also produce instant hashes on files or input streams. CIT generates both a database (.db) file and a report (.rpt) file. Those files contain user-friendly tables with useful information about the status of your baseline.

Hash Code Creation

The CIT tool produces a database of cryptographic hash values for every file it is directed to manage. In normal operation, this is every file on the system with the exception of special files like /dev, FIFO and proc files. By generating a cryptographic hash database, CIT creates a baseline. This database allows for a faster virus scan by feeding a report to VFind that confirms which files have been changed, or have remained the same. From here, VFind only needs to scan the files that have changed, saving precious time. CIT not only allows for a faster virus scan, but also a faster response from an organization after an attack or if unauthorized use of the network is occurring. CIT functions excellent as a Baseline Configuration Control (BCC) tool to assist in these functions.

Choose Your Hash

As of CIT version 5.0.0 and/or VSTK 183, CIT allows you to choose between MD5, SHA3-256, SHA3-384, SHA3 512 hash functions. The current default hashing option is SHA3-256, previous to VSTK 183, the default hashing option was MD5

File Analysis

With CIT, evidence of addition, deletion, modification, or duplication of files is easy to detect. When a file is added or deleted, CIT will note the change and reflect it in the report file once the scan is complete. This enables the user to pinpoint any corrupted or suspicious files instantly and respond accordingly. The simplicity of the program, along with the ease of use, makes CIT a powerful tool with a wide array of applications.

Track User Behavior

CIT is also an excellent personnel monitoring tool. A trained security officer can determine a great deal about the user activity on a system by examining CIT reports. It is usually possible to tell if a user is not doing their job or is attempting to tamper with the system baseline configuration.

Media Integrity

CIT can ensure that the contents of a tape, disk or email attachment are what they are supposed to be, by referring to a secure database of known CIT hash values, it is possible to determine if the contents have degraded or been modified. This is extremely valuable when dealing with programs or critical documents distributed over a network or archived in offline storage. It is a 100% reliable way of knowing what was sent, was what was received, and what the user thought was on removable media, is in fact, what is on the media. The CIT hash values are one way traps. It is not possible to determine the contents of a file by knowing the hash value unless the user already has a copy of the exact contents. This means that it is possible to securely distribute hash values over open channels such as fax or Internet for even the most sensitive of documents.

System Maintenance and Customer Support

Generally, the end user does not know why their system is not working but they are sure they were not the cause. Using CIT, a system can be baselined when it is installed and updated. At anytime in the future if the system fails, the help desk can run CIT. CIT will reveal every file which has been added, deleted, modified, marked as a having dangerous content, or is a duplicate of other files already on the system. This can reduce the time for diagnoses to about 15 minutes while the duplicate files feature can save significant amounts of disk space. When a system is malfunctioning the engineer can review the on-system CIT reports and run an audit against the CIT baseline database. CIT will identify every way in which the system no longer conforms to the baseline.

Forensic Evidence

Every activity on a system leaves fingerprints. CIT can be used as a data reduction method to find these fingerprints quickly. This is especially useful when dealing with insider attacks that may be performed by disgruntled employees. In a network CIT can be used to ensure the contents of a file transmitted to another system is what was received. If the hash code is transmitted via a secondary channel then it can also be used to reveal third party tampering by a “man in the middle” attack. Using CIT, create a hash of the entire contents of a removable media source. That hash value can be used as the serial number of the media and will also verify that the contents have not degraded.

CIT Use Case Example:

(Note this report is not real. It was formatted for this document and is representational.)

```
Added Files:Modified Files:Deleted Files:
/home/george/abc.c/home/george/main.c/home/harry/a.c
/home/george/def.h/home/george/a.c/home/harry/c.c
/home/harry/xyzzy.c/home/george/b.c/home/harry/1.c
/home/harry/567.c/home/george/c.c/home/harry/2.c
/home/harry/xyz.jpg/home/harry/123.c/home/harry/3.c
/home/harry/report.txt/var/adm/sulog
/var/log/uucp/Log/var/mail/horse

Duplicate Files:
/home/george/abc.c with /home/harry/567.c
/home/george/def.h with /home/harry/xyz.jpg
/home/george/main.c with /home/harry/report.txt
```

When a security analyst views the example just given, the first call made should be to have Harry removed from the building. The second call should be to convene an emergency meeting. Why? A little more information needs to be gathered but it is now known that Harry deleted all of the work he did, copied work from George, which he should not have done, and was attempting to hide the fact that he copied files from George. It can also be seen that George has been very busy doing his assigned task.

At first glance it appears that Harry is at worse a spy, or a malcontent, or at best, someone that is breaking the rules by nosing around the system and attempting to hide the fact. A detailed examination of the file contents along with other standard security investigation methods will reveal if this is a false alarm. If it is indeed a false alarm, then Harry is still showing behavior that is undesirable.

Another piece of intelligence that was picked up on this report is the fact the /var/log/uucp/Log file was created. This means that someone on that system was running the uucp command. The uucp command was the primary method of moving email and files between systems prior to the invention of the network. It also allowed users to execute commands on remote systems using email. Since the use of uucp program on this system was unauthorized it appears to have been used as a back channel. It should also be noticed that a mail file was modified for a user called horse. Since there is no authorized user called horse, this raised a serious red flag.

The fact that it was modified means that it existed in the past and additional information can be gathered by viewing old CIT logs and system backups. This is also an indication of what the unauthorized uucp connection was doing and the fact that another channel besides uucp may be in use since the uucp log was new while the mail file was already in existence.

Yet another piece of information is that the /var/adm/sulog was modified. The sulog file is a record of all attempts by users on the system to execute the su command. Each time su is executed, an entry is added to the sulog file. The su command gives someone the ability to switch user identities. It is often used to become the system super user.

The reported changes in system log files reveal which processes have been executed over a known time frame. This is in addition to identifying saved email, edited files, database files that were modified, ensuring that files that must not exist (such as known Trojan horses or obsolete programs / documents) do not, and generally determining what is happening and not happening in the system.

One of the most interesting things that is also revealed by this report is timing. The system date and time logs are of little value since they can be easily modified. On the other hand, if CIT is run nightly – or over any predetermined interval – it can be determined that modifications to the file occurred within the period between runs. Knowing the actual time window in which something happened is critical when doing an investigation.

The file /var/log/uucp/Log was new but /var/mail/horse was modified; this indicates that this problem may be longer standing than it would appear at first glance. Attempts to use the su command combined with CIT's other indicators can indicate that hacking attempts may have been in progress.

In fact, a starting guess would be that Harry already had a back door and was using an smtp transport to send/receive email but decided to add the uucp system.

All of this information is based on the results of the CIT run shown above. It should also be noted that all of the past CIT logs are available on the backup tapes! A wealth of information on Harry's past activities can now be learned with timing nailed down to 24 hour windows.

Recap of why Harry needs to be removed from the building pending investigation:

1. Harry deleted all of his work. This was not authorized, and destruction of company property is normally a firing offense.
2. Harry made unauthorized copies of George's work. This is suspicious.
3. Harry tried to hide the fact he made copies of George's work. This is very suspicious and dishonest.
4. Someone was using an unauthorized back channel communications tool on the system. This is very suspicious.
5. Someone has fully or partially penetrated security on the system as evident since they could execute uucp.

Customer Case Study

A customer case study demonstrates a typical use of CIT after a break in. This part of the document was written by the customer with light edits for brevity and clarity.

As an Internet Administrator whose main servers are Sun Unix systems, I always seem to be fighting that never ending battle of keeping crackers off my computers. Even with the security patches, router access control and just simply phoning providers after finding someone suspiciously banging on our door, someone got in.

One morning I tried to log into our main server after a long weekend and found that my terminal was booby trapped. No matter what key I hit garbage was displayed. I did a telnet into the server from another computer and looked around. Using CIT I found that the /etc/rc boot and other files were modified. If I had rebooted the rc files would have taken effect giving the intruder more control. I found all of the modified files using CIT and using a backup tape replaced them. Using CIT it took 10 minutes to figure out where the intruder came from. I must say that CIT has really made my life administering my systems easier. Besides saving my skin that day and using it other times to check for suspicious changes, I look at the output every day to give me an indication of what is going on in the whole system. Great tool!*

Input

CIT can take input in three ways.

1. cit with a filename argument.
In this mode cit will generate a cryptographic signature for that file and write it to stdout. No database is created or updated.

Example:

```
$ cit HelloWorld.java
```

2. cit reading a data stream from the standard input.
In this mode cit will generate a cryptographic signature for the data stream and write it to stdout. No database is created or updated.

Example:

```
$ cat BigFile.tar | cit -S
```

3. cit reading a filename list from the standard input.
This is the default mode of operation. In this mode, cit accepts a list of filenames on standard input. A cryptographic signature is generated for each file and compared to a previous signature for that file stored in a database (by default, cit.db; this can be overridden with the -db option). A list of new and modified files is written to standard output. A report of new, modified and deleted files is placed in cit.rpt (by default; again, see the -db option).

Example:

```
   $ find / -type f -print | cit
```

CIT Command Line Options

Version – [183-184]

```
--md5
Use MD5 hash function.

--sha3-256

Use SHA3-256 hash function. (This is the default)

--sha3-384
Use SHA3-384 hash function.

--sha3-512
Use SHA3-512 hash function.
```

Version - [181-184]

--license=path
Specify an alternate path to the LICENSE file.

-c, --copyright
Display copyright information and then exit. All other options will be ignored.

-h, -?, --help
Display usage message and then exit. All other options will be ignored.

-v, --version
Display version information and then exit. All other options will be ignored.

-S, --stdin
Display the cryptographic hash of standard input on standard output.

--cleanup
Totally remove all cit work and database files. Suitable for use when a totally new database is wanted, or to release the disk resources that cit is using. Use with caution!

-db databasename, --database-name databasename
Use databasename as the base name for the database. The database itself would be named databasename.db, the report file, databasename.rpt

--dbpath databasepath
Use databasepath as the directory in which the database is located. Default: Current directory.

--overwrite
Ignore any pre-existing database; always create a new database.

-d, --dupcontents
Reports (usually in cit.rpt) all of the files that have duplicate hash values.

-do, --dupcontents-only
Duplicate contents only: cit only creates a cit.dup file based on the existing cit database listing all files that have duplicate hash values.

-f, --file-changed
Hash the file and compare the hash value with the value found in the cit database. Report whether the file has changed or not.

-nb, --no-backup
Do not retain the old database after the run (typically kept in cit.db0).

-nc, --no-cleanup
Cit will not remove any tmp files.

-nr, --no-report
Do not generate the cit.rpt report file.

-nu, --no-update
Report changes in the file system, but do do not update the database.

-t tmp, --tmpdir tmp
Temp Directory: cit will create tmp files in tmp. If this option is not specified, cit will use the appropriate temp location for the system.

-uo, --uniqcontents-only
Unique contents only: cit only creates a cit.uniq file based on the existing cit database listing all files that have unique hash values.

--
End of Options: Tells cit that all remaining arguments are to be treated as filenames, even if they start with a dash.

Avatar Baseline Configuration Tool
[183-184]

Preface

Modern enterprise computers are valuable targets, and their value grows as they become more involved in the business cycle. If an administrative system can be exploited by malicious code, the entire network is put at risk. When that happens, malware that may have cost only a few hundred dollars to develop can control hundreds of millions of dollars worth of data and hardware. Avatar provides a solution to this problem by providing proactive and reactive system defense capabilities.

Usage

```
avatar create [options] [database] [config file]
avatar [add/delete] [options] [database] [files to add/delete]
avatar [check/correct/update/export] [options] [database]
```

**Note – The update and export commands have been added as of VSTK 184.*

Description

Avatar maintains the state of a file system in a baseline, stored in a user-defined location, and uses that baseline to check the contents of file systems.

Avatar can be used:

1. Interactively by working in command line
2. Automatically in a mode invoked by scheduling processes (such as UNIX cron)
3. Responsively as part of a stimulus from other packages (e.g. malware detection)

Avatar protects files by recording the following:

File name
Location
Permissions
Ownership
Cryptographic signature of file content
Actual content

When Avatar determines that one or more of these attributes is incorrect, it can restore the attributes to their recorded state by calling its correct function, or save the new file system state by calling update.

Primary Features

The most important function of Avatar is response. If the system baseline is modified, for any reason, it will be detected by Avatar. The way Avatar reacts to a system change can be customized depending on what the user wants out of the tool. The value of Avatar's response system is that it enforces discipline by an objective, automated process which can execute as frequently or infrequently as the user desires. Avatar security policies can be maintained on a file by file basis, for an entire machine, or for an entire network.

The ability to maintain the baseline configuration also provides extensive immunity to new unknown software attacks within the baseline. If a binary or script virus infects a file, then Avatar will simply restore the file to it's original form. This effectively destroys the virus, does so without explicitly identifying it as a dangerous file in the first place, and prevents file degradation from recurrent manipulations.

Baseline Configuration

If a binary or script virus infects a file, then the file will be overwritten by the baseline version of the file. One of the advantages of the Avatar system for internet based systems such as web servers is that shortly after a hacker modifies a web page, the Avatar system can be automatically invoked and restore the damaged page. If Avatar is run dozens of times per day then the hacker would quite literally have to break in and modify the system dozens of times per day. This is a huge incentive to leave the system alone. If the system baseline configuration is modified, for any reason, it will be detected by Avatar and returned to the correct configuration.

Intrusion & Detection

Intrusion is defined as gaining unauthorized access to a system, or as unauthorized changes made to a system by an authorized user. When an unauthorized person gains access to a computer, their actions will always be dictated by their goals. If the intruder is a passive reader, then the activity will be captured in the system logs. If the intruder is using the system as a platform for further attacks, then the intruder will want to ensure future access, and will have to change the baseline configuration. Modification of the system such as changing permissions, insertion of Trojan back doors, modifications of the baseline, or destruction of critical system files, can all be detected and corrected using Avatar.

Response

Avatar can react quickly and restore systems to their baseline configuration once an intrusion is detected. If automatic checks are being performed, Avatar can also proactively detect and correct unauthorized changes prior to major system failure.

If a binary or script virus infects a file, then the file will be deleted and replaced. This effectively destroys the virus. When a virus infects a file, it modifies it. In the process of infecting the file, it is common for the file to be damaged. Or worse, the exploit is a new unknown viruses that cannot be disinfected. None of these problems exist with Avatar since a captured copy of the original file is used to overwrite the infected file. This also works for all forms of software attacks in baseline configured programs, not just viruses or hacker attacks.

Avatar Database

Multiple Avatar Databases can exist and be invoked in any order necessary. This allows Avatar to be used for multiple purposes, including rapid system software update distribution and configuration for thousands of workstations without the need for personnel to visit them. System customization can also be accomplished by a smaller Avatar database which can be stored locally or at a central location.

Example

The first step in setting up Avatar is to create the configuration file. After that, the user should run the create command to do the initial baseline creation:

```
avatar create /home/user/database.tar /home/user/Desktop/config.txt
```

After that initial setup, files can be added or removed from the baseline as the user pleases:

```
avatar add /home/user/database.tar /home/user/Documents/Example/NewFile.xml
avatar delete /home/user/database.tar /home/user/Documents/Example/OldDirectory/
```

The above commands would add an excel document to the database called NewFile.xml, then remove all files from the OldDirectory folder from the database. Note that this would not delete the files from the system, but rather it would simply ensure they are no longer being tracked by Avatar. Once all the files the user wants to be included in the baseline have been added, the check, correct, or update commands can be manually or automatically run to analyze and maintain the baseline system:

```
avatar check /home/user/database.tar
avatar correct /home/user/database.tar
avatar update /home/user/database.tar
avatar export –json /home/user/database.tar
```

Commands & Options - AVATAR

Commands - [184]

```
--license=path
Specify an alternate path to the LICENSE file.

update – Updates any changes to the file database.
export – Exports the Avatar database as either a json object file or an xml file.
```

Options

```
--json, -j
Only usable with the export command. Designates the command to export the database as a
json object file.
--xml, -x
Only usable with the export command. Designates the command to export the database as an
xml file.
--license=<path>
This command allows the user to enter a custom path to a VSTK license. Should only be
used if the user has moved their license to a non-default location.
```

Commands - [183]

```
create – Creates the baseline using the config file to determine the file system to add.
add – Takes the input file/file system and adds it to the baseline. Ignores files
already                         inside of the database.
delete – Removes file/file system from the baseline.
check – Checks contents of baseline against current state of file system and reports any
changes that have been made.
correct – Checks the baseline Avatar Baseline Configuration Tool
```

Usage - [181-182]

(e - EXISTENCE) The existence rule states that the file(s) must exist. It does not infer any other rules. This is extremely valuable for files that must exist but whose contents change constantly. Two examples of files of this type are log files and password shadow files.

(p - PERMISSIONS) The permissions rule states that the permissions of a file must not deviate from the baseline configured permissions. Generally all system command, log and configuration files have critical permission settings that must not change. This can be combined with the "e" and "o" rules to provide maximum protection for files whose contents need to change over time.

(o - OWNERSHIP) The ownership rule states that the owner of the file must not deviate from the baseline configured ownership. For example, the ownership of the password shadow file can be used as a back door into a system , while ownership of a log file can be used to stealthily erase evidence of activity on a system.

(s- CRYPTOGRAPHIC SIGNATURE) The cryptographic signature also known as a hash value states that an alarm should be activated if the contents of the file change but no further action should take place. This is extremely useful when used within other programs and when attempting to catch hackers. An additional use is to allow a script or program to verify that a critical file conforms to the baseline configured contents without overwriting any deviations from the baseline.

(c - FILE CONTENTS) The contents rule states that the contents of a file must not change. If the contents change for any reason, the file is overwritten with the correct baseline configured contents.

(!) A rule exists to force a pathname to not be baselined. This is most valuable for scratch file areas such as the Unix "/tmp" and "/var/spool" areas.

(R – RECURSIVELY) The opposite also exists which forces Avatar to baseline the entire path recursively.

(E – EVERYTHING) Finally a shorthand rule exists which has the same operation as selecting the "eposcR" rules.

Commands - [181-182]

Avatar has eight modes of operation: create, update, iupdate, add, delete, configuration, check, and correct.

The first six commands create, examine and modify the baseline. The last two use the baseline to check and/or correct individual files, directory hierarchies, or whole file systems.

Each of these commands is invoked by invoking the avatar program with the command name as its first argument. For example:

```
$ avatar create foo bar n.cfg
```

Most commands take other arguments as well which come after the function name (i.e., foo, bar and n.cfg above). They will be explained in the entry for each command.

In general, though, most commands have root and baseline as their first two arguments. These are the root of the file system to be used and the name of the baseline directory to be used (or created) respectively.

The Configuration - [181-182]

Everything in a baseline maintained by avatar is controlled by the baseline's configuration. The configuration describes what files and directories to check and what attributes to check for each of them. The configuration is initialized by the create command from a configuration file.

An example of a possible real configuration is to protect the entire file system with the exception of /home and /logs. The configuration file used to initialize this configuration via the create command could look like:

```
E /
! /home
! /logs
```

The configuration can also allow the system to record varying sets of attributes for different files and directories. It also allows the specification of a directory to be recursively baselined with certain files and sub-directories excluded or recorded with different attributes.

Configuration File Format

The configuration file is used by create to control the initial configuration of the baseline. Once created, the configuration is part of the baseline and can be modified by add or delete. It can be queried by the configuration command.

Each line of the configuration file contains a pathname to be baselined which can be either a file, special file, or a directory.

Each pathname is prefixed by a list of the attributes to baseline, followed by a space. The attributes are represented by single letters as follows:

e = existence
p = permissions
o = ownership
s = cryptographic signature
c = file contents
! = do not baseline the pathname
R = baseline the recursive contents of directories
E = everything (eposcR)

Thus, a configuration file that consisted of the single line:

```
E /
```

baselines the entire file system. Individual files or directories listed can form exceptions to the attributes listed for a recursively baselined directory. So, if the above configuration file had the line:

```
!R /var
```

then /var and its contents would not be baselined.

The Commands

The description of each command begins with its invocation. The parameters for all commands are order dependent and are labeled with names in italics. Optional parameters are enclosed in square brackets.

create

The create command baselines all the pathnames listed in configfile, with root prefixed, placing the baseline database in the baseline directory.

If provided, the altbaseline parameter specifies the location of an alternate baseline which is a secondary source of information used by the check and correct commands (See the extended discussion of this below). There can be one alternate baseline for each baseline. It can be set via the create, update or iupdate commands.

Note that a baseline may consist of a pointer to an alternate baseline, such as on a CD-ROM or an NFS mounted drive.

If create has no problems, it will exit with a status of 0; otherwise, it will exit with a non-zero status.

```
$ avatar create root baseline configfile [altbaseline]
```

update

The update command updates the baseline in the baseline directory with information about the pathnames in the configuration (with root prefixed). If altbaseline is provided, avatar uses it as a second source of information. Exit status is 0 if there are no problems.

```
$ avatar update root baseline [altbaseline]
```

One example of a use of the update command would be to modify the baseline due to the installation of a product upgrade on the system when the files of the upgrade reside in an already configured (in the configuration) directory. Another use is to change the alternate baseline without updating any other baseline contents via the invocation:

```
$ avatar update / , , altbaseline
```

The commas indicate that file information is not to be updated.

iupdate

The iupdate command performs the same function as update except that the baseline administrator is interactively prompted for a yes/no response on each file that needs updating. Only files for which a `yes' response is given will have their baseline information updated.

```
$ avatar iupdate root baseline [altbaseline]
```

Avatar configuration root baseline

configuration

The configuration command outputs a report to standard output describing the current configuration of baseline. This is in a form suitable for use as a configfile argument to the create command. The root argument to this program has no effect and is only present for purposes of consistency. An example of using this command to output the entire configuration of a baseline residing in the directory /baseline is:

```
$ avatar configuration / / baseline
```

check

The check command checks the directory structure based at root (or the individual file, with root prefixed, if specified) against the information in the baseline database in the baseline directory. Exit status is 0 if there are no problems. Exit status is 1 is there if a discrepancy between baseline and the file system. In that case, a report of the discrepancies is written to standard output. The exit status is greater than 1 in the case of other problems.

```
$ avatar check root baseline [file]
```

correct

The correct command attempts to restore the directory structure based at root (or the individual file, with root prefixed, if specified) to the state described in the baseline database in the baseline directory. Correct automatically invokes check for the purposes of determining which corrections are needed. Exit status is 0 if the restoration is completely successful and a report of the corrections made is written to standard output. Otherwise, the exit status is non-zero and a report of the problems encountered is written to standard error.

```
$ avatar correct root baseline [file]
```

Avatar add root baseline attributes path

add

The add command allows the addition of files to the baseline. It also updates the corresponding file information in the baseline. attributes is an attribute list as described for the configuration file above. An example of its use is adding protection to a disk drive that was recently added to the system.

```
$ avatar add root baseline path
```

delete

Similarly, the avatar delete command allows the deletion of files from the baseline.

```
$ avatar delete root baseline path
```

Which is also equivalent to:

```
$ avatar add root baseline ! path
```

which uses the ! flag to indicate that the path is not to be baselined. Information about path is removed from baseline and the configuration is modified so that subsequent updates to the baseline will not include information about path.

LBT & LBH
[179, 180, 181, 182, 183, 184]

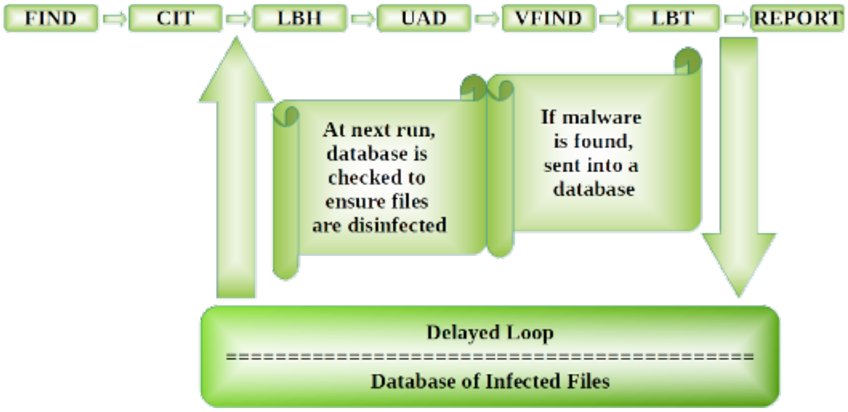
These tools are critical when using the CIT Cryptographic Integrity Tool in conjunction with VFind.

The Loop Back Head (LBH) and Loop Back Tail (LBT) programs provide a method of performing locked loop back logic for the purpose of ensuring that no target files are missed when virus scanning utilizing CIT for target reduction. These loop back target files are identified as files which were known to be infected at the last run.

Explanation:The CIT tool produces a target list of files which have been modified or added to the system. This target list is then used to scan the files by VFind for viruses. If the administrator chose to ignore the warnings of infected files from a system scan, then CIT may not report the file as changed the next time it runs. This is a serious problem since it could leave an infected file undetected. This problem is resolved by storing the file name of the infected target files in the loop back database called lb.db. The lb.db database is a simple text file and can be read by humans. When Loop Back Head runs and before it accepts a target file list from CIT it reads the lb.db database and transmits its contents to standard output for scanning by VFind. This ensures that files which were previously known to be infected are scanned again and are in fact scanned before new files.

In addition to these features, the LBT program provides the "restrict" feature which allows the users to eliminate the use of the "grep" command when parsing output from VFind for the purpose of data reduction.

Diagram of Data-flow for LBH and LBT


FIND
CIT
LBH
UAD
VFIND
LBT
REPORT
At next run, database is checked to ensure files are disinfected
If malware is found, sent into a database
Delayed Loop
==
Database of Infected Files

Name

lbh - CyberSoft's LoopBack Head Tool

Description

CyberSoft, Inc.'s LoopBack Head Tool is part of the VFind Security ToolKit. LBH is used along with LBT to ensure that unchanged but infected files are re-analyzed by VFind. LBH will read filenames from a database. This database can be created any way the user wishes, so long as it is a flat text file that contains one filename per line. LBT is provided as part of the VSTK to use VFind output to create a database suitable for use with LBH. After reading from the database, LBH will read data from the standard input stream and write it to the standard output stream.

Usage

Input

LBH can read filenames from up to two places.

LBH will read a database created by a program such as LBT.

LBH will also read filenames from the standard input stream. LBH will read from stdin regardless of whether or not it successfully parsed a database. This behavior is called passthru mode.
Example:

```
$ find / -type f | cit | lbh -db=lb.db | uad -s -ssw | vfind -ssr | lbt -db=lb.db -a
```

LBH Command Line Options [180-184]

-c, --copyright
Display copyright information and then exit. All other options will be ignored.

-h, -?, --help
Display usage message and then exit. All other options will be ignored.

-v, --version
Display version information and then exit. All other options will be ignored.

-db=name, --database=name
Use name as the database. Without this option, lbh will enter passthru mode (read from stdin and write to stdout).

Name

lbt - CyberSoft's LoopBack Tail Tool

Description

CyberSoft, Inc.'s LoopBack Tail Tool is part of the VFind Security ToolKit. LBT is used along with LBH to ensure that unchanged but infected files are reanalyzed by VFind. LBH will read filenames from a database. This database can be created any way the user wishes so long as it is a flat text file that contains one filename per line. LBT is provided as part of the VSTK to use VFind output to create a database suitable for use with LBH. LBT will read data from the standard input stream and write it to the standard output stream. LBT will create a database if the -db=name option is specified.

<u>Usage</u>

Input

LBT reads output from VFind and depends on VFind's uniform chevron output.
Example:

```
$ find / -type f | cit | lbh -db=lb.db | vfind | lbt -db=lb.db -a
```

Output

LBT can be used with the -r=<value> option to choke VFind's output. The choke method is discussed in detail in the VFind man page.

LBT Command Line Options

-c, --copyright
Display copyright information and then exit. All other options will be ignored.

-h, -?, --help
Display usage message and then exit. All other options will be ignored.

-v, --version
Display version information and then exit. All other options will be ignored.

-db=<value>, --database=<value>
Use name as the database. Without this option, LBT will enter passthru mode (read from stdin and write to stdout).

-t dir, --tmpdir dir
Set the directory LBT uses for its temporary database file to dir. Without this option, UAD will use the default temp directory appropriate to the operating system.

-a, --append
When a pre-existing database is specified, the append option causes new entries to be appended to the end of the database, otherwise, LBT will overwrite the database.

-r=<value>, --restrict=<value>
The restrict option allows LBT to use the choke method (as described in VFind) to limit output as described in the table below.

Num	Chevron	Meaning
1	##==>	Informational Message
2	##==>>	VFind Warning
3	##==>>>	Serious VFind Condition
4	##==>>>>	Possible Virus Detection

THD - Trojan Horse Detector
[179, 180, 181, 182, 183, 184]

THD answers the question, "How does a user find a chameleon Trojan horse attack when there is no contents to scan?" The chameleon Trojan horse attack works because a user is able to redirect a system command to a program of the same name in a different location. The chameleon may or may not have contents. If the chameleon has contents, it can be located using VFind. If it does not have contents, then THD can locate it, because by definition, the file name must have the same name as another program on the system or the attack will not work. A Chameleon Trojan that does not have contents damages the system by disabling important system commands with empty files that do nothing. This will tend to break script programs in addition to delaying efforts by the users and system administrators.

An example of a common chameleon attack is to use the "touch" command to create an empty file with execute permission in the user path prior to the actual commands. Since many users put their home directory first in their path prior system directories these attacks are commonly performed on a per user basis. Commands on Unix/Linux systems which are commonly attacked are: ls, cd, echo, pwd, rm, mv, mail, vi, ed, grep,login, logout, sh, csh, ksh, lp, ps and kill.

Name

thd - Trojan Horse Detector

Description

CyberSoft, Inc.'s Trojan Horse Detector is part of the VFind Security ToolKit. THD is used to detect trojan horse files. It does this by analyzing the basenames (i.e., file names not including any directory part) of full path names that it reads from standard input. It's general enough to be used for many other system analysis/administration purposes as well.

THD reads a list of file names (one per line) from its standard input and generates a report to standard output that lists all files with identical basenames.

If an alarmfile is provided, it should be a text file with file names (one per line) to watch for (e.g., 'ls'). Files with the same basename as one listed in the alarmfile will be listed in the report.

If an ignorefile is provided, it should be a text file with basenames to ignore when checking for duplicate file names (e.g., 'README').

Usage

Input

THD takes input from stdin. CyberSoft suggests that the user uses the UNIX find command to feed a list of files to THD.

Example:

```
$ find / -type f -print | thd --ignore-dotfiles -a myalarms -i myignores
```

Output

THD 's output is very uniform. It will be in this form:

```
(alarm | dup):<set_number>:<instance_number>:<filename>
```

The numbers start at 0. An example output, assuming the use of an alarms file containing 'sendmail', might be as follows:

```
dup:0:0:/bin/ls
dup:0:1:/usr/bin/ls
dup:1:0:/bin/cat
dup:1:1:/usr/bin/cat
alarm:0:0:/home/foo/sendmail
alarm:0:1:/home/bar/sendmail
```

The above example shows that two suspect files were found, as indicated by the 'alarm' lines. Also note that two sets of two files each with the same basename were discovered in two different directories as indicated by the 'dup' lines. The regularity of this output makes it suitable for input to other programs.

THD Command Line Options [180-184]

--license=path
Specify an alternate path to the LICENSE file.

-c, --copyright
Display copyright information and then exit. All other options will be ignored.

-h, -?, --help
Display usage message and then exit. All other options will be ignored.

-v, --version
Display version information and then exit. All other options will be ignored.

-a alarmfile, --alarmfile alarmfile
Tells THD to read filenames from alarmfile. Alarmfile is a text file that contains filenames (one per line) that will be included in the report when found on the system.

-i ignorefile, --ignorefile ignorefile
Tells THD to read filenames from ignorefile. Ignorefile is a text file that contains filenames (one per line) that will be ignored when duplicates are found on the filesystem. These files will not appear in the report.

-id, --ignore-dotfiles
Ignore all files beginning with a '.' and do not report them as duplicates.

-nr, --no-report
Do not generate a report.

THD Example Script

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#!/bin/sh
#
# Name: thd.sh
#
# Copyright: 2003, 2004, 2005, 2006 CyberSoft Operating Corporation, All Rights
Reserved.
#
# Description: This is a general script that will demonstrate how to run the Thd
program.
#

VPATH=/
TOPTS=
pflag=
tflag=
while getopts p:t: name
do
        case $name in
        p)
                pflag=1
                pval=$OPTARG
                ;;
        t)
                tflag=1
                tval=$OPTARG
                ;;
        ?) printf "Usage %s [-p <path>] [-t thd options] \n" $0
                exit 1
                ;;
        esac
done

if [ ! -z "${pflag}" ]; then
        VPATH=${pval}
fi
if [ ! -z "${tflag}" ]; then
        TOPTS=${tval}
fi

PATH=/usr/bin:/usr/sbin:/bin:/sbin:$PATH
export PATH

VSTK_HOME=<vstk_home>
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

BHead - Binary Head
[179, 180, 181, 182, 183, 184]

BHead is a simple tool. It is a program that reads a specified number of bytes from a file or byte stream and writes them to standard output. While this is not a complex task, the problems the tool can solve can be complex. One such problem is that Unix systems running on Intel based PC architecture hardware can be infected with boot sector viruses. Unix systems do not have a convenient way to read just the boot sector for scanning. Using the Unix "dd" command the entire raw disk drive can be read and output as a byte stream. Scanning the entire drive would be a redundant waste of time, however, using BHead the bytestream can be cut to just the portion of the drive that contains the boot sector. BHead can also operate on all forms of removable media.

Name

bhead - Binary Head tool

Description

BHead reads from the standard input and outputs the first numbytes bytes to standard output. This is especially useful when the user only needs to scan a specific number of bytes from a raw device such as a boot sector or tape.

BHead Command Line Options [180-184]

-c, --copyright
Display copyright information and then exit. All other options will be ignored.

-h, -?, --help
Display usage message and then exit. All other options will be ignored.

-v, --version
Display version information and then exit. All other options will be ignored.

-b numbytes
This option tells bhead how many bytes to output to standard output.

Bhead Example Script: Boot.sh

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#!/bin/sh
#
# Name: boot.sh
#
# Copyright 2003, 2004, 2005, 2006 CyberSoft, Inc. All Rights Reserved.
#
# This is an example script on how to use bhead with VFind.
# One possible use for this is to scan a boot sector on a
# PC based Unix system without having to process the entire
# disk drive.  This may not be a problem for floppy diskettes
# but could take a long time on a large hard disk drive unless
# bhead is used.
# boot.sh requires the following command line options:
#  -d <raw device path> which specifies the full path of the
#  raw device of the hard drive containing
#  the boot sector
#  -b <blocking factor> blocking factor of the disk drive
#

PATH=/usr/bin:/usr/sbin:/bin:/sbin:$PATH
export PATH

VSTK_HOME=/opt/vstk
export VSTK_HOME

DEVOPT=
BFOPT=
dflag=
bflag=
while getopts d:b: name
do
        case $name in
        d)
                dflag=1
                devval=$OPTARG
                ;;
        b)
                bflag=1
                bfval=$OPTARG
                ;;
        ?) printf "Usage: %s -d <raw device path> -b <blocking factor>\n" $0
                exit 1
                ;;
        esac
done

if [ -z "${dflag}" -o -z "${bflag}" ]; then
        printf "Usage: %s -d <raw device path> -b <blocking factor>\n" $0
        exit 1
fi

/bin/rm -f sysreport
touch sysreport
echo "VFind OUTPUT REPORT -  BHEAD" >> sysreport
$VSTK_HOME/bin/vfecho_n "System Name: " >> sysreport
hostname >> sysreport
date >> sysreport
df >> sysreport
echo "============================================================" >> sysreport

# The user must specify -d <raw device path> to the correct device
# for their system and must specify -b <blocking_factor> to the
# correct blocking factor for their disk.
# Note that if the input blocking factor and the output blocking factor is
# different, you cannot use the "bs" option.  Refer to the "man" pages for "dd"
# for more information.  These values may be different for hard disks and floppy
# disk drives.

dd if="${devval}" bs="${bfval}" | $VSTK_HOME/bin/bhead -b 1024 | \
      $VSTK_HOME/bin/VFind --stdin |grep "##==>>>"  >> sysreport

echo "============================================================" >> sysreport
date >> sysreport
df >> sysreport
mail root < sysreport
```

```
Example of BHead with CIT to check boot sector for changes
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#!/bin/sh
#
# Name: boot_CIT.sh
#
# Copyright 2011 CyberSoft, Inc. All Rights Reserved.
#
# This is an example script on how to use CIT to detect a change in the
# boot sector of a drive. One possible use for this is to check a boot
# sector on a PC based Unix system without having to process the entire
# disk drive.  This may not be a problem for floppy diskettes
# but could take a long time on a large hard disk drive.
#
# boot_CIT.sh requires the following command line options:
#  -d <raw device path> which specifies the full path of the
#  raw device of the hard drive containing
#  the boot sector
#  -b <blocking factor> blocking factor of the disk drive

PATH=/usr/bin:/usr/sbin:/bin:/sbin:$PATH
export PATH

VSTK_HOME=/opt/vstk
export VSTK_HOME

DEVOPT=
BFOPT=
dflag=
bflag=
while getopts d:b: name
do
        case $name in
        d)
                dflag=1
                devval=$OPTARG
                ;;
        b)
                bflag=1
                bfval=$OPTARG
                ;;
        ?) printf "Usage: %s -d <raw device path> -b <blocking factor>\n" $0
                exit 1
                ;;
        esac
done

if [ -z "${dflag}" -o -z "${bflag}" ]; then
        printf "Usage: %s -d <raw device path> -b <blocking factor>\n" $0
        exit 1
fi

/bin/rm -f sysreport
touch sysreport
echo "-- CIT BOOT OUTPUT REPORT --  " >> sysreport
$VSTK_HOME/bin/vfecho_n "System Name: " >> sysreport
hostname >> sysreport
date >> sysreport
df >> sysreport
echo "============================================================" >> sysreport

# The user must specify -d <raw device path> to the correct device
# for their system and must specify -b <blocking_factor> to the
# correct blocking factor for their disk.
# Note that if the input blocking factor and the output blocking factor is
# different, you cannot use the "bs" option.  Refer to the "man" pages for "dd"
# for more information.  These values may be different for hard disks and floppy
# disk drives.

if [ ! -f "./cit.db" ]; then
        echo "No database, running CIT"
        find . -type f | $VSTK_HOME/bin/cit
        exit 1
fi

dd if="${devval}" of=boot.image  bs="${bfval}" count=2
$VSTK_HOME/bin/cit -f ./boot.image >> sysreport 2>&1
```

```
echo "============================================================" >> sysreport
date >> sysreport
df >> sysreport
mail root < sysreport
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

JDIS - Java Disassembler

JDIS is a tool that performs fast disassembly of Java bytecode. There is a version of JDIS called JADE (Java Advanced Disassembly analysis Engine) built into VFind. It is important to disassemble Java byte code prior to virus scanning because it is the only way to associate constant pool entries with opcodes. This association allows the virus scanner to scan for what is actually going on in the program instead of guessing. VFind is the only anti-virus tool to provide scanning of Java disassembly.

JDIS is also provided as a stand-alone tool so that a security officer can disassemble a Java bytecode program for manual analysis. This can be valuable when confronted with a potential new, unknown Trojan Horse written in Java.

Name

jdis - Java Disassembler tool

Description

JDIS is CyberSoft's Java Disassembler. Java classes (compiled java sources) are examined and disassembled. The output of the disassembly is written to stdout. The output from JDIS is intended for analysis, not recompilation. JDIS Command Line Options

JDIS Command Line Options [180-184]

-c, --copyright
Display copyright information and then exit. All other options will be ignored.

-h, -?, --help
Display usage message and then exit. All other options will be ignored.

-v, --version
Display version information and then exit. All other options will be ignored.

-f filename
jdis will disassemble filename. Filename must be a compiled java class.

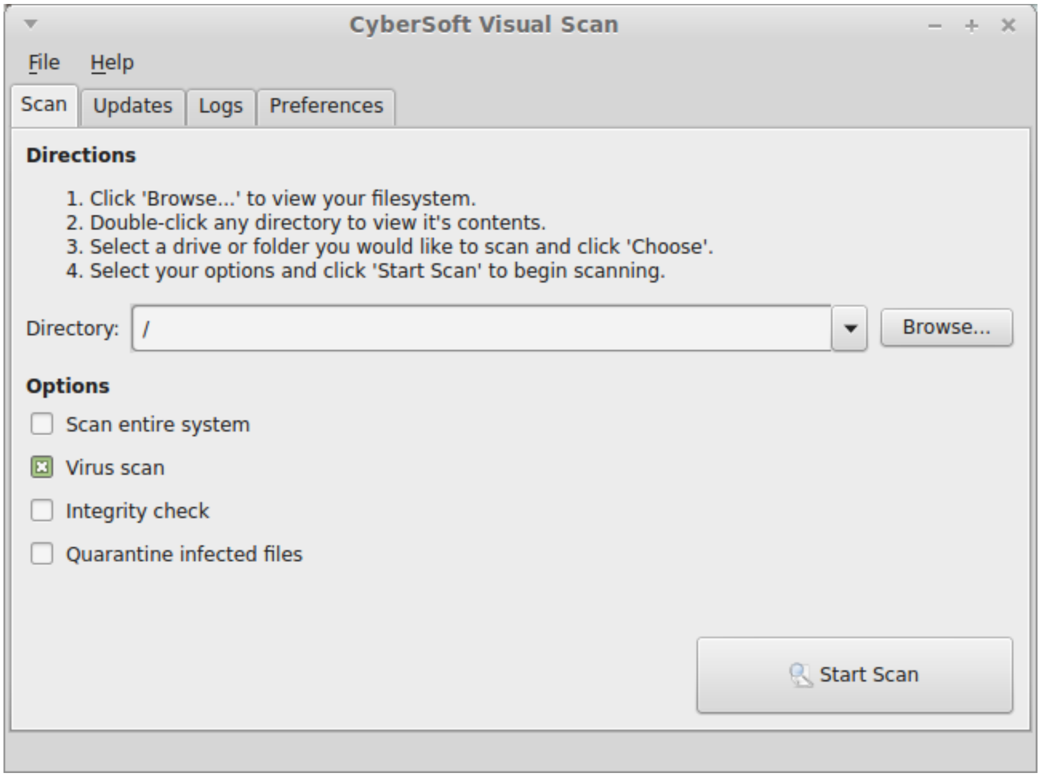
Visual Scan
[180, 181, 182, 183, 184]

CyberSoft has always provided advanced graphical user interfaces for those users who want the ease of a GUI under Unix/Linux and Microsoft Windows. CyberSoft is committed to meeting all technological demands and remaining on the forward edge of technology. This is version 1.4, having advanced from X-windows XView, Motif, TCL/TK, HTML to our latest offering using advanced C++ with Qt.

Visual Scan provides more functionality than ever before while maintaining an easy to understand, intuitive interface with advanced features. Features include scanning, log file management, quarantining, scheduling, automatic updates and system integrity management all from one interface.

Visual Scan is cohesive and well integrated, making it easy to get started and leverage VSTK's core functionality to work for the user.

Visual Scan screen shot - Start Up Page



Directions

1. Click 'Browse...' to view your filesystem.
2. Double-click any directory to view it's contents.
3. Select a drive or folder you would like to scan and click 'Choose'.
4. Select your options and click 'Start Scan' to begin scanning.

Directory: /

Options

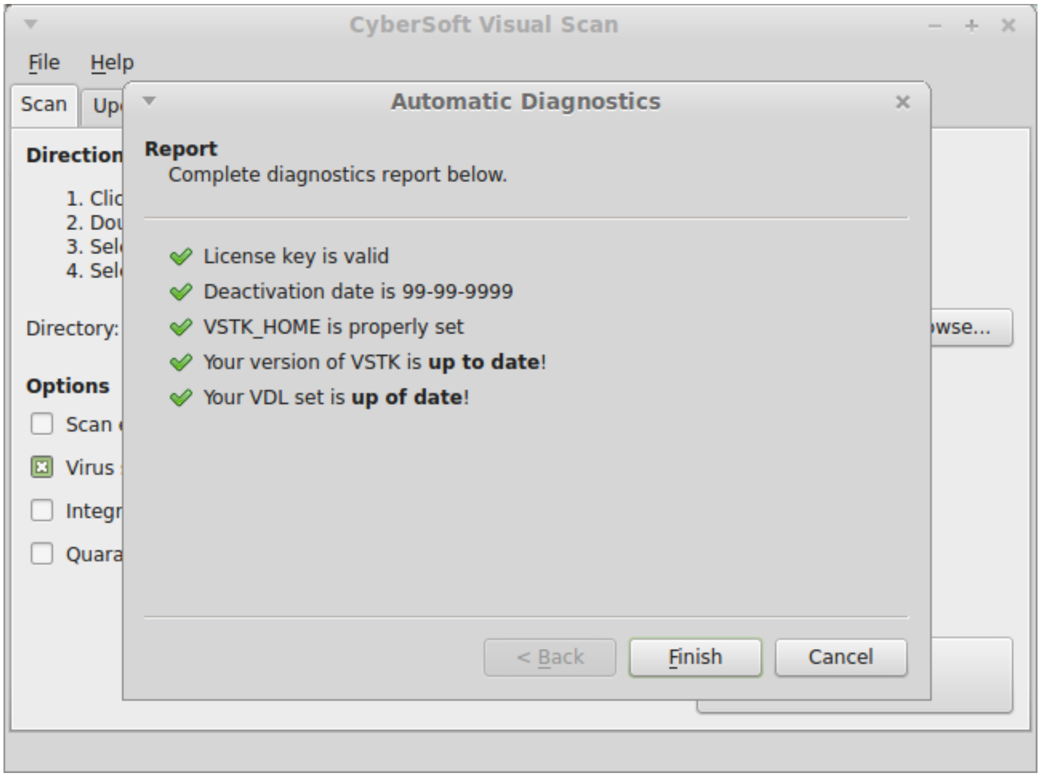
☐ Scan entire system

☒ Virus scan

☐ Integrity check

☐ Quarantine infected files

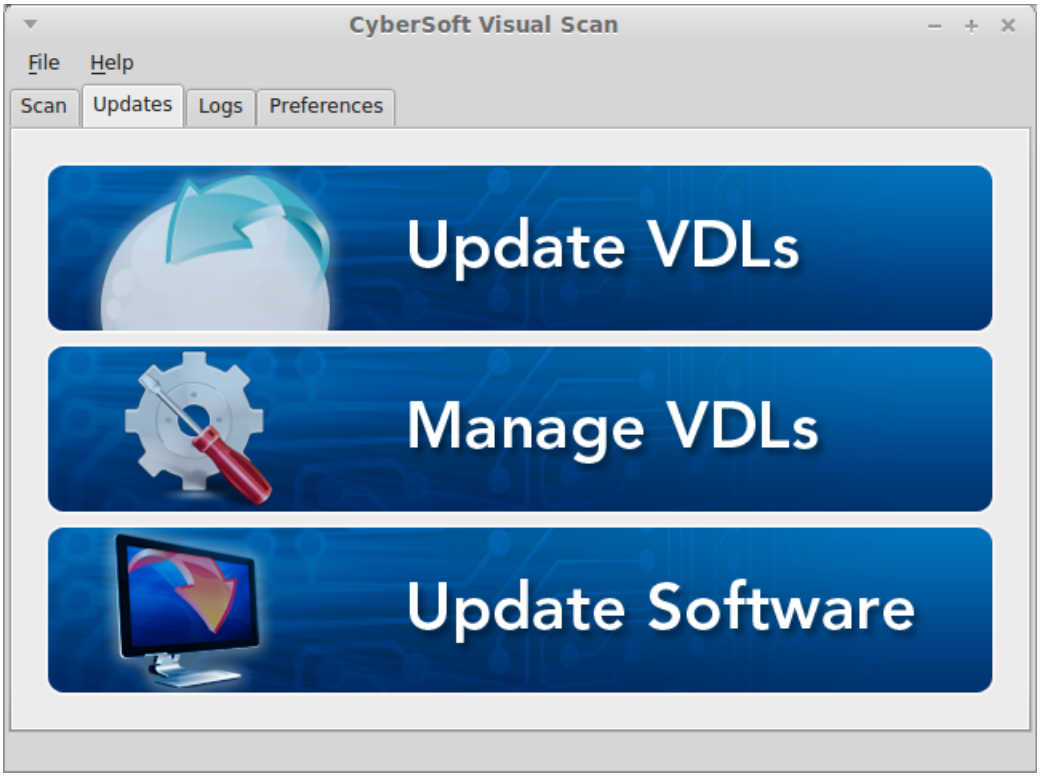
Visual Scan Help screen shot - Automatic Diagnostics Panel



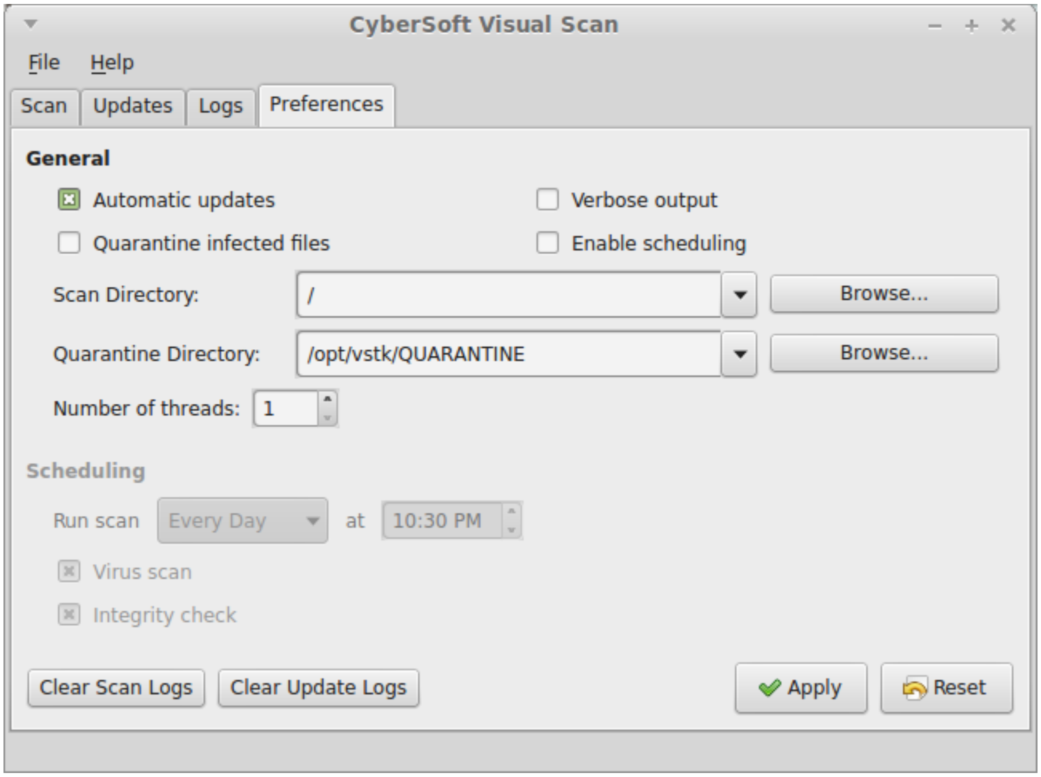
Report
Complete diagnostics report below.

✓ License key is valid
✓ Deactivation date is 99-99-9999
✓ VSTK_HOME is properly set
✓ Your version of VSTK is **up to date**!
✓ Your VDL set is **up of date**!

Visual Scan screen shot - Updates



Visual Scan screen shot - Preferences



CyberSoft Visual Scan

General

☒ Automatic updates

☐ Verbose output

☐ Quarantine infected files

☐ Enable scheduling

Scan Directory: /

Quarantine Directory: /opt/vstk/QUARANTINE

Number of threads: 1

Scheduling

Run scan Every Day at 10:30 PM

☒ Virus scan

☒ Integrity check

Example Script for Visual Scan

```
#!/bin/sh
# Wrapper script for the visual-scan GUI for vfind

PATH=/usr/bin:/usr/sbin:/bin:/sbin
export PATH

VSTK_HOME=/opt/vstk
export VSTK_HOME

exec $VSTK_HOME/programs/visual-scan "$@" >/dev/null 2>&1
```

VFind CVDL Language

Defining and Analyzing Patterns

The Leader in Total Security Solutions For
Linux, UNIX and all *NIX systems

By Richard J. Perry and Peter V. Radatti
Edited by: Justin Honer

19 February 2019

CYBERSOFT OPERATING CORPORATION

1958 BUTLER PIKE, SUITE 100
CONSHOHOCKEN, PA. 19428 U.S.A.
Telephone:+1 610-825-4748?
Fax:+1 610/825-6785
Website: www.cybersoft.com
General Email: sales@cyber.com
Providing Security Tools Since 1988

CVDL Table of Contents

Chapter 7: CVDL and Text

Chapter 8: High-Level CVDL

Chapter 9: CVDL and Regular Expressions

Chapter 10: CVDL Macros

Chapter 11: CVDL Meta Operators

Chapter 12: CVDL Syntax Summary

Chapter 13: Performance and Testing

Chapter 1 Pattern Analysis

1.1 Pattern Analysis for Computer Security[1]

This introduction1 is somewhat philosophical in nature but sometimes that is exactly what is needed to convey meaning. Peter's belief is that after these concepts have been learned, a new concept of pattern analysis can be developed. Pattern analysis is not the "end-all" of computer security but it is a very big hammer. Sometimes (often), when a big hammer is at the user's disposal, other methods just don't matter. What works, works!

Most of computer security is about being able to tell the good from the bad. This is called pattern analysis. To a certain extent everyone does pattern analysis. When a person looks at an orange and recognizes it as such, their brain just did pattern analysis. The brain interpreted what the eyes saw, compared it to known objects, rejected things that were similar but not "it", and arrived at an answer. In fact, the brain may have been wrong. It may not have been an orange but a tangerine. The ability to tell the difference between oranges and tangerines requires additional parameters that might not be available.

If thought about clearly, pattern analysis may be seen as an overwhelming problem that is usually done wrong. Two examples where pattern analysis was done wrong that changed the course of human history were Pearl Harbor in Hawaii and the World Trade Center in New York City.

The raw information for these events was available, but they were not able to be put together to make a picture until it was too late. Hindsight is 20-20. Think of the problem like a one million piece puzzle, that when arranged, could make over a thousand different images, depending on how the pieces were arranged. When looking at a million pieces, the end picture is not previously known or given, and must be interpreted or thought of by the creator, with what they are able to find. As the puzzle is put together, new pieces are found, and perhaps a different picture is formed or found to be more important than the original, whether in reality or in perception. No part of this process is right or wrong, correct or incorrect, because the problem and solution are nebulous, constantly changing.

Only once an end is known, can the pieces be put together in that means. But if the end couldn't have been imagined previous to the end, it will not be found until an external force makes it occur. This is the case with both Pearl Harbor and the World Trade Center. Nobody thought that these events were possible. It was unthinkable, unimaginable. But once an external force came along, and caused that terrible end, the pieces could later be found figure out how the event happened. This would be something that previously would have been considered an unknown unknown, but is now a known unknown.

It is now known that the puzzle builders did not know of these possibilities, but at the time, the puzzle builders did not know that they did not know this was a possibility. In both cases, the information necessary to predict and prevent these events existed prior to the event, but was lost in the noise. That is because in real life, pattern analysis is a very difficult problem. The volume of data itself makes it difficult and frequently what patterns to look for are not known or recognized until it is too late. Fortunately, this work is about digital pattern analysis where the problems of volume and hindsight are still large but within the realm of the possible.

There are two aspects to volume making problems complex. The first is that a whole lot of "stuff" needs to be processed. The second is that given large volumes of data, patterns the user is looking for will naturally occur as a misidentification. This is actually related to the well known postulate that an infinite number of monkeys typing on an infinite number of typewriters will eventually reproduce the entire works of Shakespeare. It is, however, different because when dealing with an infinite number, everything is possible.

In computer security everything is finite even if it seems infinite from a practical point of view. An example of this is that many digital photos are 640 by 480 pixels which yield a total of 307,200 pixels. Using 256 colors, the number 256307200 is derived. This number represents every single photo that has ever been taken or will ever be taken at that size. It contains every picture of every face, building, animal, aliens on distant planets, in fact anything that can be contained in a photo of that size. This number is finite but still so large that it becomes meaningless.

When a pattern exists but is not found, it is called a miss. Pearl Harbor and the World Trade Center were misses. When a pattern is correctly found it is called a hit. An example is when a virus scanner finds a virus. When a pattern is found but it is not correct it is called a false hit. There are several types of false hits, including a special type called a "ghost hit". An example of a false hit is when a virus scanner detects a virus that is not there. An example of a ghost is when a virus scanner detects a virus that was disinfected. The virus is actually there but moot. There are excellent reasons to want to know when ghosts are present but it is most useful to know they are ghosts.

In 1988 Peter wrote a program called "ring toss". Peter named it that based on the backyard game called quates where a person tosses a ring onto a stake in the ground. It is basically the game horseshoes, but with rubber rings. The object of the program is to find all of the letters in a complex sentence in positional order within a single file. Peter had an entire server to run the program against. What Peter found was that many files triggered the ring toss program as a hit. When the files were examined, it was discovered that the program was detecting was data below the noise level. In reality, there was no hit. The pattern was complex and long, and the amount of data to search was finite.

There was no significant data in the detected files but the way the program was looking for the given pattern gave false hits. This brings about the point that the way in which a pattern is looked at matters.

The problem of volume exists in computer security. The volume of data is much smaller than in the real world, but still very large. The issue is that users are looking for patterns that are small. Many computer viruses are only about 60 bytes long! Trojan Horses and Worms can be much longer. Lexical patterns are rarely more than 300 bytes long. The ratio of the pattern size to the volume of data to be searched is critical to the difference between a hit and a false hit. To avoid false hits, the pattern size and complexity needs to increase when the volume of data to be searched increases. When looking for a virus, the user is limited by the maximum size of the virus so additional data such as target file identification and pattern location become meaningful.

Peter calls pattern complexity "data richness". For example, the pattern zero, zero repeated a few hundred times is not data rich. There is no statistical significance to the pattern. In fact, this exact pattern will false hit on almost all executables since many executables set aside variable space using the byte value zero.

What has just been said is that pattern analysis is a game of statistics. The pattern to be searched needs to have a significant length and needs to have significant data richness IN RELATION to the target files that need to be searched. In addition, the pattern must be searched for in a way that is appropriate for the problem at hand. Do any of this wrong, and there will be a false hit.

Another view is that pattern analysis is puzzle solving. There are many different types of parameters that can add data richness beyond the actual pattern. For example, if a user knows that the specific pattern they are interested in can only validly exist in a file of a specific type, the user can increase the data richness of the pattern master by restricting its use to files where the pattern can exist. This also helps to reduce false hits because if a matching pattern does exist in a file of the wrong type it will not be flagged. Positional information can also be of value when enhancing data richness. For example, if a pattern only exists at a specific location in a file, restricting the search for that pattern to where it can exist positionally in the file enhances data richness.

The next part of understanding pattern analysis is that it really exists as two separate problems. The first is to create the master search pattern, "The thing to look for." This is not always obvious or easy. The second is to find the master pattern in target files. This is actually a different set of problems which require different skills. An example is that the analyst who detects a virus for the first time and writes the master pattern can be a different person from the person who wrote the program that executes the master pattern. They are almost always different from the people who run the program in the field.

The good news for the mathematically challenged is that they do not need to know statistics to do this well. In fact, certain people with mental problems do this well. In the movie, "A Beautiful Mind" about Dr. John Forbes Nash, Jr. the viewer sees scenes where Dr. Nash sees patterns that don't exist. The problem as told in the movie was that sometimes with schizophrenia it can be hard to tell the difference between a hit and a false hit. (The movie was not a true representation of the problem Dr. Nash and others suffer but is a good example to make this point.)

Another type of mental condition that seems to make people good at pattern analysis is Attention Deficit Hyperactivity Disorder, ADHD. Some people with ADHD trained in pattern analysis can see the patterns they are looking for. In one case, Peter found an ADHD person with a high IQ who was a natural talent at creating pattern masters.

So far, only simple pattern masters where there is a near one to one correspondence between the pattern master and the search pattern have been discussed. There are additional types of pattern masters which can match valid patterns where the exact pattern is unknown in advance. A simple example of that would be searching for the pattern, "hello" and it is known that some people might spell the word "hello" with the number zero in the letter-O place or the number one in the letter-L place. Of course they may use any or every combination of the above. (This is common for people who learned to type before computers.) A pattern master for a pattern of that type could be described verbally as:

Search for the letter 'h' while ignoring case.
Search for the letter 'e'
Search for the letter 'l' or the number '1'
Search for the letter 'l' or the number '1'
Search for the letter 'o' or the number '0'

This is a very simple little pattern master yet it can detect 16 actual patterns, all of which are valid. Add the ability to ignore white space between letters or search for a prefix and postfix of white space and the data richness is still increased while the number of actual patterns that can be detected increases substantially. Peter defines techniques that deal with this problem as "pattern looseness". Again, while pattern looseness opens the number of patterns that can be found with a single master, data richness dictates that false hits cannot be included!

An area where pattern looseness is extremely useful is when trying to detect a member of a family of data. Most commonly, this is to detect a new member of a family of computer viruses such as the Bagle virus. It is, of course, extremely valuable to be able to have a pattern master that detects all or most of the members of a family. Imagine when a new strain of the Bagle virus is released and even before it becomes known, it is already detected by a family pattern master that included enough looseness to detect it! Pattern looseness also has applications in the biological sciences and in lexical analysis such as preserving secrets.

Another area of study in pattern analysis is data enhancement through transitional formatting. Peter believes that he implemented the first transitional pattern analysis engine in VFind in 1999 when CyberSoft noted that pattern masters of Java byte code were not reliable because of the way the Java system itself worked. The solution was to create a transitional engine that prepared the data for pattern analysis.

In the case of the Java byte code, CyberSoft created a fast, light disassembly system that processed the binary byte code into reliable and easily searched disassembly code. This is backward from a straight pattern search because instead of modifying the pattern master the user modifies the data being searched. Transitional formatting of data to enhance pattern analysis is a very powerful and effective valid tool that can reveal data when no other tool can. Always remember to look at all sides of the problem!

Another aspect of transitional pattern analysis is the meta-data creation that is the transitional phase of the problem. It is necessary to arrive at a transitional function that will process the raw data into meta-data that by its nature lends itself to pattern analysis. This is not as easy as it seems and sometimes the resulting meta-data can not be searched by a simple pattern master but by an enriched pattern master that includes information about the resulting meta-data.

An example of this is when a CPU emulator is invoked to emulate a binary. For the purposes of pattern analysis, this is most often used to detect attack programs such as viruses. The results of emulation may be several useful patterns. The first may be a hexadecimal string of values that have been decrypted or decoded by the emulated code. The second may be a string of events that triggered flags in the emulator.

For example, if a program generates a list of target files and then modifies those target files, the setting of the emulator flags are useful information in themselves. In fact, the user might generate separate pattern masters for a problem of this type. The first is the decoded string of values. That can be a straight-forward pattern master that is used solely against values provided to it by the emulator (Notice that since the pattern master is only applied against results from the emulator, that data richness is enhanced). The second is a pattern master that detects dangerous results of an emulated program. In both cases, the results of the emulation must be interpreted. That interpretation is also "pattern analysis".

Another method of pattern analysis is straight statistical analysis such as the Bayes method. In Bayes analysis, a database is created. The target is statically compared to the database to see how closely it compares to the database. The higher the statistical probability that the target under examination matches the contents of the Bayes database the better the chances are that the target belongs to the class of targets in the database. The drawback to this is that everything depends upon a properly created database.

If the database contains flaws, then the analysis will be flawed. This is why an unsolicited bulk e-mail (UBE) filter will have problems detecting some e-mail as UBE while detecting desired e-mail as such. One of the ways that CyberSoft solved the problem of the comparative database in the Bayes method was to not directly rely upon human population of the database. This solution was implemented in the CyberSoft Safe Internet Email product[2] and found to be very successful.

CyberSoft creates pattern masters with both enough data richness and data looseness to detect a good number of UBE messages. The result is still not good enough to significantly reduce UBE messages from a reader's perspective but it is good enough to populate the Bayes database. The pattern masters and the Bayes method can then be applied against CyberSoft's target file synergistically. Notice that this is another form of transitional data. CyberSoft created a pattern master which detected patterns which were then used as a method of detection of other patterns. The universe of pattern analysis is only limited by imagination.

Just as a matter of clarification, only the CyberSoft implementation of Bayes pattern analysis has been discussed, to determine if something is UBE or not. While this is the most common use of Bayes, it is not limited to that. It can be used to determine if a target conforms to any type of database. This is why VFind allows the user to create multiple Bayes databases and CVDL programs. The user could create a database to determine if a target has a high probability of being a terrorist communication, criminal communication or is in a specific foreign language.

For example, if a user creates a database that contains classical literature written in Latin then they would have a way of determining if a target is a classical literature written in Latin. If the user dumped a large amount of unspecialized communications written in French, then they would have a way of detecting French. Put in a large collection of love letters, and then love letters can be detected. The secret is that a user can statistically determine if something conforms to the Bayes database. What the user puts in the database is unlimited. Peter believes that a user could even put in the hexadecimal strings that compose virus bodies and statistically determine if a target was a virus. There is no practical limit to this technology.

The reader should note that so far that the use of pattern analysis for the purpose of hostile software detection or UBE has been discussed (anti-virus professionals are very fussy about the use of the term "computer virus" but the general public uses the term to mean any hostile software program). Pattern analysis is not limited to these areas. Users can apply pattern analysis to anything. One of the more interesting uses for pattern analysis is searching stock market results real time looking for trends that can result in a gain for the operator.

Pattern analysis can also be used to determine if something is a forgery. In fact, if very large databases of pattern analysis programs were developed, they could be used to analyze and determine even very complex problems such as medical diagnosis or structural analysis. In this view, pattern analysis is a way of capturing expert knowledge in a very specific and highly usable way.

One of the esoteric aspects of pattern analysis is that the tools used define the pattern masters used. Many end users create their own tools to solve single use pattern analysis problems. More sophisticated users may use a language like Perl or Regex but there are significant limitations to these languages, the most important limitation being that there are practical upper limits to what these languages can perform, especially in the number of simultaneous pattern masters that can be solved for. If there is only a small number of pattern masters or a small amount of data to be targeted, then any size hammer will work.

On the other hand, if a pattern master is complex, is dependent upon other pattern masters, requires a large number of simultaneous searches, uses Boolean logic or the user just has a vast amount of data to search, then only a really big hammer will solve that problem.

Peter likes to think that the pattern analysis system that he designed and has been developing since 1990 is the largest general purpose pattern analysis hammer in the world. It started out life as a UNIX virus scanner but very quickly became a pattern analysis system for use with lexical analysis.

It has continued to be enhanced over the years and is now capable of doing pattern analysis in all fields. That tool is called VFind and is part of the VFind Security ToolKit family of products. More information can be found about VFind at cybersoft.com.[3]

There is one more aspect of pattern analysis that is often overlooked by users. That aspect is as important as the creation of the pattern master and often goes hand in hand with it. That aspect is what to do with the knowledge created by the pattern analysis. In the event that the knowledge is that the pattern fits a known virus, delete the virus. If it is UBE, do not forward it.

If the knowledge is more complex, knowing something does or does not match a pattern is of no value without a plan for action. In the example given earlier, what would have happened if the airport screeners had been able to recognize the terrorists who boarded the planes that crashed into the World Trade Center but did not have a plan or authority to deal with them? Peter's guess is that they would have allowed them to board and the event would still have happened. If, however, they had a plan to detain the terrorists, then the event would not have happened. Sometimes even a simple plan is effective and almost always, any plan is more effective than no plan.

1.2 Overview

This introductory chapter concludes with a note on the examples and some exercises. Accompanying material includes examples.zip containing all of the example files, and a CVDL (CyberSoft Virus Description Language) reference card.

Chapter 2 lists the VFind command line options and briefly discusses the CyberSoft VDLs and use of UAD with VFind. Chapter 3 introduces enough basic CVDL to understand Chapter 4 on the VFind scan engines. Chapter 5 is devoted to the CBayes scan engine, and Chapters 6 to 12 form the major portion of this reference on CVDL. Chapter 13 concludes with discussions and examples of performance testing.

1.3 A Note on the Examples

Interactive examples were performed on a Sun/Solaris system using csh with prompt '%'. In csh, '|&' can be used to pipe both standard output and standard error into another program, so many of the examples run VFind like this to limit visible output to just detections:

```
% VFind ...| & grep VIRUS
^^^
promptpipe stdout and stderr
```

The equivalent command using a Bourne shell is:

```
$ VFind ...2>&1 | grep VIRUS
^^^^^
promptmerge stderr with stdout
```

1.4 Exercises

1. Check the CyberSoft web site for updates on the VSTK ToolKit and virus definitions.

2. Check any website for text or binary information and create a VDL to match it. This is more difficult than it may appear at first. (Think HTML)

End Notes

1. P. Radatti, "Pattern analysis for computer security" http://cybersoft.com, whitepapers
2. SafeInternetEmail. http://safeinternetemail.com/
3. CyberSoft Operating Corporation. http://www.cybersoft.com/

Chapter 2 Using VFind

2.1 CyberSoft VDLs

The VFind Security ToolKit, including programs and data files, is installed in a $VSTK HOME directory, usually /opt/vstk/. In VSTK HOME, the sub-directory data/vfind/ contains the master VDL list file vdl.list, and VDL files as described in Chapter 4. The CyberSoft VDL files are written in plain text using the CVDL language, and may be examined as an aid in learning to write custom patterns.

For most of the examples presented here, the VFind -liboff='*' option is used, which skips the CyberSoft VDLs in order to focus on user-written VDLs.

2.2 Using UAD with VFind

In practice, for scanning arbitrary files, CyberSoft's UAD tool is used in a pipeline or as a forked process from the VFind daemon (if turned on via SVSP) to unpack, decode and/or uncompress archives and other composite files, such as .zip or .tar.gz files or e-mail with encoded attachments. UAD transfers data, file names, and file types to VFind using CyberSoft's SmartScan protocol[1] specified by the uad -ssw (SmartScan write) and VFind -ssr (SmartScan read) command-line options, for example:

```
% uad -ssw somefiles...| vfind -ssr other_vfind_options...
```

Note, by default as of release 182 UAD is sub-processed by VFind by default.

For simplicity, the examples here use standalone VFind, without UAD, but for most kinds of production use of the VSTK ToolKit it is recommended to use both UAD and VFind with SmartScan.

2.3 Exercises

1. Install VFind on a system, or use an existing installation. Use VFind to scan all files on the system, or at least all of those files that are readable for to the user. Record the number of VDLs reported, number of files scanned, real and CPU run-time used, and peak RAM usage.

2. Examine the VFind CyberSoft VDLs, and create an example file which will match one of the VDLs.

End Notes

1. CyberSoft, "Smartscan." http://cybersoft.com/, whitepapers, programming examples.

Chapter 3 Basic CVDL

CVDL is the CyberSoft Virus Description Language. It was first defined in January 1992 to provide flexibility in scanning for viruses and general purpose scanning. Since that time, CVDL has undergone many enhancements and continues to actively evolve to meet new scanning needs.

CVDL is used to define patterns for scanning for most of the engines within VFind, i.e. the vdl, vdlc, vdle, vdlE, vdlm, and jadevdl engines. This chapter introduces just enough basic CVDL so the reader will be able to understand the examples in the next chapter on VFind Scan Engines. The rest of CVDL is described in Chapters 6 to 12.

3.1 Pattern Size Limit - No Limit on Input Data, 65537 bytes pattern limit

Before using CVDL to create patterns, it is important to understand how data is scanned by VFind. First of all, note that there is no limit on the size of the input data being scanned. To accommodate limitless data, VFind uses an input data sliding scan window of size 1 MB (1024*1024 = 1048576 bytes), with an overlap of 64 KB (64*1024 = 65536 bytes) between windows. So after scanning the first MB of input data, the next window of data scanned consists of the last 64 KB from the first window followed by an additional 983040 bytes (1048576 - 65536) of new data.

Patterns matching 65537 bytes or less of data will always work, but patterns matching 65538 or more bytes of data may not work if the matching data overlaps a scan window edge.

For example, if 65538 bytes of data are to be matched and the first 65537 of those bytes occur at the end of one scan window, it will not match in that window since it is missing the last byte of the data; the next window will start with the last 65537 bytes of the data and will also not match since it is missing the first byte of the data.

3.2 VDL Format

A VDL is a named set of CVDL patterns. VDLs start with a colon character, followed by the VDL name, comma, definition, and terminating # character:

```
: name , definition # ; comment
; comment to end of line after semi-colon
```

The VDL name can consist of any characters, including spaces, but cannot contain a comma, since the comma is used to separate the name from the definition. The VDL can extend over multiple lines, and white space is ignored except in the name and literal strings.

The VDL definition consists of a sequence of patterns and operators which can span multiple lines. The definition is terminated by the # character. The semicolon (;) character is used to specify a comment which extends to the end of the line.

The simplest type of patterns are literal strings, e.g. "abc", and the simplest type of operators are concatenation and offsets. These are described in the following sections.

3.3 Strings

A CVDL string pattern is a sequence of characters and escape sequences enclosed in "double quotes".

An escape sequence is \c or a hex escape sequence. If character c is n, r, or t this represents new line, carriage-return, or tab, respectively; otherwise, it represents character c itself. A hex escape sequence is \x or \X followed by two hex digits.

Prefixing a string with a tilde (~) specifies case-insensitive matching, e.g. ~"abc" will match "abc", "ABC", "AbC", etc. Inside of a string, lines ending with a backslash ("\") specify continuation on the next line.

VDL string examples:

```
:v1, "abc\"efg" # ; matches: abc"efg
:v2, "abc\x22efg" # ; same as v1
:v3, "abc\x22\
efg" # ; same as v1
```

3.4 Concatenation and Offsets

VDL pattern elements are concatenated into larger patterns by using a comma, for example: "abc", "def"

is the same as: "abcdef"

In a VDL definition the comma operator means followed by. An offset range for concatenation can be specified using the @ operator, for example:

"abc", @0-10, "def"

will match "abc" followed by "def" at an offset of anywhere from 0 to 10 bytes from the end of "abc". So it will match "abcdef", "abcXdef", ..., "abcXXXXXXXXXXdef", where X represents any byte.

In the offset range operator start-end, where start or end may be omitted to use the defaults of 0 and 32767. So @-10 is the same as @0-10, and @20000- is the same as @20000-32767. If only one value is specified with no hyphen, i.e. @val, that is the same as @val-@val, which is a fixed offset instead of a range.

Note that an offset range only makes sense if preceded by a pattern to offset from. It does not make sense to specify an offset range at the beginning of a VDL segment because there is nothing to offset from. The beginning of a VDL is automatically checked at all offsets from the beginning of the scanned data. The VFind -d, -dup-check option (see Section 2.1), which checks for duplicate VDL names and definitions, also checks for offsets at the beginning of a VDL segment, and reports those as parser errors.

Offset ranges are useful for matching data which contains fixed patterns separated by variable offsets. For example, here are "unsubscribe" sentences from four different spam e-mail messages:

```
% cd examples/vdl/spam
% foreach i (1 2 3 4)
? echo s$i
? cat s$i
? end
s1
If you would like to unsubscribe from receiving further s2
You can unsubscribe anytime if you want.
s3
Or, you may unsubscribe via postal mail by printing s4
If you wish to unsubscribe from future mailings
%
```

If it is decided that detecting the word "you", case-insensitive, followed closely by "unsubscribe" may be a good way to match such sentences, it could be using the following VDL:

```
% cat s.vdl
:SPAM unsub, ~"you", @-20, "unsubscribe" #
% VFind --liboff='*' --vdl=s.vdl s{1,2,3,4} |& grep VIRUS
##==>>>> VIRUS POSSIBLE IN FILE: "s1"
##==>>>> VIRUS ID: CVDL SPAM unsub
##==>>>> VIRUS END OFFSET: 32, matched: to unsubscribe
##==>>>> VIRUS POSSIBLE IN FILE: "s2"
##==>>>> VIRUS ID: CVDL SPAM unsub
##==>>>> VIRUS END OFFSET: 19, matched: can unsubscribe
##==>>>> VIRUS POSSIBLE IN FILE: "s3"
##==>>>> VIRUS ID: CVDL SPAM unsub
##==>>>> VIRUS END OFFSET: 23, matched: may unsubscribe
##==>>>> VIRUS POSSIBLE IN FILE: "s4"
##==>>>> VIRUS ID: CVDL SPAM unsub
##==>>>> VIRUS END OFFSET: 26, matched: to unsubscribe
%
```

Offset ranges can be used to match executable code fragments irrespective of variable data values. For example, the disassembly shown in Section 4.3.2 includes the following code:

```
emu: 00004f23 bf474f mov DI 0x4f47 ; DI=0x0 ; \xbfGO
emu: 00004f26 b86b1b mov AX 0x1b6b ; AX=0x0 ; \xb8k\x1b emu: 00004f29 90 nop ; ; \x90
emu: 00004f2a fc cld ; ; \xfc
emu: 00004f2b b9b504 mov CX 0x4b5 ; CX=0x0 ; \xb9\xb5\x04 emu: 00004f2e 90 nop ; ; \x90
emu: 00004f2f 2bda sub BX DX ; BX=0xffff DX=0x0 ; +\xda
emu: 00004f31 2bd8 sub BX AX ; BX=0xffff AX=0x1b6b ; +\xd8
emu: 00004f33 33d9 xor BX CX ; BX=0xe494 CX=0x4b5 ; 3\xd9
emu: 00004f35 310d xor DS:[DI] CX ; DS=0x0 DI=0x4f47 CX=0x4b5 ; 1\x0d
emu: 00004f37 3105 xor DS:[DI] AX ; DS=0x0 DI=0x4f47 AX=0x1b6b ; 1\x05
emu: 00004f39 f8 clc ; ; \xf8
emu: 00004f3a 43 inc BX ; BX=0xe021 ; C
```

This is part of a polymorphic virus decryption engine, and each time the virus replicates, this engine code is copied with different initial values for the DI and AX registers. If the byte values from column three of the disassembly is taken:

```
bf474fb86b1b90fcb9b504902bda2bd833d9310d3105f843
```

and create a VDL:

```
:vpoly,"\xbf\x47\x4f\xb8\x6b\x1b\x90\xfc\xb9\xb5\x04\x90\x2b\
\xda\x2b\xd8\x33\xd9\x31\x0d\x31\x05\xf8\x43"#
```

That will match this particular sample, but will not match other copies of the virus which use different initial DI and AX values. But using offsets, the VDL can be modified as follows so that it will match irrespective of the initial DI and AX values:

```
:vpoly,"\xbf",@2,\xb8",@2,\x90\xfc\xb9\xb5\x04\x90\x2b\
\xda\x2b\xd8\x33\xd9\x31\x0d\x31\x05\xf8\x43"#
```

3.5 Entropy and Serial Correlation

To avoid false hits on clean data, VDL patterns should be taken from "data rich" sections of the samples to be matched. "Data rich" can be defined as patterns which are unlikely to appear randomly even in compressed data. These are generally patterns which have high entropy and low serial correlation.

Entropy is a measure of information content, with units of bits-per-byte as used here, so it ranges from a minimum of 0 to a maximum of 8. Entropy is computed from the probabilities of occurrence of each possible byte value 0...255:

$$e = \sum_{i=0}^{255} pi \log_2 (1/pi)$$

Figure 3.1: Entropy computation

Where the probabilities pi are estimated using frequency of occurrence of each byte.

It is possible for a highly correlated data sequence to have high entropy, even though the data is not very randomly distributed. Thus, in conjunction with the entropy calculation, the serial correlation coefficient should also be checked.

The serial correlation coefficient is a measure of the amount that sequential data is related. A correlation coefficient always lies between -1 and +1. When it is zero or very small, it indicates that the data values are (relatively speaking) independent of each other. But when the correlation coefficient is close to ±1 it indicates linear dependence between data values (Knuth[1] page 70).

For n data samples U0, ..., Un-1the serial correlation coefficient is defined as:

$$s = \frac{n\,(U_0U_1 + U_1U_2 + \ldots + U_{n-2}U_{n-1} + U_{n-1}U_0) - S_1^2}{nS_2 - S_1^2}$$

Figure 3.2: Serial Correlation Coefficient Definition

where S1 and S2 are defined as:

$$S_1 = U_0 + U_1 + \cdots + U_{\mathrm{n}-1}$$

$$S_2 + U_0^2 + U_1^2 + \cdots + U_{n-1}^2$$

Figure 3.3: S1 and S2 Definitions

The sample entropy and serial correlation coefficient should be computed over a sliding window through the sample data, and the window with maximum (e - 8|s|) selected to produce a VDL string pattern. The factor of 8 simply scales the absolute serial correlation coefficient value to be in the same range as entropy to balance the maximization.

Three simple example files were created for testing entropy and serial correlation calculations:

```
% cd examples/vdl
% cat mkdata.sh
#! /bin/sh
dd ibs=1 count=100 < /dev/zero > zeros.dat
dd ibs=1 count=100 < /dev/random > random.dat
gzip < /var/log/syslog | dd ibs=1 skip=100 count=100 > gzip.dat
% perl es.pl < zeros.dat e = 0, |s| = 1, e-8|s| = -8
% perl es.pl < random.dat
e = 6.4, |s| = 0.0478, e-8|s| = 6.02
% perl es.pl < gzip.dat
e = 6.22, |s| = 0.176, e-8|s| = 4.81
%
```

The example file zeros.dat contains all 0 bytes, so it has very low entropy and high correlation (the es.pl Perl script used to compute e, |s|, and (e - 8|s|) is shown below). In contrast, the random data has very high entropy and low correlation, and the gzip/compressed data has high entropy but more correlation than the random data.

Now suppose a VDL pattern to match the file sleep.exe from the examples/vdlE/ directory needs to be created. Evaluating the entropy and correlation over a window of 100 bytes at a few different file offsets yields:

```
% sh
$ for i in 0 1000 2000 10000
> do
> x=‘dd if=../vdlE/sleep.exe ibs=1 skip=$i count=100 2>/dev/null | perl es.pl‘
> echo "offset=$i $x"
> done
offset=0 e = 4.96, |s| = 0.064, e-8|s| = 4.45 offset=1000 e = 5.16, |s| = 0.233, e-8|s|
= 3.3 offset=2000 e = 4.43, |s| = 0.139, e-8|s| = 3.32 offset=10000 e = 5.09, |s| =
0.172, e-8|s| = 3.72
$
```

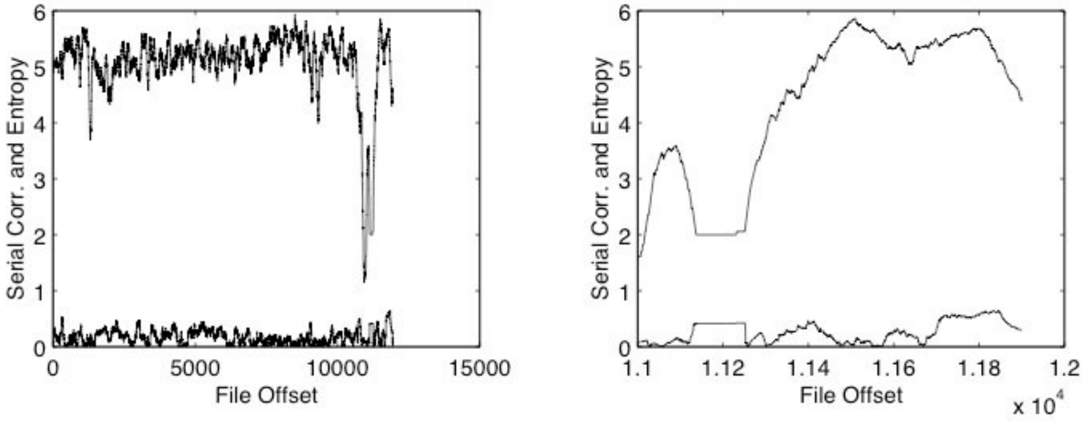


Figure 3.4: Serial Correlation and Entropy vs. File Offset - 100 byte sliding window

So it may now be decided to use the offset=0 data, since that had the largest (e - 8|s|) value; however, a more detailed analysis, illustrated in Figure 3.3, would consider all possible file offsets. The left side of the figure plots the serial correlation and entropy using a window of 100 bytes sliding over the whole file. The right side is zoomed in starting at file offset 11000. Further analysis reveals that the maximum value of (e - 8|s|) occurs at file offset 11506, where the values are:

```
% dd if=../vdlE/sleep.exe ibs=1 skip=11506 count=100 | perl es.pl e = 5.86, |s| =
0.0166, e-8|s| = 5.73
```

That file data can be extracted in hex:

```
% dd if=../vdlE/sleep.exe ibs=1 skip=11506 count=100 | od -tx1
0000000 c0 bf ff ff 2d 00 10 73 0a f7 d8 b1 04 d3 e0 2b
0000020 f8 33 c0 8e c0 fd 8a 5c fd 8a 3c 8b 6c fe 8a d7
0000040 8a c3 8b cd 83 ee 03 8b de f6 c2 01 75 03 2b d9
0000060 43 4b 8b 6f fe 8a 67 fd 8a 3f 86 dc f6 c2 01 74
0000100 07 4e e3 06 f3 aa eb 02 f3 a4 84 d2 78 38 b1 04
0000120 8b c7 f7 d0 d3 e8 81 cf f0 ff 8c c2 2b d0 73 08
0000140 f7 da d3 e2
```

and create the corresponding VDL pattern:

```
:sleep.exe, "\xc0\xbf\xff\xff\x2d\x00\x10\x73\x0a\xf7\xd8\xb1\x04\xd3\xe0\x2b\xf8\x33\
\xc0\x8e\xc0\xfd\x8a\x5c\xfd\x8a\x3c\x8b\x6c\xfe\x8a\xd7\x8a\xc3\x8b\xcd\
\x83\xee\x03\x8b\xde\xf6\xc2\x01\x75\x03\x2b\xd9\x43\x4b\x8b\x6f\xfe\x8a\
\x67\xfd\x8a\x3f\x86\xdc\xf6\xc2\x01\x74\x07\x4e\xe3\x06\xf3\xaa\xeb\x02\
\xf3\xa4\x84\xd2\x78\x38\xb1\x04\x8b\xc7\xf7\xd0\xd3\xe8\x81\xcf\xf0\xff\
\x8c\xc2\x2b\xd0\x73\x08\xf7\xda\xd3\xe2"#
```

Following is the perl script used to compute entropy and serial correlation:

```
% cat es.pl
undef $/; @b = split("",<STDIN>); $len = 1+$#b;
# compute entropy $e
for( $i = 0; $i < 256; ++$i) { $x[$i] = 0; }
for( $i = 0; $i < $len; ++$i) { $c = ord($b[$i]); ++$x[$c]; }
$e = 0; $l2 = log(2);
for( $i = 0; $i < 256; ++$i) { if( $x[$i] > 0) {
$p = $x[$i]/$len; $e += $p * log(1/$p)/$l2; } }

# compute serial correlation coefficient $s
$d = ord($b[0]);
$s1 = ord($b[$len-1]); $s2 = $s1*$s1; $s12 = $s1*$d;
for( $i = 1; $i < $len; ++$i) { $c = $d; $d = ord($b[$i]);
$s1 += $c; $s2 += $c*$c; $s12 += $c*$d; }
$s1 *= $s1; $d = $len * $s2 - $s1;
if( $d == 0.0) { $s = 1.0; } else { $s = ($len * $s12 - $s1)/$d; }
printf "e = %.3g, |s| = %.3g, e-8|s| = %.3g\n", $e, abs($s), $e-8*abs($s);
```

3.6 Exercises

1. Write a program to compute e, |s|, and (e-8|s|) over an entire file using a sliding window of adjustable size, find the file offset to maximize (e-8|s|), and output the corresponding VDL.

2. Create a VDL to match some executable file on the system, then check all executable files for false hits.

End Notes

1. D.E. Knuth, The Art of Computer Programming, Second Edition, Volume 2, Seminumerical Algorithms. Addison–Wesley, 1981.

Chapter 4 VFind Scan Engines

VFind scans data using several different engines which implement pattern matching techniques appropriate for different kinds of data. This chapter describes each engine in detail with examples, except for the cbayes engine which is described separately in Chapter 5. It is important to understand the VFind engines because VDLs and other patterns that may want to be matched must be designed for use with a specific engine. Pattern analysis engine design is a constantly evolving and ongoing process that needs to react not only to projected needs, but also to real world events. For this reason, this section of the manual is subject to change without notice.

The engines are:

vdl - The most general and basic scan engine. This engine can be used to scan any kind of data.

vdlc - A case-insensitive scan engine which should be used only for scanning text.

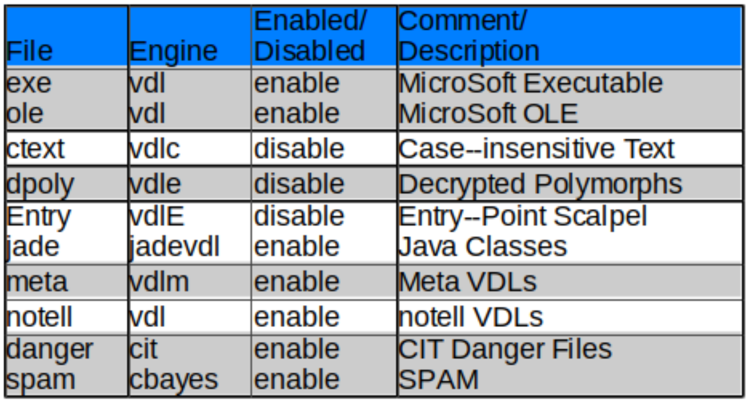
vdlm - A meta engine which detects patterns in the results of the other engines rather than in the data.

jadevdl - A Java disassembler engine which scans the results of Java class file disassembles rather than the original data.

md5/cit - An MD5-hash-based engine for efficient whole-file matches using hash databases.

cbayes - A scan engine based on Bayesian probability classification.

The engines are configured with pattern files specified in VSTK HOME/data/VFind/vdl.list which contains entries specifying pattern file name, engine, enable/disable, and description. Except for CIT engine pattern files, which reside in the VSTK HOME/data/cit/ directory, all of the pattern files reside in the VSTK HOME/data/VFind/ directory.
Table: Example vdl.list file:



File	Engine	Enabled/ Disabled	Comment/ Description
exe	vdl	enable	MicroSoft Executable
ole	vdl	enable	MicroSoft OLE
ctext	vdlc	disable	Case--insensitive Text
dpoly	vdle	disable	Decrypted Polymorphs
Entry	vdlE	disable	Entry--Point Scalpel
jade	jadevdl	enable	Java Classes
meta	vdlm	enable	Meta VDLs
notell	vdl	enable	notell VDLs
danger	cit	enable	CIT Danger Files
spam	cbayes	enable	SPAM

Each engine also has a command-line option to specify additional user-defined pattern files.

-vdl=file Read additional virus descriptions from the specified file.

-vdlc=file Read additional case insensitive virus descriptions from the specified file.

-vdlm=file Read additional meta virus descriptions from the specified file.

-jadevdl=file Load additional virus signatures from file. File contains VDL models for hostile java applets and applications.

-md5=file Read additional MD5 signatures from the specified file.

-cbayes=file Read additional cbayes data from the specified file.

4.1 VDL Engine

For general purpose scanning for binary patterns in arbitrary file types, the vdl engine should be used. Some attention should be given to constructing patterns in parts or using hex notation to avoid false hits on the pattern file itself.

For example, if the pattern to be detected is "abcdefg", VDL ag1 could be created in file ag.vdl: :ag1, "abcdefg" # however, VFind would report a match when scanning the ag.vdl file itself. To avoid such matches, the pattern could be either split into two parts as in VDL ag2, or write part of it in hex or decimal as in ag3, and ag4 :

```
% cd examples/vdl
% cat x.txt abc
abcd
----

ZZabcdefghij
% cat ag.vdl
:ag2, "abc", "defg" #
:ag3, "abc\x64efg" #
:ag4, "abc", 100, "efg" #
% VFind --liboff='*' --vdl=ag.vdl x.txt ag.vdl |& egrep 'Checking file|VIRUS'
##==> Checking file: "x.txt"
##==>>>> VIRUS POSSIBLE IN FILE: "x.txt"
##==>>>> VIRUS ID: CVDL ag4
##==>>>> VIRUS END OFFSET: 23, matched: \x0a----\x0aZZabcdefg
##==>>>> VIRUS POSSIBLE IN FILE: "x.txt"
##==>>>> VIRUS ID: CVDL ag3
##==>>>> VIRUS END OFFSET: 23, matched: \x0a----\x0aZZabcdefg
##==>>>> VIRUS POSSIBLE IN FILE: "x.txt"
##==>>>> VIRUS ID: CVDL ag2
##==>>>> VIRUS END OFFSET: 23, matched: \x0a----\x0aZZabcdefg
##==> Checking file: "ag.vdl"
```

Another example is the pattern provided by EICAR[1] for use in testing anti-virus scanners:

```
X5O!P%@AP[4\PZX54(P^)7CC)7}
$EICAR-STANDARD-anti-virus-TEST-FILE!$H+H*
```

The pattern should be specified all on one line with no blanks or new line characters, but it is split over two lines above to avoid detection if this report is scanned. The EICAR printable string is actually a legitimate MS/DOS.COM program, and when run it prints the message:

EICAR STANDARD-anti-virus-TEST-FILE!

Detection of the EICAR test pattern includes additional constraints which will be considered later in Section 6.7. Here is a shell script to create a VDL which will match the pattern anywhere in a file:

```
% cat eicar.sh
#! /bin/sh
# create eicar hex vdl from eicar.com
echo ':EICAR,"\c'
perl -e 'undef $/; foreach $b (split("",<STDIN>)){ printf("\\x%02x",ord($b));}'
echo '"#'
% ./eicar.sh < eicar.com > eicar.vdl
% fold eicar.vdl
:EICAR,"\x58\x35\x4f\x21\x50\x25\x40\x41\x50\x5b\x34\x5c\x50\x5a\x58\x35\x34\x28
\x50\x5e\x29\x37\x43\x43\x29\x37\x7d\x24\x45\x49\x43\x41\x52\x2d\x53\x54\x41\x4e
\x44\x41\x52\x44\x2d\x41\x4e\x54\x49\x56\x49\x52\x55\x53\x2d\x54\x45\x53\x54\x2d
\x46\x49\x4c\x45\x21\x24\x48\x2b\x48\x2a"#

% VFind --liboff='*' --vdl=eicar.vdl eicar.com |& grep VIRUS
##==>>>> VIRUS POSSIBLE IN FILE: "eicar.com"
##==>>>> VIRUS ID: CVDL EICAR
##==>>>> VIRUS END OFFSET: 68, matched: TEST-FILE!$H+H*
%
```

For a pure binary example, suppose the user has used an analysis of entropy and/or serial correlation (see Section 3.5) or other means to identify that offset lines 9 and 10 of an octal dump of the file binary.dat would produce a good VDL. A VDL can be created semi-automatically using a small shell script:

```
% cat binary.sh
#! /bin/sh
# create vdl from octal dump input echo ':vbin,"\c'
cut -c8- | sed 's/ /\\x/g' | tr -d '\n'
echo '"#'
% od -tx1 binary.dat | head -10 | tail -2
0000200 3c 7e 81 8d c2 8a bb 3e de ae 97 fd 6f a5 3c 20
0000220 c0 58 c7 35 22 c5 19 f4 6c 60 91 b2 77 44 6c a2

% od -tx1 binary.dat | head -10 | tail -2 | ./binary.sh > binary.vdl

% fold binary.vdl

:vbin,"\x3c\x7e\x81\x8d\xc2\x8a\xbb\x3e\xde\xae\x97\xfd\x6f\xa5\x3c\x20\xc0\x58\
xc7\x35\x22\xc5\x19\xf4\x6c\x60\x91\xb2\x77\x44\x6c\xa2"#


% VFind --liboff='*' --vdl=binary.vdl binary.dat |& grep VIRUS
##==>>>> VIRUS POSSIBLE IN FILE: "binary.dat"
##==>>>> VIRUS ID: CVDL vbin
##==>>>> VIRUS END OFFSET: 160, matched: X\xc75"\xc5\x19\xf4l'\x91\xb2wDl\xa2
%
```

4.2 vdlc Engine

As will be discussed in Section 13.2, VFind uses a parallel search engine design, so scanning runtime is mostly independent of the number of VDL rules. However, the general vdl parallel search engine does not handle case-insensitive (Section 3.3) or wildcard white space (Section 7.5) patterns, so VDLs containing such patterns generally run slow. The vdlc engine was created specifically to provide fast/parallel run-time for these patterns.

Note that use of the vdlc engine only affects the scanning speed and does not affect the accuracy of pattern detection. VDL patterns are always checked completely and serially if triggered by a parallel search engine, to avoid false detections. This means, for example, the following VDL:

```
:mixy, "Hello" and ~"goodbye" #
```

will only match text which contains "Hello" (case-sensitive), and "goodbye" (case-insensitive, so "GoodBye" and "GOODbYe" also match) regardless of whether the vdl or vdlc engine is used, as can be seen in the sample run below: file t2.txt is matched but t1.txt is not; however, scanning will be faster for this VDL if used with the vdlc engine.

```
% cd examples/vdlc
% cat mixy.vdl
:mixy, "Hello" and ~"goodbye" #
% cat t1.txt
abc heLLO GoodBYE
% cat t2.txt
abc Hello GoodBYE
% VFind --liboff='*' --vdl=mixy.vdl t1.txt t2.txt |& grep VIRUS
##==>>>> VIRUS POSSIBLE IN FILE: "t2.txt"
##==>>>> VIRUS ID: CVDL mixy
##==>>>> VIRUS END OFFSET: 17, matched: c Hello GoodBYE
% VFind --liboff='*' --vdlc=mixy.vdl t1.txt t2.txt |& grep VIRUS
##==>>>> VIRUS POSSIBLE IN FILE: "t2.txt"
##==>>>> VIRUS ID: CVDL mixy
##==>>>> VIRUS END OFFSET: 17, matched: c Hello GoodBYE
%
```

4.3 vdlm Engine

The vdlm (m for meta) engine detects patterns in the results of the other engines rather than in the data. For each input file scanned, all "VIRUS ID" results are saved in a temporary buffer. When the end of the input file is reached, and all other engines are finished scanning the file, then the vdlm engine scans the buffer of results.

Take the first example from Section 4.1; based on the hits reported, the temporary buffer scanned by the vdlm engine would contain the following three lines:

```
CVDL ag4
CVDL ag3
CVDL ag2
```

Using the high-level CVDL "AND" operator from Chapter 8, a meta vdl could be written to match if both "ag2" and "ag3" are found:

```
% cd examples/vdlm
:ag2+ag3, "CVDL ag2" AND "CVDL ag3" #
% cat ag23.vdl
:ag2+ag3, "CVDL ag2" AND "CVDL ag3" #
% VFind --liboff='*' --vdl=../vdl/ag.vdl --vdlm=ag23.vdl ../vdl/x.txt |& grep VIRUS
...
##==>>>> VIRUS ID: CVDL ag2+ag3
##==>>>> VIRUS END OFFSET: 17, matched: DL ag4\x0aCVDL ag3
%
```

In conjunction with notell VDLs described in Section 11.4, meta VDLs can be useful for detection with UAD/SmartScan input that decomposes archives into separate components. Since each component is scanned separately, it is not possible to create a regular VDL which would match across components. However, notell VDLs can be created to match individual components, and then a meta VDL can be written to match on the notell hits. For more information and examples see Chapter 11 and Section 11.5 in particular.

4.4 jadevdl Engine

The jadevdl engine is a Java disassembler which scans the results of Java class file disassemblies rather than the original data. The engine is based on the VSTK JDIS tool, which can be used for analysis of class files to design VDLs.

Note that Java disassembly does not produce Java source code, it produces the instructions that comprise the Java byte codes, for each of the methods in a class. These are documented in the Java Virtual Machine Specification[2]. The following example shows a Java applet which may be hostile since it attempts to read a local file (SYSTEM.DAT):

```
% cd examples/jadevdl
% cat applet.java import java.io.*; import java.applet.*;
public class applet extends Applet {
        public void start() {
                try { BufferedReader fin = new BufferedReader(
                        new InputStreamReader( new FileInputStream(
"SYSTEM.DAT")));                        String line;
                while( (line = fin.readLine()) != null) {
                        // ...process line ...
                }
                fin.close();
                } catch( Exception e) { }
        }
}
%
```

Using jdis on the applet class file, the user can analyze the disassembly:

```
% jdis -f applet.class

public class applet extends java.applet.Applet

Method public void <init>()
0 aload_0
1 invokespecial #8 <Method: (void) java.applet.Applet.<init>()>
4 return

Method public void start()
        0 new #4 <Class: java.io.BufferedReader>
        3 dup
        4 new #6 <Class: java.io.InputStreamReader>
        7 dup
        8 new #5 <Class: java.io.FileInputStream>
        11 dup
        12 ldc #1 <String: "SYSTEM.DAT">
        14 invokespecial #11 <Method: (void)
java.io.FileInputStream.<init>(java.lang.String)>
        17 invokespecial #9 <Method: (void)
        java.io.InputStreamReader.<init>(java.io.InputStream)>
        20 invokespecial #10 <Method: (void) java.io.BufferedReader.
<init<(java.io.Reader)>
        23 astore_1
        24 goto 27
        27 aload_1
        28 invokevirtual #13 <Method: (java.lang.String)
java.io.BufferedReader.readLine()>
        31 dup
        32 astore_2
        33 ifnonnull 27
        36 aload_1
        37 invokevirtual #12 <Method: (void) java.io.BufferedReader.close()>
        40 goto 44
        43 pop
        44 return
        }
```

Based on the above, a VDL is created to match "SYSTEM.DAT" followed within 80 bytes by "FileInputStream". Note below that using the VDL in the general CDL scan engine it does not match, however using it in the jadevdl engine it does match the class file:

```
% cat system.vdl
:system, "\"SYSTEM.DAT\"", @-80, "FileInputStream" #
% VFind --liboff='*' --vdl=system.vdl applet.java applet.class |& grep VIRUS
% VFind --liboff='*' --jadevdl=system.vdl applet.java applet.class |& grep VIRUS
##==>>>> VIRUS POSSIBLE IN FILE: "applet.class"
##==>>>> VIRUS ID: CVDL system
##==>>>> VIRUS END OFFSET: 468, matched: FileInputStream
%
```

4.5 MD5/CIT Engine

The md5 engine is an MD5-hash-based[3] engine for efficient whole-file matches using hash databases. The engine is related to the VSTK CIT program (Cryptographic Integrity Tool), which can be used to create MD5 databases. The "dangerfile" distributed with cit is an example of an md5 database. In general, for use with the md5 engine, a database should contain one line per hash consisting of 32 hex digits (the file hash or "signature") followed by a blank or tab, followed by the file name. Empty lines and lines starting with # are ignored, and any trailing blanks, tabs, carriage-returns, and new lines are ignored. The following example shows a script used to create an md5 database from a set of files using either CIT or OpenSSL[4]:

```
% cd examples/md5
% cat mkdb.sh
#! /bin/sh
# read file names from stdin and create md5 db while read pathname
do
        # use cit or openssl
        h='cit "$pathname"'
        #h='openssl md5 "$pathname" | awk '{ print $2 }''
        fname='basename "$pathname"'
        echo "$h $fname"
done

% find ../cbayes/z -name '*.txt' -print | ./mkdb.sh > z.db
% cat z.db dab3b1bc468c2a147f705a016b88842f z1.txt
373cee1606a140f8a436e0dbb0d7698e z2.txt
e0dccb5ed73f5e1f1373f7c752109eee z3.txt
07bf0b1b489b6086469cce20a60ae9ca z4.txt
4b6daf7ef35a5a704aa389e5c89e43cb z5.txt
5725d4a05911e0fb73a617503d368c4e z6.txt
39e8b6d60503540c9414e9d687a3038f z7.txt
dd9c712d9be1623e001823d6cd2ed35b z8.txt
62b8650e7f7eac92e53217c3b6b4e6e6 z9.txt
```

Now use VFind to scan a collection of files, and verify the hits:

```
% find ../cbayes -type f -print | VFind --liboff='*' --md5=z.db |& grep VIRUS
##==>>>> VIRUS POSSIBLE IN FILE: "../cbayes/z/z1.txt"
##==>>>> VIRUS ID: MD5 z1.txt
##==>>>> VIRUS POSSIBLE IN FILE: "../cbayes/z/z2.txt"
##==>>>> VIRUS ID: MD5 z2.txt
##==>>>> VIRUS POSSIBLE IN FILE: "../cbayes/z/z3.txt"
##==>>>> VIRUS ID: MD5 z3.txt
##==>>>> VIRUS POSSIBLE IN FILE: "../cbayes/z/z4.txt"
##==>>>> VIRUS ID: MD5 z4.txt
##==>>>> VIRUS POSSIBLE IN FILE: "../cbayes/z/z5.txt"
##==>>>> VIRUS ID: MD5 z5.txt
##==>>>> VIRUS POSSIBLE IN FILE: "../cbayes/z/z6.txt"
##==>>>> VIRUS ID: MD5 z6.txt
##==>>>> VIRUS POSSIBLE IN FILE: "../cbayes/z/z7.txt"
##==>>>> VIRUS ID: MD5 z7.txt
##==>>>> VIRUS POSSIBLE IN FILE: "../cbayes/z/z8.txt"
##==>>>> VIRUS ID: MD5 z8.txt
##==>>>> VIRUS POSSIBLE IN FILE: "../cbayes/z/z9.txt"
##==>>>> VIRUS ID: MD5 z9.txt
```

4.5.1 Hash Theory and Collisions

Cryptographic hash functions like MD5 are designed to take an arbitrary amount of input m and produce a fixed-length number (128 bits for MD5) H using a one-way transformation, H = h(m). One-way means that the inverse function h-1(H) does not exist, and it is not possible to reproduce the original input m from the hash. A good hash function should produce random looking results, uncorrelated for similar inputs. If it is likely that different inputs may produce the same hash value, i.e. a collision, that would lead to false hits in scanning.

An important property of a good hash function, for use with signature-based scanning, is that it should be very unlikely to find an input m2 which hashes to a given h(m1). Let n represent the number of bits in a hash, with N = 2n the total number of possible hash values. Given a random input m2, the probability that it produces the same hash as m1 is 1/N, i.e. 2-128 for MD5, which is extremely unlikely.

It can also be considered how many random inputs must be examined in order for the probability of a collision with a specific hash to reach 1/2. For k inputs there is approximately 1/2 = k(1/N), and solving for k yields:

$(4.1)\ k = N/2\ .$

For MD5, that is 2127 inputs, a very large number. For a more rigorous derivation of this and other results, see Stallings[5] page 341.

But in the overall scheme of things, input files are being examined and a decision is being made for one reason or another that some are "bad", so that MD5 signatures may be created for those, and some files are "good", so it is hoped that those do not have a collision with any of existing signatures. It is in this broader context that perhaps the most important property of a good hash function arises, that is: it is unlikely to find two different inputs which produce the same hash value.

Using an argument from Kaufman, et.al.[6] page 103, with k inputs, there are k(k - 1)/2 distinct pairs of inputs. For each pair, there is a probability of 1/N of a collision, so to reach a probability of 1/2 there is approximately 1/2 = (k(k - 1)/2)(1/N), and solving for k yields:

$(4.2)\ k = \sqrt{N}\ .$

For MD5, that is 264 inputs, a fairly large number, considering that the odds of winning the top prize in a lottery and being killed by lightning in the same day are about 1 in 255 (see Schneier[7] page 18).

4.6 Exercises

1. Why is (4.1) only approximate? Perform a more rigorous derivation. Hint: solve for P(no collisions) = 1/2.

2. Why is (4.2) only approximate (besides treating (k - 1) as k)? Perform a more rigorous derivation.

End Notes

[1] *European Institute for Computer anti-virus Research. http://www.eicar.com/.*

[2] *Java. http://java.sun.com/.*

[3] *RFC 1321 - The MD5 Message-Digest Algorithm. ftp://ftp.rfc-editor.org/in- notes/rfc1321.txt.*

[4] *OpenSSL Project. http://www.openssl.org/.*

[5] *W. Stallings, Cryptography and Network Security, Principles and Practice, Third Edition. Prentice Hall, 2003.*

[6] *C. Kaufman, R. Perlman, and M. Speciner, Network Security, Private Communication in a Public World. Prentice Hall, 1995.*

[7] *B. Schneier, Applied Cryptography, Second Edition. Wiley, 1996*

Chapter 5 Bayes Scan Engine

The VFind cbayes (CyberSoft Bayes) scan engine uses probabilistic techniques to classify data as being in one of two opposite groups. For each set of opposite groups to be detected, the classification uses a database created from samples of both groups.

The groups are generically referred to as "bad" and "good", but the actual classification reported is based on the cbayes database file name, e.g. a hit using a database named spam.dat would be reported as:

```
##==>>>> VIRUS ID: CBAYES spam 1.000000000
```

5.1 Bayesian Classification Theory

Given some input data, the problem is to decide if the data should be classified as bad or good. The decision is based on a probability P computed using Bayes rule:

$P = P(bad|data) = P(data|bad)P(bad)/P(data)$.

The apriori P(bad) may be set to 0.5 if it is unknown. P(data) is basically just a scale factor which will cancel out when the ratio P/(1 - P) is computed below.

The data is decomposed into a set of n tokens, which for text data may be words, groups of words, or other features.
data = $\{t_1, t_2, ..., t_n\}$.

Then there is:
$P(data|bad)=P(t_1, t_2, ..., t_n|bad)$
$\approx P(t_1|bad)P(t_2|bad) ... P(t_n|bad)$,

where the approximation assumes that the tokens are independent; this is called naive Bayesian Classification. For text data, using tokens derived from consecutive groups of words helps to make the approximation better. Now consider P(good|data), using Bayes rule:

$1 - P = P(good|data) = P(data|good)P(good)/P(data)$,

and P(data|good) may be approximated as:

$P(data|good) \approx P(t_1|good)P(t_2|good) ... P(t_n|good)$.

By computing the ratio P/(1 - P) the the P(data) scale factor can be eliminated:

$$\frac{P}{1-P} = \frac{P(t_1|bad)\, P(t_2|bad) \ldots P(t_n|bad)P(bad)}{P(t_1|good)\, P(t_2|good) \ldots P(t_n|good)P(good)}$$

5.1.1 Token Probabilities

For each token t_i, i = 1, ..., n, the probabilities $P(t_i|bad)$ and $P(t_i|good)$ are estimated from counts of occurrences of t_i in bad and good training data:

$b_i = count(t_i, bad)/count(bad)$ (5.8) $g_i = count(t_i, good)/count(good)$, (5.9)

where count(bad) is the number of bad data sets in the training data, and count(t_i, bad) is the number of occurrences of t_i in the bad training data. The count ratios can turn out larger than 1, but will be normalized below. b_i and/or g_i may be 0, so to avoid multiplying or dividing by zero the ratio r_i is computed:

$r_i = 1/2$, if $b_i = g_i$ (5.10)
$= b_i/(b_i + g_i)$, if $b_i \neq g_i$,

and define:

pi = max(Pmin, min(Pmax, ri))
qi = 1- pi

5.2 Text Tokenization and Hashing

Bayesian classification may be performed on any type of data, but this section will concentrate on text data and the tokenization and hashing utilities currently available in the CyberSoft CBayes toolkit. Operation of these utilities is described below: chash, chash-merge, chash-combine, chash2-merge, cbayes.

The CBayes utilities were designed to facilitate an adaptive sliding window approach to Bayes training and scanning. For detecting e-mail spam for example, collections of spam and clean hash files may be archived on a daily basis using chash. Then a window of some number of past days spam and clean archives would be merged into cumulative hash files using chash-merge. The cumulative spam and clean hash files would then be combined into one file using chash-combine. Combined hash files may themselves be merged with other combined files, e.g. from archives on multiple e-mail servers, using chash2-merge. The combined hash file is then converted by cbayes into a scan database for use with VFind.

To create a CBayes scan database, start with a set of bad and good sample training files, and split each file into words (i.e. tokens). The set of delimiters separating tokens used by emphCHash are defined by this C string:

```
" \t\r\n\f\v\"#()*+,/:;<=>?@[\\]^'{|}~"
```

The tokens are then hashed to produce fixed-size (i.e. 64-bit) data. Tokens are hashed individually as well as optionally in groups of sequential tokens. Token group level 3 is recommended and is the default for chash. Level 3 means that all individual words, as well as word pairs (pairs of sequential words), and word triplets (groups of three sequential words) will be tokenized and hashed. The hashed data is then stored in bad and good database files together with the counts of frequency of occurrence. The format of these individual bad and good hash files is called CHASH1. Sets of bad or good CHASH1 data may be merged into one file using chash-merge.

chash-combine is then used to combine one bad and one good CHASH1 file into a CHASH2 file representing all of the hashes and counts. chash2-merge can then be used to merge multiple CHASH2 filesif desired. Finally, the log differences log(pi) - log(qi) are pre-computed from the CHASH2 data and stored as a CBAYES1 database using cbayes.

During scanning, tokens and groups of tokens from the data being scanned are hashed and looked up in the cbayes database to obtain log(pi)- log(qi) which are accumulated to produce the overall probability.

5.3 Examples

z/*.txt are the "bad" files, containing data that will be used for Bayes training and eventually try to detect in new files.e/*.txt are the "good" training files, containing data opposite in some sense to the data in the "bad" files.

In these examples, the "bad" z/*.txt filesare actually just encyclopedic entries about zebras, and the "good" e/*.txt filesare entries about elephants.

```
% cd examples/cbayes
% ls -R
.:
dat e other z
./e:
e1.txt e2.txt e3.txt e4.txt e5.txt e6.txt e7.txt
./z:
z1.txt z2.txt z3.txt z4.txt z5.txt z6.txt z7.txt z8.txt z9.txt
# examine some of the z/ files
#
% foreach i (1 2 3)
? echo z$i.txt
? head -1 z/z$i.txt
? end z1.txt Zebras are members of the horse family native to central and southern
z2.txt The Plains Zebra (Equus quagga, formerly Equus burchelli) is the most z3.txt The
Mountain Zebra (Equus zebra) of southwest Africa tends to have a sleek
# examine some of the e/ files
#
% foreach i ( 1 2 3 4 )
? echo e$i.txt
? head -1 e/e$i.txt
? end
e1.txt Elephantidae (the elephants) is the only extant family in the order e2.txt An
elephant's most obvious characteristic is the trunk, a much elongated e3.txt Elephants
have three premolars and three molars in each quadrant. They erupt e4.txt Skin diseases
often occur, from which they try to protect themselves by
# create separate hash file for each z/ input text file
#
% foreach i (1 2 3 4 5 6 7 8 9)
? chash z/z$i.txt > dat/z$i.dat
? end
# note CHASH1 format
#
% head -6 dat/z1.dat
CHASH1 n 293 nfiles 1
DATA
00311bc8cf7920ed 1
0035dc18f0bbd4d7 1
# alternatively, create one hash file from all z/ input files
#
% chash z/z*.txt > dat/z-all.dat
#
# note CHASH1 format
#
% head -6 dat/z-all.dat
CHASH1 n 2219 nfiles 9
DATA
000fcfcc3008220c 1
0011ca19bf83690c 2
# check: chash-merge result is equivalent
#
% chash-merge dat/z?.dat | diff - dat/z-all.dat
---
# create separate hash file for each e/ input text file
#
% foreach i (1 2 3 4 5 6 7)
? chash e/e$i.txt > dat/e$i.dat
? end
# create merged hash file
#
% chash-merge dat/e?.dat > dat/e-all.dat
# note CHASH1 format
#
% head -6 dat/e-all.dat
CHASH1 n 2091 nfiles 7
DATA
0007bbfb33ad6b5a 2
0008267ef6acabe5 1
---
# combine z and e hash files into a CHASH2 database
#
% chash-combine dat/z-all.dat dat/e-all.dat > dat/ze-all.dat
# note CHASH2 format
#
% head -7 dat/ze-all.dat
CHASH2
n 4172 badfiles 9 goodfiles 7
```

```
DATA
0007bbfb33ad6b5a 0 2
0008267ef6acabe5 0 1
# create Bayes database for use in scanning
#
% cbayes dat/ze-all.dat > dat/ze-cbayes1.dat
# note CBAYES1 format
#
% head -14 dat/ze-cbayes1.dat
CBAYES1 n 4163 badfiles 9 goodfiles 7 group 3 notell 0
Ethresh 0
Hthresh 1
Pbad 0.5
Pdetect 0.99
Pmin 0.01
DATA
0007bbfb33ad6b5a -4.5951199e+00
0008267ef6acabe5 -4.5951199e+00
---
# check: chash2-merge can merge CHASH2 databases
#
# first create two partial CHASH2 databases
#
% mkdir tmp
% chash-merge dat/z{1,2,3,4,5}.dat > tmp/z1-5.dat
% chash-merge dat/z{6,7,8,9}.dat > tmp/z6-9.dat
% chash-merge dat/e{1,2,3,4}.dat > tmp/e1-4.dat
% chash-merge dat/e{5,6,7}.dat > tmp/e5-7.dat
% chash-combine tmp/z1-5.dat tmp/e1-4.dat > tmp/ze1.dat
% chash-combine tmp/z6-9.dat tmp/e5-7.dat > tmp/ze2.dat
# now merge and compare with dat/ze-all.dat#
% chash2-merge tmp/ze1.dat tmp/ze2.dat | diff - dat/ze-all.dat
% rm -r tmp
---
# use VFind and ze-cbayes1.dat, scan original training files
#
% VFind --liboff='*' --cbayes=dat/ze-cbayes1.dat z/*.txt e/*.txt |& grep VIRUS
##==>>>> VIRUS POSSIBLE IN FILE: "z/z1.txt"
##==>>>> VIRUS ID: CBAYES ze-cbayes1 1.000000000
##==>>>> VIRUS POSSIBLE IN FILE: "z/z2.txt"
##==>>>> VIRUS ID: CBAYES ze-cbayes1 1.000000000
##==>>>> VIRUS POSSIBLE IN FILE: "z/z3.txt"
##==>>>> VIRUS ID: CBAYES ze-cbayes1 1.000000000
##==>>>> VIRUS POSSIBLE IN FILE: "z/z4.txt"
##==>>>> VIRUS ID: CBAYES ze-cbayes1 1.000000000
##==>>>> VIRUS POSSIBLE IN FILE: "z/z5.txt"
##==>>>> VIRUS ID: CBAYES ze-cbayes1 1.000000000
##==>>>> VIRUS POSSIBLE IN FILE: "z/z6.txt"
##==>>>> VIRUS ID: CBAYES ze-cbayes1 1.000000000
##==>>>> VIRUS POSSIBLE IN FILE: "z/z7.txt"
##==>>>> VIRUS ID: CBAYES ze-cbayes1 1.000000000
##==>>>> VIRUS POSSIBLE IN FILE: "z/z8.txt"
##==>>>> VIRUS ID: CBAYES ze-cbayes1 1.000000000
##==>>>> VIRUS POSSIBLE IN FILE: "z/z9.txt"
##==>>>> VIRUS ID: CBAYES ze-cbayes1 1.000000000
%
# create some new samples, s1 with text similar to zebras
# s2 with text about elephants
#
% fortune -i -m horse > other/s1.txt
% fortune -i -m elephant > other/s2.txt
# scan: note s1.txt is detected
#
% VFind --liboff='*' --cbayes=dat/ze-cbayes1.dat other/* |& grep VIRUS
##==>>>> VIRUS POSSIBLE IN FILE: "other/s1.txt"
##==>>>> VIRUS ID: CBAYES ze-cbayes1 1.000000000
%
```

5.4 Exercises

1. Assume that the first lines from the three z/ and four e/ files shown in Section 5.3 are Bayes training data. Using Pmin = 0.01 and Pmax = 0.99, calculate pi for each of the following four single word tokens: Zebra, the, most, is. Using P(bad) = 0.5, calculate the overall probability that data containing those four words is bad.

2. For each of the dat/z?.dat and dat/e?.dat files, create a Bayes database which does not include that file, then use it to scan the associated omitted text file to determine if it is similar to the others. For example, use a Bayes database built from dat/z1-8.dat and dat/e1-7.dat to scan z/z9.txt. There are 16 text files, so 16 separate Bayes databases and scan tests must be performed. Hint: write a shell script to automate the testing.

3. Select some spam and clean e-mail samples to create a Bayes database. Test the database on the original samples and on new mail. Hint: uad -M is useful for extracting components from e-mail.

Chapter 6 Low-Level CVDL

This chapter covers the CVDL operators used at the lowest level of scanning, i.e. at the byte level of data, and sequences of bytes.

6.1 Low-level Or '|'

The low-level or operator '|' specifies the occurrence of alternative patterns at a position relative to the preceding pattern in the scanned data. For example:

```
"x", ("a" | "b"), "y"
```

matches "x", followed by "a" or "b", followed by "y".

'|' has higher precedence than concatenation (the comma operator, Section 3.4), so the parentheses in the above example are not needed. Multiple '|' patterns may be specified together, for example:

```
"x", "a" | "b" | "ZZZ" | "", "y"
```

matches "x", followed by "a" or "b" or "ZZZ", followed by "y", or "x" directly followed by "y" (matching the empty pattern "" in the or).

Do not confuse the low-level '|' operator with the high-level OR operator discussed later in Section 8.1. The high-level OR operator specifies the occurrence of alternative patterns at any positions in the scanned data.

6.2 Byte Expressions (bytes, byte ranges, characters, decimal, hex)

Bytes, byte ranges, and compliments of bytes and byte ranges can be specified using characters and decimal or hex integers. Characters are written using 'single quotes'. Hex integers may be written as 0xdd or '\xdd' where dd represents two hexadecimal digits (from the set 0..9, a..f, A..F). Byte ranges are written in the form start-end, and complement is specified using a circumflex (^) prefix.

Examples:

```
65
0x41
'\x41'
'A'
```

any of the above matches a byte whose value is 65 (decimal)

```
'a'-'f'
```

the above matches a byte whose value is in the range 0x61-0x7a

```
^0-10
```

the above matches a byte whose value is not in the range 0-10

6.3 Set Expressions {braces}

Sets of string or byte patterns are specified by enclosing the patterns in { braces }. For example:

```
{ "abc", 0-9 } matches any one of the bytes 'a', 'b', 'c', 0, 1, ..., 9
```

Sets can be complimented using ^:

```
^{ "abc", 0-9 } matches a byte which is not in the set 'a', 'b', 'c', 0, 1, ..., 9
```

Sets can also contain other sets or complimented sets.

6.4 Fuzzy Expressions (plus/minus offsets)

Fuzzy expressions are a convenient way of specifying ranges for bytes and strings. Fuzziness is specified using integers which represent plus and minus offsets. Examples:

```
FUZZY 2 100
FUZZY +-2 100are the same as: 98-102 fuzzy -2 +2 100
```

```
FUZZY -2 +3 "cow"is the same as: 'a'-'f', 'm'-'r', 'u'-'z'
```

Recognized case-insensitive spellings for the fuzzy operator are FUZZY and FUZZ.

6.5 Repetition Expressions

Multiple occurrences of bytes, byte ranges, sets, strings, or case-insensitive strings can be specified by using [number] after the expression. For example:

```
15[20]
0xF[0x14]
```

either of the above forms matches 20 occurrences of the byte value 15

```
0-10[40]
```

the above matches a sequence of 40 bytes whose values are in the range 0..10 "X"[3] matches "XXX"

The repetition expression can also specify a range, for example: "X"[0-3] matches from 0 to 3 occurrences of "X"

6.6 Indefinite Offsets (.*)

The '.*' operator is used to specify an indefinite offset, up to the next new line ("\n") character. This can be used when scanning text data to match patterns which must be restricted to a single line of text. Example:

```
~"\nSubject:", .*, ~"herbal", .*, ~"viagra"
```

will match "\nSubject: 100% Pure Herbal Potent Viagra On Sale!"

As with the '@' operator (see Section 3.4), it does not make sense to use '.*' at the beginning of a VDL segment, and such use will be reported as a parser error using the VFind -d,-dup-check option (see Section 2.1).

6.7 Absolute Offsets (ABS #)

The 'ABS' operator is used to specify an absolute offset from the beginning of the scanned data to match a pattern. The 'EOD' operator is used to match exactly at end-of-data. Example VDLs:

```
; match Bourne shell script file header
;
:a1, ABS 0, "#!", WS0, "/bin/sh" #
; match "abc" followed by "def" within the next 20 bytes,
; and "01234" at absolute position 14
;
:a2, "abc", @0-20, "def" AND ABS 14, "01234" #
; match the 8-byte Microsoft signature header which appears
; at the very beginning of most Microsoft application files,
; and "\xFE\xCA" anywhere in the scanned data
;
:MS/VBA, ABS 0, "\xD0\xCF\x11\xE0\xA1\xB1\x1A\xE1" AND "\xFE\xCA" #
; match "zzz" only if it appears exactly at the end of the scanned data
;
:a3, "zzz", EOD #
; match a scanned data file which is exactly 4 bytes long and contains "abc\n"
;
:a4, ABS 0, "abc\n", EOD #
```

Section 4.1 showed an example VDL to match the EICAR[1] test pattern anywhere in a file. But a more precise definition of the EICAR test says: the file starts with the following 68 characters, and is exactly 68 bytes long. Using absolute offsets, the following VDL precisely matches that definition:

```
:EICAR TEST -- Fake Virus -- IGNORE, ABS 0, "\x58\x35\x4f\x21\x50\x25\x40\x41\
\x50\x5b\x34\x5c\x50\x5a\x58\x35\x34\x28\x50\x5e\x29\x37\x43\x43\x29\x37\x7d\
\x24\x45\x49\x43\x41\x52\x2d\x53\x54\x41\x4e\x44\x41\x52\x44\x2d\x41\x4e\x54\
\x49\x56\x49\x52\x55\x53\x2d\x54\x45\x53\x54\x2d\x46\x49\x4c\x45\x21\x24\x48\
\x2b\x48\x2a", EOD #
```

But a further refinement of the EICAR specification states: The first 68 characters is the known string. It may be optionally appended by any combination of white space characters with the total file length not exceeding 128 characters. The only white space characters allowed are the space character, tab, LF, CR, CTRL-Z. Using a combination of absolute offsets, a set, and repetition expression, the following VDL precisely matches that definition:

```
:EICAR TEST -- Fake Virus -- IGNORE, ABS 0, "\x58\x35\x4f\x21\x50\x25\x40\x41\
\x50\x5b\x34\x5c\x50\x5a\x58\x35\x34\x28\x50\x5e\x29\x37\x43\x43\x29\x37\x7d\
\x24\x45\x49\x43\x41\x52\x2d\x53\x54\x41\x4e\x44\x41\x52\x44\x2d\x41\x4e\x54\
\x49\x56\x49\x52\x55\x53\x2d\x54\x45\x53\x54\x2d\x46\x49\x4c\x45\x21\x24\x48\
\x2b\x48\x2a", {" \t\r\n\x1a"}[0-60], EOD #
```

6.8 Offset Groups

Offset ranges, indefinite offsets, and absolute offsets can be applied to groups of patterns, resulting in simpler and more readable VDLs. Examples:

The VDLs:

```
:o1, "t1", @-10, ("t2"|"t3") #
:o2, "t1", .*, ("t2"|"t3") #
:o3, ABS 0, ("t2"|"t3") #
```

are equivalent to:

```
:o1, "t1", (@-10,"t2"|@-10,"t3") #
:o2, "t1", (.*,"t2"|.*,"t3") #
:o3, ABS 0,"t2"|ABS 0,"t3" #
```

6.9 Exercises

Select some spam and clean e-mail samples and write VDLs to match the spam samples. Run the VDLs on new mail to split it into spam and clean samples, then use those samples to create a Bayes database.

End Notes

[1] *European Institute for Computer anti-virus Research. http://www.eicar.com/.*

Chapter 7 CVDL and Text

Although VFind treats all data as binary, this just means that no data is skipped when scanning, not even line terminators that may appear in text files. A text file is simply a special case of a binary file, one that tends to not have 8-bit or control characters, with human readable data organized by lines and separated carriage-return and/or new line characters. For scanning text data, CVDL provides a set of specialized operators described in this chapter.

CVDL also provides a file type restriction directive, described in Section 11.1, that can be used to restrict the scope of VDLs to be applied only to text files, which can reduce scanning time and the potential for false hits.

7.1 White space (W0, W1)

There are two special operators, 'W0' and 'W1', for offsets consisting of white space characters as defined by the C library isspace() function, i.e. characters "\t" (horizontal tab), "\n" (new line), "\v" (vertical tab), "\f" (form feed), "\r" (carriage return), and " " (space).

`W0`
matches zero or more white space characters

`W1`
matches one or more white space characters

Examples:

```
"From:", W0, ~"<spam@spam.com>"
~"Bulletproof", W1, ~"Web", W1, ~"Hosting"
```

7.2 White space (shell)(WS0, WS1)

There are two special operators, 'WS0' and 'WS1', for offsets consisting of characters which are treated as white space by the UNIX shells, i.e. characters " " (blank), "\t" (horizontal tab), and "\\\n" (backslash new line):

`WS0`
matches zero or more shell white space characters

`WS1`
matches one or more shell white space characters

Examples:

```
"/bin/rm", WS1, "-rf", WS1, "/"
"cat", WS0, ">>", WS0, "/etc/passwd"
```

7.3 White space and Punctuation (WP0, WP1)

There are two special operators, 'WP0' and 'WP1', for offsets consisting of white space and punctuation characters as defined by the C library isspace() and ispunct() functions. The punctuation characters are: ! " # $ % & ' () * + , - . / : ; < = > ? @ [\ ] ^ _ ' { | } ~

`WP0`
matches zero or more white space or punctuation characters

`WP1`
matches one or more white space or punctuation characters

Example:

```
~"bomb", WP1
```
matches "bomb " and "bomb. " but does not match "bombastic" or "bombproof".

7.4 Skipping White space and Punctuation (~~)

The '~~' operator was designed for matching phone numbers, and is an extension of the case- insensitive string matching '~' operator. '~~' performs case-insensitive string matching while skipping any white space and punctuation characters. White space and punctuation characters are those defined by the C library isspace() and ispunct() functions. Examples:

```
~~"123-456-7890"
```
matches "(123)-456-7890" and "123.456.7890" and "1 2 3 - 4 5 6 - 7 8 9 0", etc.

```
~~"800 FREE CAR"
```
matches "(800) - F r e e C a r !!!", etc.

7.5 Wildcard White space (~W)

The '~W' operator was designed for case-insensitive matching for sequences of words separated by white space, where white space characters are those defined by the C library isspace() function. The " " (blank) character acts as a white space wildcard in the VDL, and will match one or more white space characters in the data being scanned. Examples:

```
~W" this is a test"
```
matches " this \r\nis a test" and matches "\nthis \r\nis a test"

```
~W"\nthis is a test"
```
matches "\nthis \r\nis a test" but does not match " this \r\nis a test"

7.6 Digits (\d+)

The \d+ operator matches one or more digits, and is named after the similar Perl operator. This can be used, for example, to detect obfuscated URLs:

```
"http://", \d+, "/"
```
matches URLs like http://3626287830/

```
"http://0", \d+, ".", "0", \d+, ".", "0", \d+, ".", "0", \d+, "/"
```
matches URLs like http://00000325.0000030.00000341.00000116/

Which can also be written using a macro:

```
$define zerod "0", \d+
"http://", $zerod, ".", $zerod, ".", $zerod, ".", $zerod, "/"
```

CVDL macros are discussed in detail in Chapter 10.

7.7 Only Digits (~#)

The'~#' operator matches only digits, skipping all other characters, over a default maximum range of 30 bytes of scanned input data. The maximum range of scanned input data can be specified by placing a number between '~#' and the digit string.

Examples:
```
~#"code 1234 sub-code 567"
```

matches the digits 1234567 in sequence, regardless of any intervening non-digit characters, over any 30 byte range of scanned input data, e.g. it will match "1abc2efg34---5 6 7"

```
~#60"code 1234 sub-code 567"
```
As above, but over a maximum range of 60 bytes of input data.

7.8 Floating-point Comparison (%f>)

The operator pair '%f>' can be used to extract a floating-point value from scanned text data and compare it to a constant. It will match if the value is greater than the constant. For example:

```
"Fuz1=", %f > 0.5
```

will match if the data following the string "Fuz1=" is a floating-point value greater than 0.5, e.g. it will match `"Fuz1=0.76"`.

'%f' should only be used in conjunction with a file-type restriction or preceding match so that the presence of floating-point data is assured. As the first pattern in a VDL it is only tried once, at the beginning of the scanned data buffer.

The '%f >' operator pair was first added to CVDL on an experimental basis for use in checking Bayesian spam probabilities, and has subsequently been found to be useful for checking other types of numerical text data. Eventually, floating-point comparison operators besides > (i.e. <, ==, <=, >=, !=) may also be added to CVDL.

7.9 Exercises

Write a VDL to detect the following data, including the two new line characters:

Mary had a little lamb
Its fleece was white as snow.

a) in MS/Win text;
b) UNIX text;
c) platform independent.

Chapter 8 High-Level CVDL - AND, OR, XOR and NOT

Originally, CVDL had only one high-level logic operator, '&', meaning AND. The current logical operators are "AND", "OR", "XOR", and "NOT", which allow construction of arbitrarily complex logic patterns. Also, a file size operator is available, and is used with comparison operators to produce logical values.

The precedence of the logic operators is, in order of decreasing precedence: NOT, AND, XOR, OR. Parentheses can be used to group operations into a desired execution order irrespective of operator precedence.

8.1 AND and OR

AND and OR are used to logically combine pattern matching results.Let t1...t4 represent any CVDL pattern, and consider the following VDLs:

```
:v12, t1 AND t2 #
:v34, t3 OR t4 #
```

v12 will match only if t1 matches anywhere in the scanned data and t2 also matches anywhere in the same scanned data. v34 will match if either t3 or t4 (or both) match.

Both of these are independent of the positions of the matches in the data; for example, t4 could match a pattern that precedes the pattern matched by t3. In contrast, the low-level or operator '|' discussed in Section 6.1 specifies the occurrence of alternative patterns at a position relative to the preceding pattern in the scanned data.

Some examples illustrating the precedence of the operators:

```
:ex1a, "pets" AND "cat" OR "dog" #
:ex1b, ("pets" AND "cat") OR "dog" # ; same as ex1b
:ex2a, "pets" AND ("cat" OR "dog") #
:ex2b, ("pets" AND "cat") OR ("pets" AND "dog") # ; same logically as ex2a
```

VDLs ex1a and ex1b are equivalent due to the higher precedence of AND as compared to OR, and will match data that contains either: the word "pets" anywhere and the word "cat" anywhere; or the word "dog" anywhere. If the intention was to match data containing the word "pets" anywhere, and either the word "dog" or the word "cat" anywhere, then parentheses can be used as in VDL ex2a and ex2b.

8.2 XOR and NOT

NOT by its self seems to be a strange logical operator for pattern matching, since it means that a match must have been made if some pattern is not present; however, consider a situation where all files or e-mail must contain a certain notice, and the user wants to detect the lack of that notice.

An example pattern is:

```
:missing, NOT "Copyright 2005, CyberSoft Operating Corporation" #
```

XOR is the exclusive-OR operator, which by definition can be expressed using AND and OR:

```
a XOR b == (a AND NOT b) OR (b AND NOT A)
```

For example, VDL ex3:

```
; guns or bullets, but not both
;
:ex3, ("guns" AND NOT "bullets") OR ("bullets" AND NOT "guns") #
can be expressed more efficiently using XOR:
:ex3, "guns" XOR "bullets" # ; guns or bullets, but not both
```

8.3 File Size Operator

The CVDL file size operator returns the size in bytes of the file or data being scanned, if known. It can be written as 'SIZE', 'Size', or 'size', and is used with the comparison operators: < >> != == <= >=

Examples:

```
:s0, size > 100 #
:s1, size == 5 AND "abc" #
:s2, size < 5 AND "abc" #
:s3, 5 > size AND "abc" #
:s4, ((10 <= size AND size <= 20) OR size > 1000) AND "abc" #
```

VDL s0 will match if the data size is greater than 100 bytes regardless of the contents of the data. For the other examples the data must contain the string "abc" in order to match.

VDL s1 will only match if the data size is exactly 5 bytes. VDLs s2 and s3 are identical and only match if the data size is less than 5 bytes. VDL s4 will match if the data size is between 10 and 20 bytes inclusive, or greater than 1000 bytes.

Note that if the data size is not available, the size operator will not match, regardless of the comparison operator used.

8.4 File Name Operator

The CVDL file name operator compares the quantified name of the file being scanned with an arbitrary low-level CVDL pattern, and returns true if the pattern is matched by the file name.

The quantified file name is the name reported by VFind's "Checking file:" report, including the double quotes. In a quotified file name, any literal double quotes will be displayed as \",and newline characters as \n. When processing SmartScan output from UAD, the file name will include both the top-level and component-level file names. For example:

```
% cd examples/hlcvdl
% VFind --liboff='*' abc |& grep 'Checking file:'
##==> Checking file: "abc"
% uad -ssw abc.tar | VFind --liboff='*' -ssr |& grep 'Checking file:'
##==> Checking file: "abc.tar" -> "ZZabcYY"
##==> Checking file: "abc.tar" -> "abc"
```

The three quotified file names reported above are: "abc" ""abc.tar" -> "ZZabcYY" "abc.tar" -> "abc" including the double quotes.

The file name operator consists of the word 'NAME' (case-insensitive), followed by ~=, followed by the pattern to match.

For example:

```
% cat name.vdl
:contains-abc, name ~= "abc" #
:exactly-abc, name ~= ABS 0, "\"abc\"", EOD #
:exactly-abc-regex, name ~= ~R"^\"abc\"$" #
:component-abc, name ~= "-> \"abc\"", EOD #
```

The first VDL will match any file name containing the three-letter substring "abc". The second VDL uses 'ABS' and 'EOD' (see Section 6.7) to match only if the reported file name is exactly equal to the five-letter string "abc", including the double quotes, and the third VDL uses a regular expression pattern (see Chapter 9) to perform the same match. The last VDL matches on a SmartScan component file name of "abc". Example runs:

```
% VFind --liboff='*' --vdl=name.vdl abc ZZabcYY |& grep 'VIRUS'
##==>>>> VIRUS POSSIBLE IN FILE: "abc"
##==>>>> VIRUS ID: CVDL exactly-abc-regex
##==>>>> VIRUS POSSIBLE IN FILE: "abc"
##==>>>> VIRUS ID: CVDL exactly-abc
##==>>>> VIRUS POSSIBLE IN FILE: "abc"
##==>>>> VIRUS ID: CVDL contains-abc
##==>>>> VIRUS POSSIBLE IN FILE: "ZZabcYY"
##==>>>> VIRUS ID: CVDL contains-abc
% uad -ssw abc.tar | VFind -ssr --liboff='*' --vdl=name.vdl |& grep 'VIRUS'
##==>>>> VIRUS POSSIBLE IN FILE: "abc.tar" -> "ZZabcYY"
##==>>>> VIRUS ID: CVDL contains-abc
##==>>>> VIRUS POSSIBLE IN FILE: "abc.tar" -> "abc"
##==>>>> VIRUS ID: CVDL component-abc
##==>>>> VIRUS POSSIBLE IN FILE: "abc.tar" -> "abc"
##==>>>> VIRUS ID: CVDL contains-abc
```

The two file names in this example, "abc" and "ZZabcYY", both contain the "abc" substring, so the contains-abc VDL always matches. Note that the exactly-abc VDLs match "abc" when scanned directly by VFind, but they do not match when file "abc" is a component extracted from a tar archive (which does match the component-abc VDL).

As an application of the file name operator for detecting side-effects of a virus infection, it has been noted that the W32.Mytob.FT@mm virus adds a registry value containing the string "skybotx.exe" to SYSTEM.DAT. This string on its own or appearing in any file other than SYSTEM.DAT would not be suspicious. So the following VDL matches only if the string appears in SYSTEM.DAT:

```
% cat system.vdl
:W32.Mytob.FT@mm SYSTEM,
name ~= { '"', '/', '\\' }, ~"system.dat\"" AND "skybotx.exe" #
```

The file name operator above will match a quotified case-insensitive system.dat file name (e.g. "SYSTEM.DAT") or path name (e.g."\WINDOWS\SYSTEM.DAT" or "/pc/export/system.dat").

8.5 Exercises

Select a set of system text files (e.g. man pages) and look through some of them for common words and phrases. Then create a set of VDLs using combinations of AND and OR operators, and scan all of the selected files using those VDLs.

Chapter 9 CVDL and Regular Expressions

Regular expressions (regex) are a powerful pattern matching mechanism originally designed as part of the UNIX operating system, and currently standardized by the Open Group[1]. Regex operates on character strings, i.e. sequences of non-zero bytes terminated by a zero byte, and is generally line-oriented, so it is suitable only for pattern matching in text.

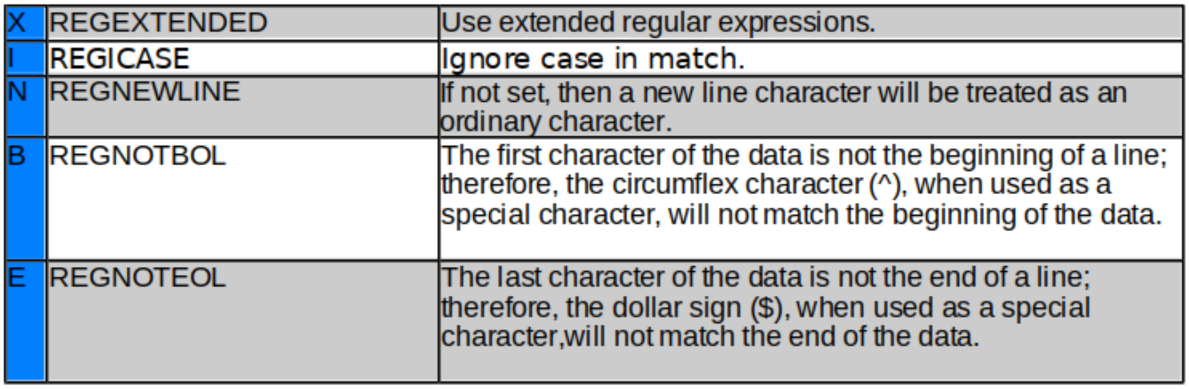
VFind and CVDL were designed for fast pattern matching on binary or text data. Some of the low-level CVDL pattern matching operators are similar to regex, but CVDL is not restricted to text. As CVDL evolved to meet the needs of specialized scanning, particularly for use in detecting e-mail spam by SafeInternetEmail[2], more operators similar to regex were added. Eventually, a direct interface to the complete capabilities of regex was added.

Note that although VFind and CVDL operate on arbitrary binary data, regex only works on text. Before invoking a regex during scanning, VFind temporarily appends a zero byte to the buffer being scanned so that the regex does not run off of the end.

This chapter describes the use of regex with CVDL, including summaries of basic and extended regular expressions. For more information about regex, recommended references are the O'Reilly book by Friedl[3], the Open Group[4], and the UNIX man pages for regcomp(3C), regexec(3C), and regex(5).

9.1 Regex Operator '~R' operator

CVDL regular expression matching is specified using the '~R' operator followed by zero or more single-letter regex option settings, followed by a string containing the regex pattern. Option settings correspond to the flags passed to regcomp(3C) to compile the pattern:



X	REGEXTENDED	Use extended regular expressions.
I	REGICASE	Ignore case in match.
N	REGNEWLINE	If not set, then a new line character will be treated as an ordinary character.
B	REGNOTBOL	The first character of the data is not the beginning of a line; therefore, the circumflex character (^), when used as a special character, will not match the beginning of the data.
E	REGNOTEOL	The last character of the data is not the end of a line; therefore, the dollar sign ($), when used as a special character,will not match the end of the data.

If no regex options are specified, the result is a case-sensitive basic regular expression with new line treated as an ordinary character. Note that a regular expression as the first pattern in a VDL is only applied once, at the beginning of the scanned data buffer.

A useful feature of basic regex, which is not available in extended regex, is the ability to tag subexpressions and then use back-references to the tags. For example:

```
% cd examples/regex
% cat bob.msg
From: somebody@somewhere.com
To: cyberbob@cyber.com
Subject: welcome to spam heaven CyberBob

hi
% cat r1.vdl
:r1, ~"\nTo:", W0, ~RI"\\([^@\n]*\\)@.*\\1" #

% VFind --liboff='*' --vdl=r1.vdl bob.msg |& grep VIRUS
##==>>>> VIRUS POSSIBLE IN FILE: "bob.msg"
##==>>>> VIRUS ID: CVDL r1
##==>>>> VIRUS END OFFSET: 92, matched: heaven CyberBob
%
```

The first part of VDL r1 matches a new line followed by "To:" (case-insensitive) followed by zero or more white space characters. The regex part produces the case-insensitive pattern "\([^@\n]*\)@.*\1" which is intended to match the userid portion of an e-mail address occurring again anywhere after the initial occurrence. It works as follows: after matching "To:", the regex sub-expression [^@\n] matches any sequence of characters up to the first @ or \n, and that match is tagged since it is enclosed by \(and \). That match would be the userid portion of the mail address.

Then the rest of the pattern @.*\1 matches @ followed by zero or more of any characters followed by an occurrence of the previously tagged sub-expression, reference by \1 in the regex.

Note that a regular expression is specified as a CVDL string, so it is subject to interpretation of escape sequences (see Section 3.3). This is why the example above used two backslashes to produce one. Also, \n in the regex string produced a literal new line in the pattern.

9.2 Trigger Data

As discussed in detail in Chapter 13, many CVDL operators, including regex, are not able to be implemented in the VFind parallel search engine. To improve run-time speed for regex, the regex string may optionally be followed by a parenthesized expression containing trigger data for the fast/parallel search. The only CVDL elements accepted in the trigger data are strings, concatenation, low-level or, parentheses, and AND (see Section 12.4).

For example:

```
:r2, ~RX"abcd.*(efgh|xxxx).*zzzz" ( "abcd", "efgh" | "xxxx", "zzzz" ) #
```

VDL r2 specifies an extended regular expression, including trigger data. The regex will not be triggered to run unless the parallel search engine first encounters the trigger data.

9.3 Summary of Basic Regular Expressions

Characters in a basic regular expression are classified as either ordinary or special. The special characters are .][\ * ^ $ and all other characters are ordinary. A special character preceded by a backslash becomes an ordinary character that matches the special character itself. The ordinary characters ( ) { } 1 2 3 4 5 6 7 8 9 become special when preceded by a backslash.

A bracket expression is a set of characters enclosed in [square brackets]. It matches any single character from the set, unless the first character in the set is a circumflex (^), which negates the set so it matches any single character not in the set. A range of characters can be specified using a hyphen, for example '[0-9]' is the same as '[0123456789]'. In a bracket expression, the special characters . *][\ lose their special meaning and become ordinary characters.

In a bracket expression the following character class expressions are supported: [:alnum:], [:alpha:], [:blank:], [:cntrl:], [:digit:], [:graph:], [:lower:], [:print:], [:punct:], [:space:], [:upper:], [:xdigit:]. So, for example, '[0-9]' may be alternatively written as '[[:digit:]]'.

A period (.), when used outside a bracket expression, matches any single character. A character followed by an asterisk (*) matches zero or more consecutive occurrences of the character. Thus '.*' matches zero or more of any characters, i.e. it matches anything or nothing. For example, "a.*b" would match "ab", "aZb", "aZZb", "aXYZXYZb", etc.

The circumflex (^) and dollar sign ($) match the beginning and end of data respectively, unless the REG NEWLINE option is set, then they match the beginning and end of a line respectively.

A subexpression can be defined by enclosing it between the character pairs \(and \). The back-reference expression \n, where n is a digit 1..9, matches the same string of characters as was matched by the n'th subexpression.

A pattern matching a single character, a subexpression, or a back-reference can optionally be followed by an interval expression of the form \{m\}, \{m,\} or \{m,n\} to match repeated consecutive occurrences of the pattern. m specifies the exact or minimum number of occurrences and n specifies the maximum number of occurrences. \{m\} matches exactly m occurrences, \{m,\} matches at least m occurrences, and \{m,n\} matches any number of occurrences between m and n, inclusive.

9.4 Summary of Extended Regular Expressions

The rules specified for basic regular expressions also apply to extended regular expressions with the following exceptions: the characters | + ? are special; the { } characters, when used as the duplication operator, are not preceded by backslashes, \{ and \} simply match the characters { and }, respectively; the back reference operator is not supported; anchoring using ^ and $ is supported in subexpressions.

Two expressions separated by a vertical-line (|) match a string that is matched by either expression. In other words, | is the extended regular expression 'or' operator. An expression followed by a plus-sign (+) matches one or more consecutive occurrences of the pattern, and an expression followed by a question-mark (?) matches zero or one consecutive occurrences.

9.5 Exercises

1. Try to construct trigger data for the regex in VDL r1 from Section 9.1. Modify the problem and produce a regex from which good trigger data can be constructed. Hint: instead of matching the userid anywhere after the initial occurrence, match it in a Subject: header.

2. For VDL r2 in Section 9.2, what is the difference between patterns matched by the regex vs. patterns matched by the trigger data? Can an equivalent VDL be written without using the CVDL regex operator?

End Notes

[1] *http://www.opengroup.org/, ed., The Open Group Base Specifications Issue 6. IEEE Std 1003.1, 2004.*

[2] *SafeInternetEmail. http://safeinternetemail.com/.*

[3] *J.E.F. Friedl, Mastering Regular Expressions, Second Edition. O'Reilly, 2002.*

[4] *http://www.opengroup.org/, ed., The Open Group Base Specifications Issue 6. IEEE Std 1003.1, 2004.*

Chapter 10 CVDL Macros

CVDL includes the ability to define macros, which expand to their definition when invoked, and simplify the repeated use of patterns. For patterns which are used frequently in VDLs, there is also a storage advantage provided by macros, discussed in Section 10.4.

10.1 Defining VDL Macros

A VDL macro is specified using $define as the first word on a line, and the entire macro definition must be contained all on one line. The syntax is:$define name value where the line contains: optional leading white space, $define, white space, name, white space, value.

The macro name must start with a letter from the set {'a'-'z','A'-'Z','' } followed by 0 or more letters or digits. Note that macro names are case-sensitive. The value continues to the end of the line, and trailing white space is trimmed.

10.2 Using VDL Macros

VDL macros are invoked by specifying their name after a $ character. Macros are lexical tokens, which means that they can not be confused with other tokens, e.g. strings. Thus: "abc", $mac, ... invokes the macro named mac, but: "abc$mac", ... does not invoke any macro, and is simply a literal string.

VDL macros can be nested to unlimited depth, so macros can refer to other macros in their definition. Macros cannot be used in a VDL rule before being defined, but they can be used in other VDL macro $define's before being defined.

10.3 VDL Macro Examples

Some examples using AND and OR:

```
$define pf1 $pets AND $food
$define pets "dog" OR "cat"
$define food "fish" OR "pie"
$define pf2 ($pets) AND ($food)

:v1, $pf1 AND "ate" #
:v2, "ate" AND $pf2 #
```

Note that pf1 resolves to:"dog" OR "cat" AND "fish" OR "pie" which is the same as:"dog" OR ("cat" AND "fish") OR "pie" but pf2 resolves to:("dog" OR "cat") AND ("fish" OR "pie")

As with C/C++ #define macros, parentheses may be used in the VDL macro definition or invocation to ensure that the intended result is obtained.

For detecting spam, macros have been used to define alternative representations of letters which are frequently used for obfuscation, for example:

```
% cd examples/macros
% cat letters.vdl
; some spam letters
;
$define letterP {"Pp\xDE\xFE"}
$define letterI {"Ii1;!|:Ll\xa1\xa6\xcc\xcd\xce\xcf\xec\xed\xee\xef"}
$define letterL {"Ll1!|:;\xef\xa1\xa6"}
$define letterS {"Ss$2Zz\xa7\x8a\x9a"}
% cat spam.vdl
; pills
;
$define pillz $letterP,@-3,$letterI,@-3,$letterL,@-3,$letterL,@-3,$letterS
:vpillz, ~"discount", .*, $pillz #
% cat x.dat
DISCOUNT p-i-1-1-z
% VFind --liboff='*' --vdl=letters.vdl --vdl=spam.vdl x.dat |& grep VIRUS
##==>>>> VIRUS POSSIBLE IN FILE: "x.dat"
##==>>>> VIRUS ID: CVDL vpillz
##==>>>> VIRUS END OFFSET: 18, matched: COUNT p-i-1-1-z
%
```

For detecting viruses, macros have been used to specify variable machine code segments to match polymorphic samples. The following examples are from the VFind VDLs version VFind13-2005-05-25-09-36-46 macros.vdl and 166+ .vdl files:

```
$define junk (0xB8-0xBF,0x00-0xFF[2])|(0xEB,0x04-0x0F)|
(0x81,0xC0-0xC7|0xD0-0xDF|0xE8-0xF7,0x00-0xFF[2])
$define crypt (0x80,0xB4-0xB7,0x00-0xFF[3])
$define incofs (0x43|0x45|0x46|0x47)
$define inccnt (0x40|0x41|0x42|0x43|0x45|0x46|0x47)
$define cmpcnt (0x81,0xF8|0xF9|0xFA|0xFB|0xFD|0xFE|0xFF,0x00-0xFF[2])
$define jzend (0x74,0x00-0x7F) ; jumps only forward
$define loop (0xE9,0x00-0xFF,0x80-0xFF) ; jumps only backward
```

```
:Fono.poly,<".COM","EXE"> $junk,@-0x100,$crypt,$junk,@-0x100,$incofs,
$junk,@-0x100,$inccnt,$junk,@-0x100,$cmpcnt,$jzend,$junk,@-0x100,$loop#
```

Besides heavy use of macros, the Fono.poly VDL also specifies a file type restriction (see Section 11.1) so that it will only scan Microsoft executable files.

10.4 VDL Macro Storage Advantage

H2: VDL Macro Storage Advantage

The data associated with the definition of a macro is stored only once, regardless of how many times the macro is used. In contrast, identical data which is repeated in VDL definitions is stored separately for each occurrence.

For example, if the string "http://" is used 100 times in VDL definitions, that string will be stored separately 100 times, and use a total of 100*8 = 800 bytes storage (internally the string includes a terminating 0 byte at the end, so uses 8 bytes). But if the user defines a macro:

```
$define http "http://"
```

and use $http in the VDL definitions instead of the literal "http://" string, then the string is stored only once, regardless of how many times the macro is used. There is an internal 16-byte overhead associated with macro data storage, so for this example the total is 16+8 = 24 bytes, compared to 800 bytes without using a macro.

10.5 Exercises

1. Contrive an example using 50 VDLs all containing a particular string. Then create a compatible VDL file with the string replaced by a macro. Compare performance, both run-time CPU and RAM usage, with and without macros.

Chapter 11 CVDL Meta Operators

The CVDL meta operators control various features of scanning. They are specified either as directives which apply to all VDLs in a CVDL file, or applied for a single VDL.

The meta operators are written inside < angle brackets > and their general syntax is:

```
< "filetype", "filetype", ...>; list of file types to scan
< ! "filetype", "filetype", ...>; list of file types to not scan
< "version=nnnnnn" >; report version number nnnnnn
< "start=value", "limit=value" >; restrict scan range
< "notell" >; turn off reporting hits
< "sticky" >; stay on if hit
< "clear" >; clear sticky hit
```

When applied to a single VDL, the meta operators are specified directly after the comma following the VDL name, before the VDL definition. When specified as part of a VDL, the directives apply only for that particular VDL. When specified outside of a VDL, the directives have VDL file scope and apply only for VDL rules which appear following the directive in the same VDL file.

11.1 File Type Restriction Directives

File type restriction directives are written in the form:

```
< "filetype", "filetype", ...>; list of file types to scan
< ! "filetype", "filetype", ...>; list of file types to not scan
< >; resets to scan everything.
```

specifying a list of one or more file types to scan or to not scan. An empty directive resets to scan everything regardless of file type.

To utilize file type restrictions most effectively, UAD should be used with the SmartScan -ssw option, piped into VFind with the -ssr option. If run without UAD/SmartScan input, VFind does recognize the following file types on its own: "EXE", "OLE", "Java class file", "text", "text (8- bit)", ".COM", otherwise "unknown".

Note that the file type "unknown" is considered a valid type, since it means that the type is not one of the many types recognized which may be placed in a file type restriction directive, so something indeed is known about the type. In this case, all file type restrictions will be applied. For example, a VDL specifying a file type restriction of <"text"> will not scan a file type of "unknown".

Examples:

```
:v1,"..."# ; applied for all file types
<"text"> ; only "text" file types for the following vdls
:v2,"..."# ; only "text"
:v3,"..."# ; only "text"
<!"HTML"> ; no "HTML" file types for the following vdls
:v4,"..."# ; only non-"HTML"
:v5,<"JPEG","GIF">"..."# ; only "JPEG" and "GIF" file types
:v6,"..."# ; only non-"HTML"
>< ; all file types for the following vdls
:v7,"..."# ; applied for all file types.
```

Matching for file type restrictions is case-sensitive, and only requires that the VDL-restricted type be a substring of the file type.

11.2 VDL Version Reporting

Versions for VDL files and rules can be reported using an extension to the file type restriction syntax. If the user specifies a string starting with version= in a file type restriction directive, whatever follows the '=' character in that string will be printed as an informative message about the version of the VDL file or rule.

The following example specifies a version for the VDL file and a version for VDL rule 'b':

```
% cd examples/meta
% cat v.vdl

<"text","version=1.2.3">

:a, "abc"#
:b, <"version=9.9"> "bbb"#

% VFind --vdl=v.vdl hi
...
##==>> Loading VDL code from: v.vdl
##==>> All SmartScan file types disabled.
##==>> SmartScan file type '*text*' enabled.
##==>> VDL file 'v.vdl' Version: 1.2.3
##==> VDL model for 'a' loaded.
##==> VDL 'b' Version: 9.9
##==> VDL model for 'b' loaded.
##==> Checking file: "hi"
...
```

11.3 VDL start/limit specification

The start/limit specification is not a CVDL operator; instead, it fits in with the VDL <"meta- data">, like the file-type restrictions and version= specification.

For example:

```
:x, <"start=10","limit=20"> "abc" #
```

The VDL above will only scan data starting at offset 10 with a limit of 20 bytes scanned. In general, if start is not specified it will start at offset 0, and if limit is not specified there is no limit. Thus the user can use start and limit together, or just one of them. Where appropriate, by restricting the offset and size of data scanned by a VDL using the start/limit specification, VFind may run faster.

The start/limit specification can be mixed in with file type restrictions and the version specification.

For example:

```
% cat y1.vdl
:y1, <"text", "start=100", "limit=1000", "version=1.2.3"> "abc", @-100, "efg" #
% VFind --vdl=y1.vdl
...
##==>> Loading VDL code from: y1.vdl
##==> VDL 'y1' Version: 1.2.3
##==>> SmartScan file types enabled for VDL model 'y1': '*text*'
##==> VDL model for 'y1' loaded.
...
% cat y2.vdl
:y2, <!"GIF", "JPEG", "start=100"> "frog", @-100, "fish" #
% VFind --vdl=y2.vdl
...
##==>> Loading VDL code from: y2.vdl
##==>> SmartScan file types disabled for VDL model 'y2': '*JPEG*' '*GIF*'
##==> VDL model for 'y2' loaded.
```

11.4 VDL notell specification

The notell specification is not a CVDL operator; instead, it fits in with the VDL <"meta-data">, like the file-type restrictions and version= specification. It turns off reporting of individual patterns, the same as the VFind -notell= and -notells= command-line options.

For example:

```
% cat n.vdl
<"notell">
:x,"a"#
:y,<!"notell">"b"#
:z,"c"#
% cat n
cab
% VFind --vdl=n.vdl -p n
...
##==> VDL model for 'x' loaded (notell).
##==> VDL model for 'y' loaded.
##==> VDL model for 'z' loaded (notell).
...
##==> Checking file: "n"
##==>>>> VIRUS POSSIBLE IN FILE: "n"
##==>>>> VIRUS ID: CVDL y
11.5. META VDLS 81
##==>>>> VIRUS END OFFSET: 3, matched: cab
##==> Number of possible virus infections found in file "n": 1
```

The first notell specification turns off reporting for all subsequent VDLs in the n.vdl file; however, for VDL y only, the notell is overridden by a !"notell" specification, so VDL y will be reported. All three VDLs x, y, and z hit on file n, but only y is reported.

The notell specification can be useful in conjunction with meta VDLs which are discussed in the next section.

11.5 Meta VDLs

Meta VDLs match on the names of other VDL hits, not on the data being scanned. They were described previously in Section 4.5. Meta VDLs reside in a separate search engine and are loaded from a file using the -vdlm= option.

Continuing the notell example from the previous section, if a meta VDL is created to match on detection of VDLs x and z then it will match even though x and z are not reported:

```
% cat meta.vdl
:x and z, "CVDL x" and "CVDL z" #
% VFind --vdl=n.vdl --vdlm=meta.vdl -p n
...
##==> Checking file: "n"
##==>>>> VIRUS POSSIBLE IN FILE: "n"
##==>>>> VIRUS ID: CVDL y
##==>>>> VIRUS END OFFSET: 3, matched: cab
##==>>>> VIRUS POSSIBLE IN FILE: "n"
##==>>>> VIRUS ID: CVDL x and z
##==>>>> VIRUS END OFFSET: 6, matched: CVDL z
##==> Number of possible virus infections found in file "n": 2
```

In conjunction with notell VDLs, meta VDLs can be useful for detection with UAD/SmartScan input that decomposes archives into separate components. Since each component is scanned separately, it is not possible to create a regular VDL which would match across components. Notell VDLs can be created to match individual components, and then a meta VDL can be written to match on the notell hits.

For example:

```
% cat notell.vdl
<"notell">
:password, <"text"> ~"password" #
:image, <"text"> ~".gif", WP1 OR ~".jpeg", WP1 OR ~".bmp", WP1 #
:encrypted zip, <"encrypted zip"< 0-255 #
% cat meta2.vdl
:password and image and encrypted zip,
"CVDL password" AND "CVDL image" AND "CVDL encrypted zip" #
% uad -ssw -sst -M -z e.msg | VFind -ssr --vdl=notell.vdl --vdlm=meta2.vdl -p
...
##==> Checking file: "e.msg" -> "npxormprwo.gif"
##==> Checking file: "e msg" -> ""
##==> Checking file: "e.msg" -> "Your_complaint.zip"
##==> Checking file: "e.msg" -> "cnxkvl.exe"
##==> Checking file: "e.msg" -> "gchuttf.sys"
##==> Checking file: "e.msg" -> ""
##==>>>> VIRUS POSSIBLE IN FILE: "e.msg" -> ""
##==>>>> VIRUS ID: CVDL password and image and encrypted zip
##==>>>> VIRUS END OFFSET: 54, matched: L encrypted zip
##==> Number of possible virus infections found in file "e.msg": 1
```

" The password VDL matches the word "password" case-insensitively in any text component. The image VDL matches any of three typical image file extensions. The encrypted zip VDL demonstrates matching by file type; it will match the first byte in any component (since 0-255 is the range of all possible byte values) but is restricted to files of type "encrypted zip".

Hits on the notell VDLs will not be reported, but the VDL names will be accumulated for scanning by the meta VDL engine. The VDL in meta2.vdl simply requires all three notell VDLs to be present. The e.msg file shown scanned above is an actual e-mail virus sample where the password for the encrypted zip attachment is contained in the GIF image component. Besides the attachments, the text body of e.msg contained the following, which was matched by the password and image notell VDLs:

For security purposes the attached file is password protected. Password -- <img src="cid:npxormprwo.gif">

11.6 VDL sticky and clear specifications

VDL hits are normally reset to zero after processing a file, but there may be situations where one would want a VDL hit to be persistent. For example, if the components of the e.msg file from the previous example were extracted into separate files and then scanned, the meta VDL would not hit.

This case can be handled using the "sticky" specification, which makes VDL hits stick around for all subsequent scanned files, for use with meta VDLs. Sticky VDL hits can be cleared if necessary using a VDL with the "clear" specification.

Redoing the previous example, sticky.vdl is the same as notell.vdl except it has the "sticky" specification; meta2 clear.vdl is the same as meta2.vdl except it has the "clear" specification. The mail message components have been extracted into separate files in subdirectory e/, which also contains an unrelated file hi.txt:

```
% cat sticky.vdl
<"notell", "sticky">
:password, <"text"> ~"password" #
:image, <"text"> ~".gif", WP1 OR ~ ".jpeg", WP1 OR ".bmp", WP1 #
:encrypted zip, <"encrypted zip"> 0-255 #
% cat meta2_clear.vdl
<"clear">
:password and image and encrypted zip,
"CVDL password" AND "CVDL image" AND "CVDL encrypted zip" #
% ls e/
Your_complaint.zip body.html hi.txt
% uad -ssw -sst -z e/* | VFind -ssr --vdl=sticky.vdl --vdlm=meta2_clear.vdl -p
...
##==> Checking file: "e/Your_complaint.zip"
##==> Checking file: "e/Your_complaint.zip" -> "cnxkvl.exe"
##==> Checking file: "e/Your_complaint.zip" -> "gchuttf.sys"
##==> Number of possible virus infections found in file "e/Your_complaint.zip": 0
##==> Checking file: "e/body.html"
##==>>>> VIRUS POSSIBLE IN FILE: "e/body.html"
##==>>>> VIRUS ID: CVDL password and image and encrypted zip
##==>>>> VIRUS END OFFSET: 18, matched: L encrypted zip
##==> Number of possible virus infections found in file "e/body.html": 1
##==> Checking file: "e/hi.txt"
##==> Number of possible virus infections found in file "e/hi.txt": 0
```

Note that if the meta VDL did not contain the "clear" specification it would hit on hi.txt and all subsequent files due to the persistent sticky VDL hits.

11.7 Exercises

Select one spam message which has a combination of features suitable to be matched by notell and meta VDLs. Write the VDLs and test them on that message and a collection of other clean and spam messages received.

Chapter 12 CVDL Syntax Summary

CVDL syntax is described as a set of recursive parsing rules which are applied to "tokens" produced by lexical preprocessing. Preprocessing skips comments and non-literal white space, and expands macros.

We start with some definitions:

hex escape sequence \x or \X followed by two hex digits.

escape sequence \c or hex escape sequence. If character c is n, r, or t this represents new line, carriage-return, or tab, respectively; otherwise it represents character c itself.

INTEGER A decimal or hex integer (e.g. 1234, 0xa9BE, 0Xff) or a single character or escape sequence in single 'quotes', (e.g. 'a', '\n', '\x9f') or an escape sequence with no quotes.

BYTE An INTEGER in the range 0 to 255 inclusive.

DOUBLE A non-negative floating-point decimal value (e.g. 1.234)

STRING Zero or more characters or escape sequences in double "quotes".

VDLNAME Following a colon (:), any characters except comma (,) up to a terminating comma forms the VDL name.

12.1 Top-Level CVDL

The top level of CVDL is parsing of a file of VDLs:

```
file:
empty
file vdl
file <meta_list>
file <>
vdl:
:VDLNAME, top_or #
:VDLNAME, <meta_list> top_or #
:VDLNAME, <>> top_or #
meta_list:
STRING
!STRING
meta_list, STRING
```

The first rule says that a file of VDLs is either empty or is a file followed by a vdl or meta operator. When a VDL file is opened for processing, the user initially has nothing (i.e. the file is opened but has not been read yet), so the first line of the first rule is always matched immediately.

If the file is not completely empty, the next thing to expect to find is either a VDL or a meta operator such as a file type restriction list (see Chapter 11). Meta operators start with either a string or a negated (!) string, optionally followed by more strings separated by commas. VDLs start with colon (:), followed by the VDL name, comma, optional meta operator, then 'top or' (see below), then '#'.

Once the first VDL or meta list is read, that matches the file rule, which then continues to apply recursively. For example, if a file contained three VDLs, the file rule above would recursively match over processing time 0, ..., 3, as follows:

```
file(0) = empty
file(1) = file(0) vdl = empty vdl1 file(2) = file(1) vdl = empty vdl1 vdl2 file(3) =
file(2) vdl = empty vdl1 vdl2 vdl3
```

the syntax description continues with the top-level (i.e. high-level) CVDL operators: OR, XOR, AND, NOT, SIZE, and NAME (see Chapter 8):

```
top_or:
        top_xor
        top_or OR top_xor

top_xor:
        top_and
        top_xor XOR top_and

top_and:
        top_not
        top_and AND top_not

top_not:
cat
size
name
        NOT top_not
        ( top_or )

size    SIZE RELOP INTEGER
        INTEGER RELOP SIZE

RELOP:
        <       >       !=      ==      <=      >=

name:
NAME "~= cat
```

The list of operators above falls through to 'cat' which is described below. RELOP is the list of relational operators used in conjunction with the SIZE operator. Note that all CVDL operators are recognized case-insensitively, so 'AND' may be written as 'and', etc.

12.2 Low-Level CVDL

The basic low-level CVDL operators are described in Chapter 6, and the text and regex operators are described in Chapters 7 and 9:

```
cat:
        or
        cat , or
or:
        code
        or | code
code:
        data
        ( cat )
        @ range_expr, data
        @ range_expr, ( cat )
        .*, data
        .*, ( cat )
        ABS INTEGER , data
        ABS INTEGER , ( cat )
        %f > DOUBLE
        %f > - DOUBLE
        ~R...STRING
        ~R...STRING ( trig_and )
range_expr:
        INTEGER
        INTERGER -
        - INTEGER
        INTEGER - INTEGER
```

The list of operators above depends on 'data' and 'trig and' which are described below. As an example of parsing using the description above, 'a', 'b' | 'c', 'd' would result in the parse tree (('a' CAT ('b' | 'c')) CAT 'd') where 'CAT' represents concatenation (i.e. the comma operator).

Note that 'range expr', '.*', and 'ABS' may be applied to 'data' or '(cat)' corresponding to the discussion of offset groups in Section 6.8.

12.3 CVDL Data

CVDL 'data' is defined in terms of several data types including byte expressions, string data, set data, etc. as follows:

```
data:
        byte_expr
        string_data
        pseudo_data
        set_data
        ^ set_data
        W0
        W1
        WP0
        WP1
        WS0
        WS1
        EOD
        \d+
        repetition_expr
        fuzzy BYTE
        fuzzy STRING
fuzzy:
        FUZZY INTEGER
        FUZZY +- INTEGER
        FUZZY -+ INTEGER
        FUZZY - INTEGER
        FUZZY + INTEGER
        FUZZY - INTEGER + INTEGER
        FUZZY + INTEGER - INTEGER
repetition_expr:
        byte_expr [ range_expr ]
        string_data [ range_expr ]
        set_data [ range_expr ]
        ^ set_data [ range_expr ]
byte_expr:
        BYTE
        ^ BYTE
        byte_range
        ^ byte_range
byte_range:
        - BYTE
        BYTE -
        BYTE - BYTE

set_data:
        { set_list }
set_list:
        set_item
        set_list , set_item
set_item:
        STRING
        byte_expr
        set_data
        ^ set_data
string_data:
        STRING
        ~ STRING
pseudo_data:
        ~~ STRING
        ~W STRING
        ~# STRING
        ~# INTEGER STRING
```

12.4 Regex Trigger Data

As discussed in Section 9.2, regular expressions may include trigger data to improve run-time speed. The syntax for regex trigger data is:

```
trig_and:
        trig_cat
        trig_and AND trig_cat
trig_cat:
        trig_or
        trig_cat, trig_or
trig_or:
        STRING
        (trig_cat)
        trig_or | STRING
        trig_or | (trig_cat)
```

12.5 Exercises

Based on some pattern matching operation which is not currently included in CVDL, design a new CVDL operator, specify how it would fit in with the current syntax, and discuss how it could be implemented efficiently.

Chapter 13 Performance and Testing

The three main aspects of performance and testing are: accuracy, speed, and memory usage. Testing for accuracy is straightforward; it requires scanning many clean and infected files to check for false positives (hits on clean files) and false negatives (misses on infected files). If false hits occur, the pattern or VDL must be adjusted, using techniques such as analysis of entropy and serial correlation (Section 3.5), use of high-level AND (Section 8.1) to combine pattern features, use of file-type specific engines (Chapter 4), or use of meta VDLs (Chapter 11).

This chapter covers issues related to speed and memory usage in performance and testing. Sample runs are performed on a very old and slow Sun Microsystems 333 MHz Ultra 5 system running Solaris 7, using VFind-15.4.5 compiled with the cbayes engine included, and CyberSoft VDLs version 2005-06-06-15-53-58 (13509 VDLs). It is unlikely that users currently will have any machines this old or slow. Although absolute speed results depend on the system processor, most results presented here will reflect relative speed improvements which are directly applicable to any system.

Testing should be done on an unloaded system, not an active server, and timing results should generally show that real time used is approximately equal to the sum of user and system CPU time. For comparison and generalization of results, at least three runs should be performed for each test, and the results averaged. The results for each run should be consistent, i.e. approximately the same. In certain cases, generally involving scanning a large file or large number of files, the operating system will use RAM for I/O caching, and the first run of a test may take much longer than successive runs. In this case one can perform four runs and just use results from the last three.

13.1 Testing Basics

This section presents some basic techniques for performance testing, and examine VFind startup and run-time speed. VFind, like most application software, takes some time to start up and initialize internal data structures before it actually scans any files. The startup time can be measured by not specifying any input files on the command-line, and redirecting input from the null device:

```
% cd examples/perf
% timex VFind < /dev/null |> egrep '^(real|user|sys)'
real    2.86
user    2.18
sys     0.60
% timex VFind < /dev/null |> egrep '^(real|user|sys)'
real    2.86
user    2.16
sys     0.65
% timex VFind > /dev/null |> egrep '^(real|user|sys)'
real    2.85
user    2.09
sys     0.71
```

To calculate the average startup CPU time used, add together all of the user and sys times from above and divide by 3; the result is 2.80, which is consistent with the real times shown.

To check RAM usage, run VFind interactively, suspend it at the file name prompt, then run 'top':

```
% VFind
...
Enter the name of the file to be checked:^Z
% top
PID    USERNAME  THR  PRI  NICE  SIZE  RES  STATE  TIME CPU    COMMAND
20197  perry     1    50   0     50M   50M  stop   0:02 6.84%  VFind
```

So 'top' shows that 50 MB of RAM is used.

As an aid for subsequent examples, the '3.sh' script shown below will be used to run VFind three times and display the run times. Using 3.sh to recheck the above example, the user obtains just about the same average result of about 2.80 seconds CPU time:

```
% cat 3.sh
#! /bin/sh

(timex "$@" >/dev/null; timex "$@" >/dev/null; timex "$@" >/dev/null) 2>&1 |
nawk ' # note timex can report hours:minutes:seconds, e.g. 1:15:23.20
BEGIN { real = user = sys = r = u = s = n = 0; }
function tconv( t) { m=split(t,a,":"); x=1; y=0;
while( m > 0) { y += x*a[m--]; x *= 60; } return y; }
$1 == "real" { r = tconv($2); real += r; ++n; }
$1 == "user" { u = tconv($2); user += u; }
$1 == "sys" { s = tconv($2); sys += s;
printf( "real = %.2f, user = %.2f, sys = %.2f\n", r, u, s); }
END { printf( "avg real = %.2f, cpu = %.2f\n", real/n, (user + sys)/n);
}'
% ./3.sh VFind < /dev/null
real = 2.85, user = 2.04, sys = 0.74
real = 2.90, user = 2.12, sys = 0.67
real = 2.86, user = 2.19, sys = 0.61
avg real = 2.87, cpu = 2.79
```

The initial startup time shown above, before scanning any files, is actually only part of the total startup time and represents VFind reading VDL and other engine configuration files. A further step of initialization, to compile the patterns for fast/parallel search (see Section 13.2), is required before scanning the first file. VFind-mt, the multithreaded version of VFind, does perform this further step right at the beginning, as can be seen by comparing the following run to that above:

```
% VFind-mt < /dev/null |& grep Compil
##==> Compiling scan engines...
% ./3.sh VFind-mt < /dev/null
real = 14.39, user = 13.61, sys = 0.63
real = 14.28, user = 13.60, sys = 0.55
real = 14.19, user = 13.47, sys = 0.60
avg real = 14.29, cpu = 14.15
```

The non-multithreaded version of VFind defers the final step of initialization until just before scanning the first file, so at least one file has to be scanned to see that:

```
% cat hi hi
% ./3.sh VFind hi
real = 14.37, user = 13.38, sys = 0.74
real = 14.16, user = 13.20, sys = 0.75
real = 14.23, user = 13.30, sys = 0.65
avg real = 14.25, cpu = 14 01
% ./3.sh VFind hi hi
real = 14.27, user = 13.46, sys = 0.68
real = 14.12, user = 13.27, sys = 0.71
real = 13.96, user = 13.25, sys = 0.64
avg real = 14.12, cpu = 14.00
```

The total startup CPU time of about 14 seconds is consistent with the VFind-mt run above. Note that the second run above scanned the 'hi' file twice in about the same time as scanning it once. So for such a small file, the VFind cpu-time is practically all startup overhead.

For scanning a number of files, the run-time will be much less if VFind is invoked just once to scan all of the files, vs. invoking VFind separately for each file. For example, file 'rand.dat' contains 1,000,000 bytes of random data and is scanned once, then ten times:

```
% .3.sh VFind rand.dat
real = 15.56, user = 14.65, sys = 0.76
real = 15.50, user = 14.73, sys = 0.66
real = 15.61, user = 14.63, sys = 0.83
avg real = 15.56, cpu = 15.42
% ./3.sh VFind 'perl -e 'print "rand.dat " x 10;''
real = 28.96, user = 27.91, sys = 0.82
real = 29.07, user = 27.94, sys = 0.83
real = 29.19, user = 27.99, sys = 0.71
avg real = 29.07, cpu = 28.73
```

If VFind was invoked once for each of ten scans of rand.dat, the total CPU time would be 10*15.42 = 154.2 seconds. But as shown above, invoking VFind once to scan rand.dat ten times uses only 28.73 seconds. Considering the 14 second startup overhead, only 14.73 of the 28.73 seconds were used for actual scanning, yielding 1.473 seconds per scan of rand.dat.

13.2 Fast/Parallel Search

VFind uses a fast parallel search engine design, so scanning run-time is mostly independent of the number of VDL rules. A parallel search for all patterns is always performed first, and if that produces no matches then the patterns definitely do not appear in the data; however, if the parallel search does match a pattern, that pattern must be rechecked individually, in a slower serial search mode, since the parallel search does not handle options such as offsets and regular expressions.

Only a partial set of CVDL pattern types can be handled by the fast/parallel search. It does not handle case-insensitivity and wildcard white space, but those features are handled by the vdlc engine as discussed in Sections 4.2 and 13.3. All other pattern strings consisting of at least 4 characters, including offset operators, but not including sets, fuzzy, repetition, white space, or regex operators, are handled.

By using patterns which can be included in the parallel search, run-time can be lowered significantly. The following contrived example demonstrates this using "slow.vdl" and "fast.vdl" files containing 1000 copies of a VDL. The slow VDL file uses patterns containing less than 4 characters, which will not be included in the parallel search. The fast VDL file uses patterns containing 6 characters, which will be included in the parallel search:

```
% head -2 slow.vdl
:abc-efg, "abc", "efg" #
:abc-efg, "abc", "efg" #
% wc -l slow.vdl
1000 slow.vdl
% head -2 fast.vdl
:abc-efg, "abcefg" #
:abc-efg, "abcefg" #
% wc -l fast.vdl
1000 fast.vdl
```

Using "slow.vdl" the run-time is about 30 seconds, whereas using "fast.vdl" the run-time is less than 1 second:

```
% ./3.sh VFind --liboff='*' --vdl=slow.vdl --quiet=1 rand.dat
real = 29.58, user = 29.47, sys = 0.04
real = 30.04, user = 29.72, sys = 0.05
real = 29.85, user = 29.56, sys = 0.05
avg real = 29.82, cpu = 29.63
```

```
% ./3.sh VFind --liboff='*' --vdl=fast.vdl --quiet=1 rand.dat
real = 0.18, user = 0.15, sys = 0.00
real = 0.17, user = 0.11, sys = 0.04
real = 0.17, user = 0.13, sys = 0.03
avg real = 0.17, cpu = 0.15
```

In this example, using the slow VDLs, rand.dat was actually scanned 1000 times, once for each VDL. Using the fast VDLs, rand.dat was only scanned once. Note that there are two basic ways to test an individual VDL: running the VDL on a large number of files, or running a large number of copies of the VDL on a small number of files. The tests shown here use the second method, with 1000 copies of the VDL run on one data file. In production, one should not use multiple copies of the same VDL; that is only done for testing to make the run-time large enough to be measured accurately. For production VDLs, the VFind -d,-dup-check option can be used to check for duplicate VDL names and definitions (see Section 2.1).

As mentioned at the beginning of this section, when the parallel search engine detects a match, the VDL must be run in a slower serial mode, rescanning the data, to check for an exact match. If the data really does not match the pattern, then rescanning represents a run-time penalty which would be beneficial to avoid. To avoid excessive rescanning, it is important that the parallel search engine includes strings which will rarely not hit when rescanned.

The parallel search knows the difference between AND and concatenation, and will not trigger a rescan if concatenated matches occur in the wrong order. For example, take the last 32 bytes of rand.dat:

```
% od -tx1 rand.dat | tail -3
3641040 36 12 7c 73 dc 37 5d 86 9a bc 20 32 7b 4c 50 ec
3641060 12 e9 a9 ad 56 fa d2 98 25 a0 30 3c 8b 4b 39 f1
3641100
and consider VDLs which specify those bytes with the last 16 bytes first:
% cat s2.vdl
:vand,
"\x12\xe9\xa9\xad\x56\xfa\xd2\x98\x25\xa0\x30\x3c\x8b\x4b\x39\xf1" AND "\x36\x12\x7c\x73
\xdc\x37\x5d\x86\x9a\xbc\x20\x32\x7b\x4c\x50\xec" AND
size == 1234 #
% cat f2.vdl
:vcat, "\x12\xe9\xa9\xad\x56\xfa\xd2\x98\x25\xa0\x30\x3c\x8b\x4b\x39\xf1", "\x36\x12
\x7c\x73\xdc\x37\x5d\x86\x9a\xbc\x20\x32\x7b\x4c\x50\xec" AND
size == 1234 #
```

Each VDL has a "size" operator included simply to force it to not match. VDL "vand" in file s2.vdl uses AND for the two 16-byte patterns; that part of the VDL will match in the parallel search, causing the data to be rescanned. But VDL "vcat" in file f2.vdl uses concatenation (,) of the patterns, and since they are in the wrong order, they will not match in the parallel search, and the data will not be rescanned.

The performance of the VDLs above is tested using file "slow2.vdl" containing 1000 copies of VDL vand, and file "fast2.vdl" containing 1000 copies of VDL vcat:

```
% ./3.sh VFind --liboff='*' --vdl=slow2.vdl --quiet=1 rand.dat
real = 59.14, user = 58.91, sys = 0.06
real = 59.11, user = 58.92, sys = 0.05
real = 59.11, user = 58.89, sys = 0.07
avg real = 59.12, cpu = 58.97
% ./3.sh VFind --liboff='*' --vdl=fast2.vdl --quiet=1 rand.dat
real = 0.21, user = 0.13, sys = 0.06
real = 0.20, user = 0.18, sys = 0.02
real = 0.20, user = 0.15, sys = 0.05
avg real = 0.20, cpu = 0.20
```

Using "fast2.vdl" the run-time is almost zero, whereas using "slow2.vdl" the run-time is close to 1 minute, since the parallel search triggers 1000 rescans of the data.

In cases where triggering of rescans by the parallel search is unavoidable, one should at least try to make the rescan as fast as possible. One way to do that in patterns which use the AND or OR operators is to put the fastest operations first. CVDL AND and OR operators use short-circuit evaluation, similar to the logical && and || operators in the C programming language. That is, if the left side of an AND is false, the right side is not evaluated. And if the left side of an OR is true, the right side is not evaluated.

For example, VDL vand from "s2.vdl" above uses a size operator as the last part of the VDL. The size operator is fast since it just checks the file size and does not scan any data. Thus, the VDL should run faster in a rescan if the size operator is specified first. VDL "vand-b" in file s2b.vdl is the same as VDL vand except for listing the size operator first instead of last:

```
% cat s2b.vdl
:vand-b,
size == 1234
AND "\x12\xe9\xa9\xad\x56\xfa\xd2\x98\x25\xa0\x30\x3c\x8b\x4b\x39\xf1"
AND "\x36\x12\x7c\x73\xdc\x37\x5d\x86\x9a\xbc\x20\x32\x7b\x4c\x50\xec"#

File "slow2b.vdl" contains 1000 copies of VDL vand-b, and its performance is about the
same as "fast2.vdl":
% ./3.sh VFind --liboff='*' --vdl=slow2b.vdl --quiet=1 rand.dat
real = 0.21, user = 0.14, sys = 0.05
real = 0.20, user = 0.15, sys = 0.04
real = 0.20, user = 0.13, sys = 0.07
avg real = 0.20, cpu = 0.19
```

13.3 Case-Insensitive Fast/Parallel Search

As discussed in Section 4.2, although the VDL parallel search engine does not handle case-insensitivity and wildcard white space, a separate vdlc parallel search engine is provided to perform fast scanning for VDLs consisting mostly of those types of patterns.

To illustrate the performance gain using the vdlc engine, a set of case-insensitive word pattern VDLs were first created from the online /usr/dict/words dictionary:

```
% cat mkwords.sh
#!/bin/sh
#
# make case-insensitive word VDLs
# run < /usr/dict/words > words.vdl grep '^.............*' | # at least 12 letters sed
-e "s/.*/:&,~\"&\"#/"
% ./mkwords.sh < /usr/dict/words > words.vdl
% wc -l words.vdl
1064 words.vdl
% head -3 words.vdl
:abovementioned,~"abovementioned"#
:absentminded,~"absentminded"#
:accelerometer,~"accelerometer"#
% tail -3 words.vdl
:wholehearted,~"wholehearted"#
:Williamsburg,~"Williamsburg"#
:Winnipesaukee,~"Winnipesaukee"#
```

Now compare performance between the vdl and vdlc engines using these VDLs:

```
%./3.sh vfind -liboff='*' --vdl=words.vdl -quiet=1 rand.dat
real = 55.75, user = 55.57, sys = 0.04
real = 54.73, user = 54.56, sys = 0.05
real = 54.71, user = 54.52, sys = 0.06
avg real = 55.06, cpu = 54.93

% ./3.sh vfind --liboff='*' --vdlc=words.vdl --quiet=1 rand.dat
real = 0.33, user = 0.26, sys = 0.06
real = 0.32, user = 0.28, sys = 0.04
real = 0.32, user = 0.23, sys = 0.08
avg real = 0.32, cpu = 0.32
```

Using -vdl= to load the VDLs, none can be handled by the parallel search engine, so all are run in the slower serial mode, and the run-time is almost 1 minute. But using -vdlc= to load the VDLs, all are handled by the case-insensitive parallel search, so none are run in the slower serial mode, and the run-time is less than 1 second.

13.4 Exercises

1. Collect a set of 100 clean files of various types for use in performance testing. Include some Files of each of the following types: MS/EXE, OLE, text, and unknown (e.g. random or encrypted data). Put the file names in a '100.list' file. Run VFind with input redirected from 100.list and the -sst,-smart-scan-types option to check that it is correctly detecting the file types and not producing false hits on the collection.

2. Extract a set of 100 VDLs from the current CyberSoft VDL distribution exe, ole, scanall, other, and text VDL files. Put the set into a file '100.vdl', being careful to set the same file-type restrictions for each type of VDL in each section. Run VFind using only this set of VDLs on a collection of clean files to check for any problems.

3. Concatenate 100.list with itself to produce file '200.list'; concatenate that with itself to produce '400.list'; etc. up to '3200.list'. Similarly, use concatenation starting with 100.vdl to produce VDL files'200.vdl', '400.vdl', etc. up to '3200.vdl'. These files will be used in subsequent exercises for performance testing.

e 4. For each of the VDL files from the previous exercise, measure the performance of VFind for each of the set of list files. Produce tables and a plot of the results.

5. Assume that VFind scan time can be modeled as: i.t = tpNp + tfNf(13.1) where tp and tf are CPU time per pattern (VDL) and per file scanned respectively, Np is the number of patterns, and Nf is the number of files. Using results from the previous exercise, determine linear least-square-error estimates of tp and tf .

VFind CVDL Language

Lexical Analysis

By Barbara Lynn Higgins
With Foreword by: Peter V. Radatti
Edited by: Justin Honer

4 March 2019

CYBERSOFT OPERATING CORPORATION

1958 BUTLER PIKE, SUITE 100
CONSHOHOCKEN, PA. 19428 U.S.A.
Telephone:+1 610-825-4748?
Fax:+1 610/825-6785
Website: www.cybersoft.com
General Email: sales@cyber.com
Providing Security Tools Since 1988

WARNING:

This section of the manual contains language that some may find objectionable. This language is necessary as examples of how to detect this type of language.

REASONING:

This section of the manual is included not because CyberSoft believes that customers are interested in unsolicited bulk email analysis, but because it is a real world example of complex lexical analysis. This manual was developed by CyberSoft's real world work with customers reducing hostile email messages utilizing the Safe Internet Email software product. The VDLs were used not only to identify hostile messages but to populate a collection of hostile messages that VFind could then utilize with it's Bayes Engine. This provided the ability to stop messages both directly through pattern analysis and indirectly, statistically using Bayes.

Table of Contents

Foreword by Peter V. Radatti

June 2009

I had been involved in lexical analysis for many years when I designed the VFind tool and the CVDL language. From the beginning the CVDL language was designed to be both specific to detecting hostile software in binary and source forms and to detection of words or phases in human language. This is because I was involved in many projects where the goals were to detect "dirty words". In this context "dirty words" was not something that a lover might whisper to you but proscribed words that indicated that protected information was contained within the document. At that time I was concerned with both detecting documents that were not permitted to be exported outside of an organization either by accident or as unauthorized and deliberate export. The unauthorized export I assumed used various methods of hiding the fact that the document contained proscribed data. All of this was before the Internet.

When the Internet became popular and unsolicited bulk email (UBE) became popular there was a natural link between my interest in lexical analysis and UBE. The authors of UBE were trying with great success to hide the fact that their messages were UBE while still delivering the "payload" to the intended target in a human readable way. This was just what I had designed the CVDL language to do. CyberSoft started to experiment with detection of UBE along with virus content in the same VFind tool. Mixing the data had no effect on the tool and allowed us to accomplish more than one task at the same time.

After a couple of years of research CyberSoft concluded that not only was the VFind CVDL language good for detecting all forms of software attacks but was also a great "dirty word" detector, UBE detector and a fine all around general purpose pattern analysis program. Contained herein is a document written by CyberSoft's Senior Data/Virus Analyst at that time, Barbara Lynn Higgins, describing how she was able to use VFind's CVDL to detect UBE. It is very interesting to read and also gives you a greater idea of some of the other lexical problems that can be solved with these tools.

Thank you,
Peter V. Radatti
CyberSoft Operating Corporation
Chairman / Founder

Chapter One: The Basics

VDL Format

A "VDL" (short for "CVDL", or CyberSoft Virus Definition Language) is a signature taken from the source that is used by VFind to detect any matches containing the enclosed string of data. It is comprised of a name, the data string, and a terminus character, as seen below:

```
:SPAMVDL G 01111,"string data"#
```

The colon at the extreme left margin indicates the beginning of a new signature.

The name SPAMVDL is common across all spam vdlsets and is added by VFind once the signature is completed.

The letter following the name indicates in which vdlset the signature will reside. The vdlset names are as follows:

vdlset	Name	Description
B	Business	business services or products, domain hosting, e-marketing, etc.
C	Contest	lotteries, sweepstakes, online contests
E	Community	dating websites, non-pornographic socialization offers
R	Credit	credit cards, mortgages, loans, refinancing, refinancing, debt consolidation, etc.
L	Gambling	online poker, slots, casino, gambling, etc.
G	General	foreign language emails, misspelled/spammer words, etc.
H	Harassment	profanity, obscene/offensive language and descriptions, depraved offers, etc.
M	Medical	online pharmacies, discount drugs, weight loss offers, sexual tretments, etc.
F	Offers	free/discounted items and services, replica merchandise, financial deals, etc.
P	Pornography	adult sites, sexual services and merchandise, etc.
T	Technology	illegal software, discounted hardware/software, cable/satellite services/equipment, etc.
U	Unsubscribe	unique unsubscribe language or websites
X	Scams	phishing for personal information, overseas money gambits, etc.

The number after the vdlset letter is sequential from the last VDL number in the associated vdlset, which in this case would be 01110, and will be added by VFind once the signature is completed.

The comma separates the name of the VDL from the actual data taken from the spam itself. The only part of the VDL that is displayed in SIE is the name. If the comma is absent, the entire VDL will be displayed. The comma also serves to separate one or more data strings from each other and from macros and functions.

The quotes enclose the data taken from the spam email. This is the information VFind needs to find the data indicating an email qualifies as spam. There can be several data strings in a single VDL as long as other conditions of the CVDL language are met.

The hash mark at the end indicates to VFind that it should stop scanning for data and report any spam detections caused by that VDL.

CVDL Operators Used to Create Lexical VDLs

Apart from the operators introduced above, there are several other commonly used operators that have very specific meanings within the VDL context.

~ (tilde) : used in front of a data string, it indicates case-insensitivity.

~~ (double tilde) : used in front of a data string, it indicates case, punctuation, return, and whitespace insensitivity. Usually used for phone numbers or wrapped text.

$macro : used in front of a word to indicate a macro, or pre-defined action.

,.*, (comma-dot) : skips over data in a line until the next data string is found. Only used on a star-comma single line - does not wrap.

@-xxdata (jump) : used between data strings to indicate maximum distance to jump until next data string can be located. Not influenced by any characters or formatting.

( ) (exclusive) : used around a data string to indicate that the string should be read as a unit. Often used with all NOT, OR , or AND functions

| (low level or) : used between data strings to indicate "this or that". Cannot be used to indicate "this and/or that". Data must be positioned sequentially in the VDL as they would appear in the source.

& (low level and) : used between data strings to indicate "this and that". Data string must be positioned sequentially in the VDL as they would appear in the source.

OR (high level or) : used between data strings to indicate "this or that". Data strings do not need to be positioned sequentially-they may be positioned in the source either before or after other data strings in the VDL.

AND (high level and) : used between data strings to indicate "this and that". Data strings do not need to be positioned sequentially-they may be positioned in the source either before or after other data strings in the VDL.

NOT (not) : used with data strings to indicate that a VDL may detect a string of data in certain files but not in others, or may indicate a particular word spelling which is not to be detected. Used with AND.

{" "} (set1) : indicates a set of data which is to be scanned together.

{'X'-'Y'} (set2) : indicates a range of data which is to be scanned at one time.

< > (limitations) : limits range where VFind can scan.

[ ] (range) : indicates number of times a data string or macro should be repeated.

W1 (whitespace) : indicates one or more whitespaces between or after data strings.

WP1 (whitespace&punct) : indicates one or more whitespaces or punctuation marks between or after data strings.

Macros

Macros are a "shortcut" for VFind that enables it to load a given string of information once and then apply it over and over again without having to reload. Several macros used in the creation of spam VDLs are used frequently enough to merit their inclusion here.

Macro	Name	Description
$DOM1,	domain	stands in for "http://","https://", "@"(as used in emails), and ".", as used in domain names.
$DOM3,	domain3	stands in for ". com",". org","net", and other domain enders, including countries (".tw",".uk", etc.)
$website	website	stands in for $DOM1,jump, $DOM3, whitespace or punctuation
$subject,	subject line	indicated the subject line of an email with its attendant punctuation.
$punct	punctuation/whitespace	stands in for all punctuation marks and white spaces. Usually used with range operator
$nspo	punctuation	stands in for all punctuation marks only-means "no spaces, punctuation only"
$d	numbers	represents all single-digit numerical characters. Frequently used with range operator for longer numerical sequences.
$a	letters	represents all single upper and lower case text letter characters. Frequently used with range operator for longer strings
$r	letters & numbers	represents all single upper and lower case text letter characters and all single-digit numerical characters
$letter	letter groups	represents a set of upper and lower case letter characters belonging to a single letter group
$punc	letter	represents a set of non-text characters used to imitate certain letters.

Chapter Two: Creating Simple VDLS: URLs, Email, and Headers

Using Urls

The easiest way to detect spam is to use the url usually supplied by the spammer. The term "usually" is used because one of the spammer's tricks is to hide a url in an image or to obscure it in some other fashion. Here are some examples of how to write a VDL using a url.

It is not necessary to use the entire url. Indeed, the urls can, and do, change slightly each time a new wave of spam is sent out. As can be seen in the examples below, the first part of the url is in constant flux.

```
   <TD><A href=3D"http://Cotto.hundertsoft.com/?Barney"><SPAN=20
   <TD><A href=3D"http://Grywacheski.hundertsoft.com/?Hlapkovsky"><SPAN =
   <TD><A href=3D"http://Bledsoe.hundertsoft.com/?Rey"><SPAN =
   <TD><A href=3D"http://Aguilera.hundertsoft.com/?Besseling"><SPAN=20
   <TD><A href=3D"http://Mihajlovic.hundertsoft.com/?Tunio"><SPAN=20
   <TD><A href=3D"http://Zhao.hundertsoft.com/?Jatupapas"><SPAN =
```

The first section after "http://" is randomly generated each time a new email is sent out. It can be composed of a word, a string of random numbers and/or letters, or can even be another website, complete with .com, .net, etc. Ignore this section. Use only the last word/characterset that ends with the domain suffix. The trailing "/" may be included to indicate that this url is part of a larger url and is not a standalone, as it could be on legitimate email correspondence.

Therefore, the VDL could look like this:

```
          T,$DOM1,~"hundertsoft.com/"#
   OR     T,$DOM1,~"hundertsoft.com"#
   OR     T,$DOM1,~"hundertsoft.com",WP1#
```

where the macro $DOM1 stands in for the "http://" segment, because otherwise VFind would read the colon as the beginning of another VDL instead of a component in the url. Using $DOM1 before the domain name enables the detection of the url without any "http://", since it also looks for the period which immediately precedes the url ("www.Zhao.hundertsoft.com/). If the trailing forward slash is also eliminated, $DOM1 will also detect the domain name in an email address ("info@hundertsoft.com").

Another approach is to look for a key word in the url, especially if it is misspelled. Note the use of the same-line jump to avoid unnecessary data in the VDL. The $website macro is used instead of $DOM1 because the domain name is not unique to this email.

In the case of:

```
   http://uk.geocities.com/julpetismee/?s3=Np.Largest.pharmancy,reduced_prices
```

the VDL could look like this:

```
          M,$website,.*,~"pharmancy"#
   OR     M,$website,@-15,"/?",@-15,~"pharmancy"#
```

The second version would be used if the url wraps around a line, si nce the ".*," feature only works on the same line. It could also be used if there is an intervening string of data in the VDL. The "dot-star" feature is more costly for VFind to use than a simple limited jump because it utilizes more resources and can slow down scan times if overused.

Yet another approach is to use a key name or phrase that accompanies the domain name. In this url, the number salad at the end will change with each sending, so don't use it.

The VDL for the URL:

```
   http://uk.geocities.com/f_w_Lancaster/?q=G2P5lf8iYc45LSb72
```

could look like this:

```
M,$website,~"f_w_Lancaster/?"#
```

Note the use in all the preceding VDLs of the "tilde" function in front of the data string. This enables VFind to detect the data string whether it is typed in upper or lower case characters. Also note that when using the "$website" macro, the preceding punctuation of the data string is omitted because it is already a delineating feature at the end of this macro. Also, the "/?" feature is interesting to include since this is not often found in a url and serves to set this VDL apart from others.

Sometimes a url can be found only within the HTML portion of the email. When analyzing an email in a text editor, the body of the email will be in the top half and the HTML will be in the bottom half. Here is an example of a url which has been hidden in the HTML of the email. The "text" part of the email was a collection of quotes, book passages, and whatever else the spammer felt like throwing in there to foil the antispam watchdogs.

The VDL for the URL:

```
<A href=3D"http://www.coherence.fallurpos.com">
```

could look like this:

```
M,$DOM1,~"fallurpos.com"#
```

When looking through HTML for the url, don't confuse a url which is listed simply because it is related to a graphic. Check all urls in this section to make sure that the url to be used is for the spammer's website. Some graphics are taken from graphics-only websites for use in the email, so they may end up being blocked if their domain name in the VDL. Here is an example of a graphic-bearing spam email:

```
<img src=3D"http://www.colcha_retalhos.blogger.com.br/casal%20praia.jpg"> border
versus
<a target=3D"_blank" href=3D"http://www.ch1.com.br/apaixonante/feminino/">
```

In this next example, the urls were the same since the spammer website was hosting the graphic, but this is not always the case.

In this example:

```
<a target=3D"_blank" href=3D"http://www.ecobusiness.edu/mailbox>
<img src="http://www.ecobusiness.edu/mailing/pics/spacer.gif"
width="15"height="20"></font></td><tdheight="1" bgcolor="#FFFFFF" width="547">
```

The VDL could look like:

```
B,$DOM1,~"ecobusiness.Edu/"#
```

Some domain names change with each new wave. Fortunately, CVDL can handle this with relative ease. As long as there is a common string in the domain name, the fluctuating string can be "finessed" with the use of a macro or two. As can be seen from the two examples, the base domain name string is the same in both urls, but the prefixes change.

Example:

```
<a href="http://www.tqrefi.net/?id=p11
```

Corresponding VDL:

```
R,$DOM1,$lclet[2-3],~"refi.net/"#
```

Example:

```
<a href="http://www.f18refi.net/?id=y86"
```

Corresponding VDL:

```
R,$DOM1,$r[2-3],~"refi.net/"#
```

In the first example, the macro for "lower-case letters" was used, and VFind was instructed to allow for 2 to 3 lower case letters preceding "refi.net". This provides some flexibility in detection, in case the spammer decides to add another letter at a later time. This VDL will still detect it.

In the second example, the macro is even more flexible than the one in the first VDL. In this case, the macro used is for "random upper- and lower-case letters and numbers 0-9". Therefore, any combination of three upper-case or lower-case letters and numbers can be inserted into this domain name and the VDL will still detect it.

Sometimes it isn't the domain name itself that is important but, rather, the data strings that follow it. Oftentimes the domain name will change with each spam wave, but the directional strings after it will remain the same.

```
From:
href="http://paypal.com.login-user2992.info/webscr.php?cmd=LogIn">https:
//www.paypal.com/cgi-bin/webscr.php?cmd=LogIn</A>

To: href="http://paypal.com.magicman.info/webscr.php?cmd=LogIn">https:
//www.paypal.com/cgi-bin/webscr.php?cmd=LogIn</A>
```

So the VDL may come out looking like this:

```
X,$DOM1,~"paypal.com ",@-15,~".info/webscr.php?cmd=LogIn"#
```

Be careful to make a VDL out of the bogus url, so pay attention to the redirects. In this case, the bogus url is first and points to the second, possibly legitimate url which is being displayed. Remember, if there are two urls with a carrot (">" or "<") between them, use the url on the open side of the carrot, or just look for "a href=", which is the HTML code that hides the real url.

Here are good examples of picking a data string carefully. This url contains two domains, but only one is the spammer's. If the trail is followed, it can be seen that the "rds.yahoo.com" would not be optimal for the purpose of creating a VDL, and the "dwo.com" is a red herring, but the "grang3.net" is prime material.

```
Go <A HREF="http://rds.yahoo.com/S=7100390/K=computer/v=7/SID=j/l=WS1/R=1/
SS=87757020/IPC=us/SHE=0/H=0/SIG=8321evjJN604/EXP=039612748/*-
http://dwo.com.grang3.net/ finish.asp">HERE</A> to make that change. <P><P>
```

As such, the VDL should look like this:

```
T,$DOM1,~"grang3.net"#
```

Other parts of this structure can be used for urls as well, just in case the spammer url changes in some way but the rest of the structure is identical from email to email.

```
Go <A HREF="http://rds.yahoo.com/S=7100390/K=computer/v=7/SID=j/l=WS1/R=1/
SS=87757020/IPC=us/SHE=0/H=0/SIG=8321evjJN604/EXP=039612748/*-
http://dwo.com.grang3.net/ finish.asp">HERE</A> to make that change. <P><P>
```

```
T,~"Go <A HREF=",$DOM1,~"yahoo.com/",@-20,~"computer/",@-200,$website,
@-100,~">HERE</A> to make that change.<"#
```

Please note that anytime a quotation mark is inside a url string, it will have to be "jumped" or escaped, or VFind will think that it signals the end of the string, and it will get a parse error report when it tries to post. Again, more discussion of this later.

Sometimes the url redirect isn't as obvious as it is in the last example. Here is a sample of a redirect that is coverted and could be easily missed,

```
href=3D"http://www.norada.ch/shop/catalog/redirect.php?action=3Durl&amp;g=oto=3D
www.norada.ch/shop/catalog/redirect.php%3faction=3Durl%26goto=3Dnexte=rmest
%252ecom">see =
```

Here there are two redirects, the first two being the same url and the third being the "real" url that should be used for the VDL. So the VDL could be written like this:

```
        F,$DOM1,~"norada.ch/",@-100,~~"nextermest","\x25\x2e","com"#
OR      F,$DOM1,~"norada.ch/",@-100,~~"nextermest","%.com"#
OR      F,$DOM1,@-100,~~"nextermest","\x25\x2e","com"#
OR      F,$DOM1,@-100,~~"nextermest","%.com"#
OR      F,"goto","\x3D",~~"nextermest","\x25\x2e","com"#
OR      F,"goto","\x3D",~~"nextermest", "%.com"#
```

Since the domain name "nextermest%252ecom" is line-wrapped, the extra spacing has to be allowed for by using the double-tilde function, but only on the part of the domain name which wraps because the rest of the name contains hex code for "%." If the double-tilde function is used on the entire name, it would ignore the "%." in the name because it is punctuation insensitive. Therefore, it is best to split off the wrapped section from the rest. As for the "%.", either the CVDL hex for the characters can be used, or if possible, turn them into text characters. VFind will treat both forms the same. However, the hex code should be retained if the characters are non-text or non-printing characters. Some urls don't exist as domain names but as IP addresses. These can be commonly found in scam spam, especially the infamous "eBay" and "PayPal" scams. In this case, the job is easy; just pick out the IP address and any attendant text that is relevant.

```
<a href="http://220.130.129.235/excite.html"> G, $DOM1,"220.130.129.235"#
OR      F,"goto","\x3D",~~"nextermest", "%.com"#
```

Using Email and IP Addresses in Headers

Before attacking the use of email addresses in VDLs, it is best to introduce email headers, since this is where many of them lurk and it must be known which ones are reliable and which ones are nonsense. Since this manual has already touched upon using IP addresses in VDLs, this is an opportune moment to show the various ups and downs of spam email headers.

This is the header of a spam email as received from the original spammer. All redirects, i.e. "received: from" lines, should be read from bottom to top, the top being the most recent redirect to the recipient. The farther down the list, the closer to the spammer.

```
<a href="http://220.130.129.235/excite.html"> G, $DOM1,"220.130.129.235"#Received:
(from uucp@localhost)
by egate.cyber.com (8.11.7/8.11.7) id j6M8YAG06381
for meg@egate.internal.cyber.com; Fri, 22 Jul 2005 04:34:10 -0400 (EDT)
>Received: from mh1.safeinternetemail.com (mh1.safeinternetemail.com
[207.8.185.17])
by mhub1.cyber.com (8.11.4/8.11.4) with ESMTP id j6M9fin12087
for <meg@egate.internal.cyber.com>; Fri, 22 Jul 2005 05:41:44 -0400 (EDT) X-anti-
virus-Scanned: anti-virus scanned by SIE - http://safeinternetemail.com/
Received: from p6018-ipad01niho.hiroshima.ocn.ne.jp (p6018-
ipad01niho.Hiroshima.ocn.ne.jp [61.214.95.18])
by mh1.safeinternetemail.com (8.13.1p1/8.13.1) with SMTP id Aj6M8WZcr026932
for <meg@cyber.com>; Fri, 22 Jul 2005 04:32:38 -0400
Received: from [159.33.141.98] (port=4578 helo=[Valparaiso])
by p6018-ipad01niho.hiroshima.ocn.ne.jp with esmtp
id 9900146789wringer7142
(Authid: EveretteDumas)
for meg@cyber.com; Fri, 22 Jul 2005 17:32:38 +0900
Message-ID: <4.30.5904061749510.15353-100000@hiroshimamonamour.net>
To: meg@cybersoft.com
Date: Thu, 25 Aug 2005 15:55:39 -0100
From: "Mauro Egan" <scnujtokisic@kewl.com>
Subject: Get It NOW!!!!!
```

It can be seen that, by following the path downward, that "159.33.141.98" is the spammer's machine or the one they are using to obscure their own identity. The IP address is in brackets because it is a reverse DNS lookup that was done by the first email intermediary. Note that there is no domain name with which to identify the spammer, but IP address may be used in the VDL if it clears a reverse DNS lookup; that is, that it is not part of a legitimate network somewhere. This spammer could have used a forged or "spoofed" address to cover their trail. When in doubt, follow the time stamps back in time to the earliest point and do a lookup on that IP.

One should always pay attention to the time stamps as these can give information about the spammer's route. As can be seen in the example below, the time stamps are radically different between the original sender and the recipient's server, but are still in keeping with the changing time zones (in italics). Since there are three hours between these time zones and the difference between the time stamps is within 20 minutes of 3 hours, the times are probably not accurate. However, this is no guarantee that the header information is not bogus.

```
Received: from 61.111.231.34 ([61.111.231.34])
by mh1.safeinternetemail.com (8.13.1p1/8.13.1) with SMTP id Aj7LJEvpr022867
for <megan@cyber.com>; Sun, 21 Aug 2005 15:14:59 -0400<---------
Received: from 80.160.32.236 by ; Sun, 21 Aug 2005 12:09:41 -0700<--------
```

Now here is the header of a spam email that has been spoofed. As the redirects are followed backward in time, it can be seen that the redirects do not flow properly.

Any break in the continuity of the redirects is a dead giveaway that the domain names have been spoofed. However, the IP address next to "25-A" is a reverse DNS lookup, which is accurate as far as it goes, but may not belong to the spammer. Also, the "helo" is not a valid address and is therefore useless for the purpose of creating a VDL, as are identifiers like "unknown". One could try using something like "(HELO Jonnie)", since it is a little more specific, but make sure to keep it case sensitive.

```
Received: (from uucp@localhost)
by egate.cyber.com (8.11.7/8.11.7) id j7U2uAJ23997
for meg@egate.internal.cyber.com; Mon, 29 Aug 2005 22:56:10 -0400 (EDT)
>Received: from mh1.safeinternetemail.com (mh1.safeinternetemail.com
[207.8.185.17])
by mhub1.cyber.com (8.11.4/8.11.4) with ESMTP id j7U2tnO07193
for <meg@egate.internal.cyber.com>; Mon, 29 Aug 2005 22:55:49 -0400 (EDT)
Received: from 25-A ([211.168.37.35])
by mh1.safeinternetemail.com (8.13.1p1/8.13.1) with SMTP id Aj7U2s6Qr031215
for <meg@cyber.com>; Mon, 29 Aug 2005 22:54:08 -0400
Received: from (helo=abstention)
by abide.dattaweb.com with asmtp (Exim 4.99)
id 1Cn4Tf-0789Vk-43
for Mccabe6Bradley@masterfox.com.sg; Mon, 29 Aug 2005 21:52:04 -0600
Date: Mon, 29 Aug 2005 19:54:04 -0800
```

This email also contains a false identity, a sure sign of a spammer. This may be used in a pinch, since sometimes these false identities are used again, but usually this is a very short-term fix. Instead of the "(helo=abstention)" identity, some spammers will use a common domain name to hide their identities, such as:

```
Received: from excite.com ([221.139.219.175])
```

Again, here the IP address in brackets is correct, but the spammer probably has nothing to do with excite.com, since the domain is missing any prefixes that might identify an individual, such as "spammonger.excite.com". Use the IP address only if it checks out as belonging to a spammer, not just to "excite.com" or another innocent party.

Sometimes an obviously false address may be used to detect spam. If a header is examined something to this effect can be seen:

```
Received: from autogamous (192.168.134.14) by gevoel.net (Femoral se 1.12) with
SMTP id
NgpgFY-WVWLMt-nS for ; Wed, 19 Oct 2005 02:38:37 -0500
```

This could actually be used to create a VDL. For one thing, it doesn't implicate any innocent computers by using a ".com" or ".net", etc. For another, certain spammers use certain "names" like this on different emails. Time to catch several birds with one stone, as they say. So, the VDL would look something like this:

```
    M,$from,~"autogamous ("#
```

where the macro "$recd" covers "(newline) Received: from ", the word "autogamous" can be made case- insensitive to cover all bases, and the requisite space and "opening parenthesis" of the accompanying IP address (which is omitted because it is probably faked) to eliminate the presence of other words following "autogamous", such as ".com", ".net", etc.

When an email has been forwarded from the original recipient to another recipient, the long headers at the top will only reflect the forwarding address and intermediate servers, not the point of origin of the original spam document. Moving past these initial headers and looking for a secondary or, in some cases, even a tertiary set of headers below them will be necessary. Here is a wonderful example of the perils of forwarding to the lexical analyst:

```
From tycobb@aurelius.com Tue Aug 30 08:03:40 2005
Return-Path: <tycobb@aurelius.com >
Received: (from uucp@localhost)
by egate.cyber.com (8.11.7/8.11.7) id j7UC5Ar00409
for support@egate.internal.cyber.com; Tue, 30 Aug 2005 08:05:10 -0400 (EDT)
>Received: from mh1.safeinternetemail.com (mh1.safeinternetemail.com
[217.8.105.17])
by mhub1.cyber.com (8.11.4/8.11.4) with ESMTP id j7UC4rO08171
for <support@egate.internal.cyber.com>; Tue, 30 Aug 2005 08:04:53 -0400 (EDT)
X-anti-virus-Scanned: anti-virus scanned by SIE - http://safeinternetemail.com/
Received: from allmaster.aurelius.com (allmaster.aurelius.com [218.34.129.18])
by mh1.safeinternetemail.com (8.13.1p1/8.13.1) with ESMTP id Aj7UC3fkC023476
for <jobs@cyber.com>; Tue, 30 Aug 2005 08:03:42 -0400
Received: from babylon.aurelius.com (caesar.aurelius.com [218.34.129.68])
by allmaster.aurelius.com (8.12.1/8.12.1) with ESMTP id
Subject: [Fwd: FW: Greets! I offer you VIAGGRRAAA!!! From: "Ty Cobb"
<tycobb@aurelius.com>
To: jobs@cyber.com
Organization: Aurelius Realty
Date: Tue, 30 Aug 2005 07:03:40 -0500
Message-Id: <1125403420.7865.36.nile@casear.aurelius.com>
Mime-Version: 1.0
X-Mailer: Evolution 2.0.4-3mdk
Content-Type: multipart/mixed;
boundary="=-c3o6LifOhWQmjK67Ec"

End of most recent forward header

--=-c3o6LifOhWQmjK67EcbX
Content-Type: text/plain
Content-Transfer-Encoding: 7bit

--=-c3o6LifOhWQmjK67EcbX
Content-Disposition: inline
Content-Description: Forwarded message - FW: Greets! I offer you VIAGGRRAAA!!!
Content-Type: message/rfc822

Return-Path: <duncanmcleod@aurelius.com>
X-Sieve: cmu-sieve 2.0
Received: from ceotpc (cpe-166-68-220-42.austin.res.rr.com [166.68.220.42])
(authenticated bits=0)
by allmaster.aurelius.com (8.12.1/8.12.1) with ESMTP id j7UBQ2vr017472
for <tycobb@aurelius.com>; Tue, 30 Aug 2005 06:26:06 -0500
Message-Id: <200508301126.j7UBQ2vr017472@allmaster.aurelius.com>
From: "Duncan McLeod"<duncanmcleod@aurelius.com>
To: "'Ty Cobb'" <tycobb@aurelius.com>
Date: Tue, 30 Aug 2005 06:26:04 -0500
MIME-Version: 1.0
Content-Type: text/plain; charset="iso-8859-1"
Get rid of this garbage, please!
Thanks – Dunc

End of original forward headers

-----Original Message-----
From: copyright@yahoo-inc.com [mailto:copyright@yahoo-inc.com]
Sent: Tuesday, August 30, 2005 2:11 AM
To: pr@aurelius.com
Subject: Greets! I offer you VIAGGRRAAA!!!

Hi, Duncie! How's it hangin"?;)
Does your baby wanna play all day?;P
```

As the above example shows, it can be rather confusing to find the relevant information amidst all this information. When the original spam email is finally found, all the information needed about the mail redirects are usually stripped away by the forwarding process. Therefore, other information available will have to be used, limited as it is.

First, a VDL can be created for the email address. It is certainly not a valid one for yahoo.

Example:

```
From: copyright@yahoo-inc.com [mailto:copyright@yahoo-inc.com]
```

Corresponding VDL:

```
M,$from,.*,~"copyright@yahoo-inc.com"#
```

Second, a VDL can be created for the elements of the subject line.

Example:

```
Subject: Greets! I offer you VIAGGRRAAA!!!
```

Corresponding VDL:

```
        M,$subject,.*,"Greets!"#
OR      M,~"Viaggrraaa!!!"#
OR      M,NOT ~"viagra" AND ({"Vv"}[1-3],{"Ii"}[1-3]
        {"Aa"}[1-3],{"Gg"}[1-3],{"Rr"}[1-3],{"Aa"}
        [1-3]),WP1#
        M,~"Hi,",@-15,"! How's it hangin"?;)",@-3,
        ~"Does your baby wanna play all day? ;D"
```

Based on unusual use of the greeting, based on unusual spelling of the drug, using the exclusion feature and repeating letters to catch all spelling variants, OR using unusual phrasing & emoticons to catch repeat emails with different websites.

The actual creation of VDLs based on lexical data will be addressed later on in this manual. A new feature has been added by some ISPs to try to identify spammers. It's an optional feature (anything in an email header which is preceded by an "X" is an option and may be subject to outside manipulation) that identifies the IP of origin of the email.

```
From rzitgiskrs@rocketmail.com Sat Jul 23 16:41:09 2005
Return-Path: rzitgiskrs@rocketmail.com
Received: (from uucp@localhost)
by egate.cyber.com (8.11.7/8.11.7) id j6NJiAk07244
for megan@egate.internal.cyber.com; Sat, 23 Jul 2005 15:44:10 -0400 (EDT)
>Received: from mh1.safeinternetemail.com (mh1.safeinternetemail.com
[217.8.185.27])
by mhub1.cyber.com (8.11.4/8.11.4) with ESMTP id j6NKq0n16875
for <megan@egate.internal.cyber.com>; Sat, 23 Jul 2005 16:52:01 -0400 (EDT)
X-anti-virus-Scanned: anti-virus scanned by SIE - http://safeinternetemail.com/
Received: from 207.8.185.17 ([222.137.134.238])
by mh1.safeinternetemail.com (8.13.1p1/8.13.1) with SMTP id Aj6NJfrNQ010050
for <megan@cyber.com>; Sat, 23 Jul 2005 15:42:19 -0400
X-Apparently-To: megan@cyber.com via 207.8.185.17; Sat, 23 Jul 2005 21:37:09 +0100
X-Originating-IP: [222.137.134.238]
Received: from 222.137.134.238 (HELO 207.8.185.17) (222.137.134.238) by
mta285.mail.scd.yahoo.com with SMTP; Sat, 23 Jul 2005 17:38:09 -0300
MIME-Version: 1.0
Message-ID: <92224564890.1225010.13793.z17@mcimail.com> Delivered-To:
megan@cyber.com
Date: Sat, 23 Jul 2005 17:41:09 -0300
From: rzitgiskrs@rocketmail.com
To: megan@cyber.com
Subject: Presenting Home Loans made simple
Reply-to: rzitgiskrs@rocketmail.com
Content-Transfer-Encoding: 7Bit
```

Note how all the IP addresses agree with each other (except, of course, where the spammer tried to hide his identity). It must be kept in mind however, that the "originating IP" is the first known computer to send out the spam. The spammer is hiding behind it, preserving their anonymity. If this data is to be used in the VDL, an IP lookup must first be done. Sites like http://ws.arin.net/whois/?queryinput and http://www.zoneedit.com/lookup.html can help to identify the site of origin of the spam as far as it can be followed by IP listings.

Now it is time to look at the email portion of CyberSoft's analysis. The example below illustrates a salient point in the use of email addresses as VDLs, which is that not all email addresses can be trusted. The address "rztgskrs42@ rocketmail.com" is obviously false, created by a randomization process which changes the email address with each spam wave. If nothing else can be found to use in an email, such as the "test" email with no subject and no body, use this, but try to see if there is a pattern among any repetitions of this email for future vdls.

Example:

```
Delivered-To: megan@cyber.com
Date: Sat, 23 Jul 2005 17:41:09 -0300
From: rztgskrs42@rocketmail.com
To: megan@cyber.com
Subject: Presenting Home Loans made simple
Reply-to:kdjlwst42@rocketmail.com
Content-Transfer-Encoding: 7Bit Delivered-To: megan@cyber.com
Date: Sat, 27 Aug 2005 17:41:09 -0300
From: rztgskrs45@rocketmail.com
To: megan@cyber.com
Subject: Home Loans made simple and easy
Reply-to: kdjlwst45@rocketmail.com
Content-Transfer-Encoding: 7Bit
```

Corresponding VDL:

```
        R,$from,.*,~"rztgskrs",$d[2-3],~"@rocketmail.com"#
OR      R,"Reply-to,.*,~"kdjlwst",
        $d[2-3],~"@rocketmail.com"#
OR      R,$subject,.*,~"Home Loans made simple"#
```

As can be seen in the above example, the "From:" and "Reply-to:" lines don't always match. This is especially common in scam spam, like the Nigerian Scam. However, in scam spam, the two addresses are usually masquerading as legitimate email addresses used by real people, so both of these addresses can be taken and used to create VDLs. That way, if one of them is reused at some point in the future, the email will still be detected.

If the Return-Path address matches the sender's domain name, which matches the "From:" address, this doesn't necessarily mean that the email is from this place or this person.

```
Return-Path: <SylviBragg@furniturebrands.com>
Received: (from uucp@localhost)
by egate.cyber.com (8.11.7/8.11.7) id j78B0Ai20796
for doodah @egate.internal.cyber.com; Mon, 8 Aug 2005 07:00:10 -0400 (EDT)
>Received: from mh1.safeinternetemail.com (mh1.safeinternetemail.com
[207.8.185.17])
by mhub1.cyber.com (8.11.4/8.11.4) with ESMTP id j78C81n02775
for < doodah @egate.internal.cyber.com>; Mon, 8 Aug 2005 08:08:01 -0400(EDT)
X-anti-virus-Scanned: anti-virus scanned by SIE - http://safeinternetemail.com/
Received: from furniturebrands.com (61-216-76-163.dynamic.hinet.net
[61.216.76.163])
by mh1.safeinternetemail.com (8.13.1p1/8.13.1) with SMTP id Aj78AwPih012841
for < doodah @cyber.com>; Mon, 8 Aug 2005 06:58:27 -0400
Message-Id: <200508081058.Aj78AwPih012841@mh1.safeinternetemail.com>
From: "Sylvi Bragg" <SylviBragg@furniturebrands.com>
To: "Yaw Harbin" <doodah@cyber.com>
Subject: =?iso-8859-1?Q?Cool_S=C1VING?=
Date: Mon, 8 Aug 2005 05:58:22 -0500
```

Since the server has given the correct IP address in the originating position (due to DNS lookup), a trace back can be conducted to find out who the real culprit is and block the IP address on principle. If it really is Sylvi Braggs, then the course of action is clear--use the address in the VDL.

Some spammers avoid using names of real or fictional people and use the same "pseudo-identity" over and over again.

```
Received: from ensim (plct.ramifis.trojan.com)
Received: from ensim (tojo.georgiesguy.jerkadoodle.net)
```

Ignore the domain name in this case. They are almost certainly spoofed and are changed regularly. However, the lazy spammer has kept their "trademark" sender signature, "ensim".

```
G,$recd,"ensim ("
```

where $recd is a macro representing "Received","\x3a\x20","from","\x20", "ensim" is case-specific, and the space and opening parenthesis signify that no other characters follow "ensim", thus avoiding false detections on an identity which contains the word "ensim".

When dealing with email addresses that also contain numbers, it is not necessary to use the "case- insensitive" (tilde) feature unless it is also desired to find the "raw" IP address, sans decimals, as well as the IP or email addresses. An address like this one may carry the IP address before the "@" and the name of the last sending computer after.

In this case, the "case-,spacing-, and punctuation-insensitive" (double-tilde) feature can be used.

Example:

```
Received: from 222137134238@rottoncowboy
```

Corresponding VDL:

```
R,~~"222.137.134.238"#
```

This feature also works with phone numbers where they could be written differently in each email. More on that later. Just remember that double-tildes are expensive in resources and may slow down the scan.

Using Email Addresses in the Body

Not all email addresses are found in the headers, of course. Quite a few can be found in the body of the email. For example, this spammer has very nicely provided the domain name of their client and a choice of ways to use it.

Example:

```
<a href="mailto:gbh@ecobusiness.edu">info@ecobusiness.edu</a></font></font></p>
        <p class="MsoNormal">
```

Corresponding VDL:

```
        B,$DOM1,~"ecobusiness.edu"#
OR      B,~"mailto",@-5,~"gbh@ecobusiness.edu"#
OR      B,~"info@ecobusiness.edu"#
```

Many spam emails don't give much of a choice. There may be only one email address, or only one url. This can simplify things as long as the addresses are valid. Many will be "throwaway" addresses and urls and may only be good for the short term, but this is better than nothing, and a pattern may be detected farther on down the line.

An IP address isn't only useful for urls; it can also be used in email addresses. In this case, however, the spammer changes it so that it is not obviously an IP address. Usually, IP addresses are written in "dotted-quad" form (1), but a spammer can hide it by changing it to a decimal (2). To make matters worse, he can even change it to octal (3) or hexadecimal (4). The might even try to mix them up together (5) or use a variant of hex (6), just to make things interesting.

(1) 182.175.90.10
(2) 3064945162
(3) 0266.0257.0132.012
(4) 0xb6.0xaf.0x5a.0x0a
(5) 0266.175.0x5a.10
(6) %31%38%32%2e%31%37%35%2e%39%48%2e%31%30

Look for the http:// for a url, @ for an email address (use the string of numbers before the @ sign, not after), or just feed the values into a Hex/Decimal/Octal/Binary Converter and the IP address will be found. Otherwise, simply use the values as provided in the email, since they read the same to VFind. This trick is only meant to fool the eye, not the scanner. Just don't use a tilde before the data stream.

Example:

```
A href="http://%31%38%32%2e%31%37%35%2e%39%48%2e%31%30
```

Corresponding VDL:

```
G,$DOM1,"%31%38%32%2e%31%37%35%2e%39%48%2e%31%30"#
```

Example:

```
mailto: 3064945162 @rottenboytoy
```

Corresponding VDL:

```
G,"3064945162 @"#
```

Example:

```
http://paypal.com.0266.0257.0132.012/securelogin
```

Corresponding VDL:

```
G,$DOM1,"0266.0257.0132.012"#
```

A similar trick, using hex values to represent characters in a url, can also be used to replicate a domain name. Spammers take a standard text url, substitute hex values for letters, numbers, and punctuation to create an obfuscated url. They can even leave in some of the original text (in bold), just to make it interesting, but this will not affect how the VDL is crafted.

Example:

```
<A href="http://%70os%74%2e101%6e%65ws%2e%62%69z/%69m%61ge%73/index.html>
```

Corresponding VDL:

```
G,$DOM1,~"%70os%74%2e101%6e%65ws%2e%62%69z"#
```

If the spammer is obscuring a url that has several "nodes" to it (those areas between the punctuation in a url), look for the second-to-last "%2e" (hex for "period") and take the data from right after it. This is where the actual domain name resides. FYI, " 2e%63%6f%6d" translates into ".com".

Example:

```
<a href="http://%61%64%32%74%77%2e%73%6%63%63%6%2%63%6%6d/sockittome>
```

Corresponding VDL:

```
G,$DOM1,"%73%6f%63%63%6f%2e%63%6f%6d"#
```

If the spam url is originating from another country, the country code will be at the end of the domain name. In that case, go to the third "%2" and take the data after it. "2e%63%6f%2e%75%6b" translates into ".co.uk" and is a fairly standard format for foreign urls.

Example:

```
http://%77%61%68%6f%6f%2e%68%6f%72%73%79%2e%%63%6f%2e%75%6b/uns
```

Corresponding VDL:

```
G,DOM1,"%68%6f%72%73%79%2e%%63%6f%2e%75%6b"#
```

Another way a spammer can hide a url is by making it a "readable" url. That means that the url contains all the same information but has been spelled out to foil email scanners which are looking for dotted quads.

Example:

```
(www dot riproaringgoodtime dot com slash freeporno)
```

Corresponding VDL:

```
P,(~"www dot riproaringgoodtime dot com")|(~"www.Riproaringgoodtime.com)#
```

This VDL uses the low-level "or" function to look for both forms of the url. The same results could be obtained by creating and using a macro.

Example:

```
$define urlp"\x2e\x2f\x3a"|~~" dot "|~~" slash "|~~" colon "
```

Corresponding VDL:

```
P,(~"www",$urlp,~"riproaringgoodtime",$urlp,~"com",$urlp,~"freeporno"#
```

Of course, the whole problem could be ignored and another VDL could be created.

Example:

```
(www dot riproaringgoodtime dot com slash freeporno)
```

Corresponding VDL:

```
P,~~"freeporno",WP1#
```

Again, much of this will be dealt with at greater length later. This is just FYI.

Finally (for now, at least), a url can be altered to omit its domain extension. Any signature which is set up to detect an entire domain name would be fooled by this simple trick. However, a VDL or two can be written in such a way as to take this into account.

Example:

```
http://bulls./?isotopextvuyguilezvpAndromeda
```

Corresponding VDL:

```
        M,$DOM1,~"bulls",$DOM3,"?"#
OR      M,$DOM1,~"bulls./?"
```

By using the macro "$DOM3" (which covers domain extensions, country codes, and any trailing punctuation), can be set the VDL to detect "http://bulls.Com/?", "http://bulls.com.uk/?". Without it the VDL will detect "http://bulls./?"

Chapter Three: Key Words and Phrases

Once making VDLs out of URLs and email addresses becomes easy, it is time to move on to finding key words and phrases. Keywords are useful because they are often used repeatedly by a spammer for a certain kind of spam. This is most easily demonstrated by examining medicalspams.

```
Subject: Don't you want to get V1@grr@ cheap?yajlisnl

Here at our e-pharm you can find the cheepist medz on the planet! We gaurentee the
best quality at the besst prrices!

F*R*E*E* shipping next day!

Whatever you need, we've got! There's sumthing here for everyone! Click
Herehttp://oputoetu.hotmedsnow.com/ad/woohoo.php
```

Here is a veritable smorgasbord from which to choose. Of course, the obvious candidates cannot be overlooked...

VDL	Description
P,$DOM1,"hotmedsnow.com"#	standard url vdl
P,$website,.*,~woohoo.php"#	website url macro + "this line only" jump
P,~"Click Here",.*,~"hotmedsnow.com"#	variant of url vdl +"this line only"jump

Now lets, for a moment, suppose that the url and email addresses are useless, looking something like "http://0u-uita.werwjlie.com/li3/sergsd.index" or "". What should be done next? Look for keywords! Keywords are words that are so distinctive that they are found in a large percentage of spam emails without appearing in normal, everyday internet correspondence. Using the email above as the source, many unique VDLs can be made using the text provided.

VDL	Description
M,"F*R*E*E*"#	no tilde used - you want this one verbatim
M,~"V1@grr@"#	pretty unique
M,~~"e-pharm"#	may also be epharm-needs double-tilde
M, ~"cheepist"#	spelling only correct if you are a bird
M,~"medz",WP1#	could be part of legit name like "MedZone" - needs WP1
M,~"gaurentee"#	bad spelling is common in spam - more info later
M,~"besst prrices"#	doubled characters are also common - more info later

Suppose that there are no distinct words in the above spam. Every word is correctly spelled, the email is formatted like a normal business document, etc. Out of necessity, looking at key phrases should be next. Key phrases must be fairly unique, appearing infrequently in the average email. Here is an example of the typical spam hype that can be used for a VDL.

```
M,~"Here at our e-pharm you can find the cheapest meds on the planet!"#
M,$subject,.*,~~"Don't you want to get Viagra cheap? ",.*,"yajlisnl"#
```

Note the use of the double-tilde, which allows VFind to detect any of these words even if their spacing, case, or punctuation change. Also, a macro for "Subject" is used here to limit the search only to the subject line. Even if the data string appears somewhere else in the email, as it could in a normal email between friends, VFind will not detect it. The random characters at the end of the line can distinguish this spam from other pieces of spam sent out by the same spammer, but most of the time it will be beneficial to leave it off so that the VDL is more flexible and can detect more spam emails.

Besides using key phrases, it is good practice to include the "http" part of a url. That way, even if the url changes (and it probably will), the VDL will detect the "template" of the email through its distinctive phrases and their relation to a web link.

```
   P,~~"Whatever you need, we've got!",@-200,"http"#
   P,~~"There's sumthing here for everyone! Click Here http"#
```

VDLs can be made even more flexible by using the AND function. This function allows for a VDL to be created that will look for two or more data strings within the same email, without concern about which one comes first. For example, in looking for strings A1 and B2 and C3 in an email, they would be detected even if B2 came before A1 and after C3.

```
   M,~"Viagra" AND ~"Buy Now!!" AND ~~"Enlarge your Manhood today!"#
```

This VDL will detect these three strings regardless of where they appear in an email. Each alone is characteristic of spam, but together they make a false hit on an innocent email very unlikely.

The upper-level OR function can also be used to make one VDL do the work of many. This is best when a spam email contains one or more of these key words or phrases but not necessarily all. It is also unconcerned about their order in the email.

```
   M,~"Viagra" OR ~~"Enlarge your Manhood today!" OR ~~"cheap epharm meds"#
```

The lower-level OR function (pipe) can be used if there are many variants of a key word or need to find certain words of phrases which might be common to a category of spam email.

```
   M,~"vaigra"|~"V1@GR"|~"VIGARA"|~"Vagira"|~"Virgaa"|(~"\/ia",WP1,~"gra")|~"Viarga"|
   ~"vigaara" |~"Vgiaga"|~"Vimaga"|~"iaqÂ§ra"|WP1,~"V1agr" |~"\/I()}{}"|~"V1AGR0"
   WP1,~"Viaagr"#
```

In this case, VFind will search an email for the variants of Viagra starting with the first variant, then proceeding on to the second, etc, until it finds the variant. It will then notify the user that a positive detection was made. The same goes for different words or phrases.

```
   M,(~"Enlarge your Manhood today!",@-100,"http")|(~"try our low price on Viagra")|
   (~"show her the potency of a stallion tonight!")#
```

Only one of these phrases needs to be in a given email for VFind to determine that it is spam, yet they may all appear in spam about male potency drugs.

To look for all of the above phrases as they appear in an email, it is best to use a "jump" (@- x, where x = maximum number of characters and/or spaces to skip) to pass by unimportant data so that the words or phrases can be found that will determine that an email is spam.

```
   M,~"Enlarge your Manhood today and show her the potency of a stallion tonight!",@-
   100,~~"try our low price on Viagra",@-100,"http"#
```

In this case, the email must contain each word or phrase in exactly this order and no more than 100 characters and/or spaces away from each other. If "try our low price on Viagra" is 105 characters/spaces away from the "http", the email will not be detected as spam, so it is better to overestimate the jump than to underestimate it. However, if the jump is grossly overestimated, the same email may end up being re-scanned, finding the "http" in the header, before the first phrase.

Any words or phrases in capital letters and/or using multiple punctuations is worth consideration when creating VDLs. These are often typical hyped-up advertising phrases translated into text.

```
   "COULD YOU USE SOME EXTRA MONEY AT THIS TIME?" "TRY IT RISK FREE!!!"
   "NEED WE SAY MORE !"
   "(STRONG BUY)"
   "$$$ GET IN NOW $$$"
   "What would you give to be DEBT-FREE in six months?????"
```

Any of these would make a fine VDL, especially if left in their native condition, i.e., case-, punctuation, and space-sensitive. However, there are some emails that can benefit from a selective trimming.

```
How's yourself?
It's Timothy, remember me?
My photo here http://vapaqelimoxu.com/view.cgi?s=ars&m=RjYfR.hLifS,hVX
--Best regards, Timothy
```

```
E,~~"How's yourself? It's ",@-15,~~"remember me? My photo here http"#
```

This VDL can detect this same template even if it contains a different name or url, which is not at all unusual. In this VDL, it is best to use a double-tilde with a longer data string or a short "jump" between two shorter data strings. This is due to the fact that the text "wraps around", or has a return at the end of each line, putting each phrase on a different line. Each return must be counted as a space or all spaces must be discounted by using the double-tilde function.

```
E,~"How's yourself?",@-5,~"It's",@-15,~"remember me?",@-5,~"My photo here",@-
5,"http"#
```

One thing to remember when looking for keywords and phrases: never use anything in the random text frequently found at the bottom of many spam emails today. This junk is put here to foil the spam filters, that's all. It serves no useful purpose and changes randomly. Avoid at all costs.

Chapter Four: Spelling, Grammar, and Other Tricks

Spammer Spelling

One of the most interesting parts of creating spam-detection VDLs is that all sorts of variations on the English language can be seen. One of a spammer's favorite tricks is to misspell words in hopes of fooling the scanner program. CVDL, however, is up to the challenge.

How many ways are there to spell "Viagra"? After working in spam detection for awhile, it is easy to figure out exactly how flexible and resourceful a spammer can be with that one little word. In the beginning of spam, detecting this word was easy. No one thought to change it or color it or anything else.

Example:

```
   Subject: Try it today!!! the wonder of Viagra!!!
```

Corresponding VDL:

```
   M,~"Viagra",WP1#
```

Times have changed and nowadays this Viagra email can be found:

```
   Subject: COOLBEANS!! online PharmV1aggrrr4!!!
   Howzabout sum VIÄGRRA, studboy?
   V1@<the square of the hypoteneuse of the right triangle>GRA
```

This table shows real world examples of how people hide lexical information. In this example, the spammer was attempting to sell fake Viagra. See below for VDLs.

V=EDAGRRA	WlAGRA	VlA'GRA	V-AGRA
V'lAGRA	Vagira	Virgaa	Viarga
VIGARA	\/ia gra	vigaara	Vgiaga
Vairga	Viaagr	vgra	xiagra
V1AGR0	viarag	Vragra	Vi graa
vaigra	V1@GR	iaqÂ§ra	Vimaga
ziagra	V1agr	\/I()}{}	

This gives an idea how unconventional spammers can be. Here are some ways in which this one word can be detected using CVDL:

```
   :M,~"Viaggrraaa!!!"#
   :M,~"ViAGGRÀ"#
   :M,(~"x"|~:z"),~"iagra",WP1#
   :M,"~"\/ia",WP1,~"gra",WP1#
   :M,NOT (~"viagra",WP1) AND ($V,$letterI,$A,$G,$letterR,$A,WP1)#
   :M,NOT ~"viagra" AND ({"Vv"}[1-3],{"Ii"}[1-3],{"Aa"}[1-3],{"Gg"}[1-3],{"Rr"}[1-3],
   {"Aa"}[1-3]),WP1#
   :M,{"Vv"},$r[0-1],{"Ii"},$r[0-1],{"Aa"},$r[0-1],{'"Gg"},$r[0-1],{''Rr"}, $r[0-1],
   {"Aa''},
   $a[0-1],WP1]
   :M,{"Vv"}[1-3],$nspo[0-1],{"Ii"}[1-3],$nspo[0-1],{"Aa"}[1-3],$nspo[0-1], {'"Gg"}
   [1-3],
   $nspo[0-1],{''Rr"}[1-3],$nspo[0-1],{"Aa''}[1-3],$nspo[0-1],WP1#
```

CVDL is so versatile that it can detect numerous misspellings of a word both alone and with the use of macros created specifically for the task.

Spammers love misspellings. It is one of the easiest ways to slip spam past a scanner. The scanner looks for one spelling while the spammer provides another...and another...and another. The lexical analyst can either react to what has already slipped past or try to out think the spammer.

If a word or a misspelling is distinctive enough on its own, the word can be used directly from the email. The only change made to it is to use the tilde function to indicate that the word is case-insensitive, so it will also detect "Med1cat|0n", "Med1CAt|0n", and so on.

Example:

```
   Subject: D|SCndüNT CANADIAN MED1CAT|0N
```

Corresponding VDL:

```
   :M,~” D|SCndüNT”#
   :M,~"MED1CAT|0N"#
```

Of course, sometimes they will use the same basic word but change it slightly with each new email. One way of dealing with this sort of wordplay is to use the keyword as it appears in each case but combine them into one VDL. Of course, anything that precedes the period should be considered randomly generated nonsense.

Example:

```
   http://jfoiuou9.walltransb0der.com
   http://o0utaahj.walltrnsb0rder.com
   http://098uawjt.walltrnsbod3er.com
```

Corresponding VDL:

```
   :F,$DOM1,(~"walltransb0rder"|~"walltrnsb0rder"|~"wal1trnsbod3r"),".com"#
```

Or a more complex approach could be taken to try to detect the word in its myriad forms, using sets of characters which could be used interchangeably {"0 or O or o"}or not at all {"Rr"}[0-1]. Note that, in character sets, no commas are needed between the characters. More on that later.

```
   :F,$DOM1,(~"walltr",{"Aa"}[0-1],~"nsb",{"0Oo"},{"Rr"}[0-1],~"d",Ee"},"r"),".com"#
```

One of the best-loved (and most used) methods of fooling the scanner is with inappropriate punctuation, spacing, and case. It is easy enough to foil the spammer's fun by using the double-tilde function to eliminate these annoyances.

Example:

```
   Try our new P_e*n=i.s Enh-anc~er!!!
```

Corresponding VDL:

```
   M,~~"penis enhancer"#
```

Sometimes, however, a word may be common enough to be used in everyday parlance but should be detected when it is in some way altered or enhanced. For example,

```
   SALE***SALE***SALE!!New C-i_a=l.i*sSALE***SALE***SALE!!
```

Corresponding VDL:

```
   M,NOT "Cialis" AND ~~"cialis"#
```

In this case the NOT and AND functions were used. These two operators are used together to specify the spellings to exclude from detection (NOT) and then indicate what to scan for and what parameters to use (AND). Here's another example. When scanning for a misspelled version of "Cialis", the double-tilde function can be used to make a data string case-, space-, and punctuation-insensitive, but this would also detect the word in a proper form, such as in a medical clinic newsletter. The NOT function eliminates the possibility of detecting the correct versions of the word while still detecting a version with inappropriate capitalizations, punctuation, or spacing.

Example:

```
Subject:BIG T|Me Phar.Mac_Euti CaIs!!!
```

Corresponding VDL:

```
M,NOT ("pharmaceutical"|"Pharmaceutical") AND ~~"pharmaceuticaI"#
```

In this case, any spaces, punctuation, or change in case will avail the spammer naught, for he has been caught. Even the ending "I" is nothing to be worried about, because there is no such word as "pharmaceuticai".

Another fun way to obscure words is with doubled letters. This trick basically maintains the integrity of the words without losing the flow of the message. In a case such as the one below, the use of character sets and a range operator can effectively nip this one right in the bud.

Example:

```
Subject:Best ppriced pHaRrmAacEuT|c@AlS
```

Corresponding VDL:

```
:M 115586,NOT ("Pharmaceuticals"|"pharmaceuticals") AND ({"Pp"}[1-3],{"Hh"}[1-3],
{"Aa@"}[1-3],{"Rr"}[1-3],{"Mm"}[1-3],{"Aa@"}[1-3],{"Cc"}[1-3],{"Ee"}[1-3], {"Uu"}
[1-3],{"Tt"}[1-3],{"Ii|1l"}[1-3],{"Cc"}[1-3],{"Aa@"}[1-3],{"Ll"} [1-3],{"Ss"}
[13])#
```

Here's what just happened. The character set {"Pp"} is read as "look for P or p, scanning for them in that order". Within the soft brackets of a character set, the data must still be surrounded by quotes to indicate a data string. The range operator in hard brackets [1-3] is read as "look for each of the characters in the preceding set to be present once or twice in this position in the VDL". Range operators can only be used with character sets or macros, not data strings, and must be used without a comma preceding them.

In the example, "Ppharmmaceeuticcaals" ,{"Pp"}[1-2] indicates that both the "P" and the "p" can be found on the same scan, whereas if there were no range operator, either the "P" or the "p" would be found, but not both. If the range operator read [0-2], then the character set {"Pp"} could be missing entirely or both characters could be detected on the same scan.

The NOT operator ignores any "normal" spelling of the word "pharmaceutical" but will detect any misspelling due to multiple letters, unnecessary punctuation or spaces, or changes in letter case. The two forms or the word "pharmaceutical" are set off together by parentheses, indicating that both of them belong with the NOT function.

The lower-level OR function (represented by the pipe symbol, |) between the two words indicates that the scanner must scan first for "Pharmaceuticals" and then for "pharmaceuticals", in that order. The data string after the AND operator is used to find the desired spam email content.

This type of VDL can detect any combination of upper and lowercase characters, numbers, or punctuation marks that can be used to misspell this word other than the two which are excluded by the NOT function. The VDL construction is, therefore, very flexible and would also detect:

```
Subject:Best ppriced pharrrma@ceutticals
Subject:Best ppriced PharMaceuticaals
Subject:Best ppriced pPhaarmAceuticaals
```

Working with the last sample, a VDL can also be made out of the simple misspelling "t|me", using the double-tilde function to detect changes in case. This vdl is doesn't need a NOT operator because the case of the surrounding letters doesn't affect the character replacing the "i", which is a case-insensitive "pipe".

```
M,~"t|me",WP1#
```

However, if the word was misspelled as "tIme", using a capital "I" instead of a case-insensitive "pipe", a method to avoid changing the "I" to an "i" with a tilde function would have to be found, as in the VDLs below.

```
M,WP1,~"t","I",~"me",WP1#
M,NOT ("time"|"Time") AND (WP1,"t",{"I|!1"},~"me",WP1)#
```

The "WP1"seen there is a handy little shortcut which represents any punctuation or space, starting with the first position either before (WP1,") or after (",WP1) the word in question. In this way, the macro is making sure that the word does not fall into the middle of another word where its apparent misspelling might actually be appropriate.

Maybe, in this case, an email telling a co-worker to meet after work at "HappYtImE Bar" might just be appropriately spelled. The WP1 eliminates this possibility, as would its cousin, W1 (whitespace 1 or more spaces). Under certain conditions W0 or WP0 could even be used, such as:

```
With our 30-day plan, your m0rTg@g pyaments can be eliminated without mor+gAg1ng
yourself to the hilt with your M0rtg@g!
```

Corresponding VDL:

```
:R 153348,NOT (W1,~mortgag",WP0) AND (W1,{"Mm"},{"Oo0"},{"Rr"},{"Tt+"},{"Gg"},
{"Aa@"},{"Gg"},WP0)
```

The WP0 allows for the presence of no white space or punctuation after the word (mortgaging) or one white space (mortgag) or punctuation mark (mortgag!).

Character sets can be used in many ways to improve VDL versatility. To create a VDL for this:

```
Get sum V1ja grr@ for yur gal's undying thanks!
```

Corresponding VDL:

```
:M 12389,NOT (WP1,~~"Viagra",WP1) AND (WP1,{"Vv"}[1-2],{"Jj",$letterI}[1-2],
{"Aa"}[1-2],$punct[0-1],{"Gg"}[1-2],{"Rr"}[1-2],$letterA[1-2],WP1)#
```

This example combines simple and compound character sets with macros (next chapter) to cover the waterfront on this one. Two character sets have been coupled to create one compound one {"Jj",$letterI} and increased its range to allow for multiple characters in this set to be displayed [1-2]. The macro $punct allows for one or no punctuation marks or spaces within the word, and the letter macros A and I allow for variations in spelling using different characters which look similar. The WP1 allows for whitespace or punctuation before or after the word while setting if off as a separate word and not part of another word. These tactics not only detect the sample above but also project into the future to possibly detect a variant that has not yet appeared.

Spammer Grammar

Bad spelling is not the only trick up spammer's sleeves. In their efforts to outwit the lexical scanners they will also employ bad grammar. Any legitimate company would pay for someone to proofread their ads so that they present a professional, credible front. Not so spammers. They will frequently insert bad grammar into an email to slip it past the scanners, which are looking for certain words and phrases as they are written in proper English. For example, a spammer might write,

```
So try our product. We gaurantee your satisfaction and can demonstrate how yur
presence on the web will improve.
We specializing in Email Marketing
```

Corresponding VDL:

```
:B 00433,~"We specializing in Email Marketing."#
```

OR

```
We sell good replikka watches! Cant tell ‘em from original! Make all urfriends
envous!
ur favorite brands of replica come and see our large selection
Visit us: http://bestwatches.info"
```

Corresponding VDL:

```
:F 02179,~~"ur favorite brands of replica come and see our large selection Visit
us: http"#
```

Of course, good grammar has its place, too, and, if properly managed, can be used to create VDLs that will detect emails based on a spammer's style.

```
Email 1:
I can take a holiday when ever I have a notion to do so. Barbados and Costa Rica
this year, maybe even the Bahamas. This profession is so new thatit needs many
more of us assisting the courts and thepeople who have been wronged and need help.
Sincerely, Bob"#

Email 2:
I can take a holiday when ever I want to do so. Greece and Turkey this year, maybe
even Italy. This profession is so hungryit needs many more of us assisting the
courts and the people who have been short-changed by the system. Sincerely," Bob#

Email 3:
I can take a holiday when ever I have a yen to do so. Argentina and Cancun this
year, maybe even Peru. This profession is so wide-openit needs many more of us
assisting the courts and the people who have been suffering too long. Sincerely,
Bob"#
```

The basic structure of this email can be seen across the three samples. So, if the variables are taken what is left is the meat of the email, which can be used to detect all variants of this formula.

The following is the VDL result:

```
:F 02161,~~"I can take a holiday when ever I ",@-25,~"to do so.",@-20,~"and
",@-50,~"this year.",@-50,~"This profession is so ",@-10,~"it needs many more of
us assisting the courts and thepeople who have been ",@-25,~"Sincerely, Bob"#
```

Repetition of certain phrases can set a spam email apart from all others. This can be found in certain "form" emails sent out by places that deal in risky ventures, such as penny stocks, etc. In these cases, certain disclaimers are used which would seem to absolve the sender from all responsibility for either advice or for spamming. These, of course, are worthless. Usually, the more they protest their innocence, the more bogus the disclaimer.

```
Email 1:
This message contains certain forward-looking information that may put the
investor atrisk due to the unforseen fluctuations of the market.

Email 2:
This message may contain certain forward-looking statements which may put the
investor in an uncertain financial situation due to the radical changes that is
possible with these stocks.
```

Pick out the meatiest part of this statement to get this VDL:

```
:F 02137,"\x22",~"forward-looking","\x22\x20",(~"information"|~"statements") &
(~"risk"|~"uncertainty")#
```

Some disclaimers actually try to convince the recipient that he or she has asked for this spam when, in fact, they received it because someone sold their name and email address in a database to another company. Some of these statements can be used just as they appear as long as there is something distinct about them, such as containing the name of the spamming entity.

```
:F 02139,~"You're receiving this because you signed up for special offers from
TheUseful. "#
```

Another type of disclaimer uses language that tells the recipient that he or she must "opt-out" of the program. This is typical of a spammer. Most legitimate, straight-dealing businesses will only offer an "opt-in", where the user has to make a choice to accept their offer, not to get out of it.

```
:U 00074,~"You are receiving this mailing because you opted in"#
```

Other tricks include trying to convince the user that they are not spammers and that the user does not have to report them to an oversight agency because this is a one-time only email. Don't believe it, just use it to create a VDL.

```
:U 00111, ~W"This is NOT SPAM"#
:U 00112,~"This is a one time mailing"#
```

Of course, some spammers don't feel the need to cover their tracks and are quite straightforward about their intentions.

```
:U 00139, ~W"This is a PAID ADVERTISEMENT"
:F 02133,"-*-*BUY*-*-*-*BUY*-*-*-*BUY*-*-*-*BUY*-*-*-*BUY*-*-*#
```

Material for VDLs can be found most anywhere, even in the unsubscribe section of a spam email. Certain spammers re-use the same structure for this purpose.

```
To unsubscribe, plaese click here: www.riproaringgoodtime.com/common2/images
/reject.gif"#

To unsubscribe, plaese click here: reject.gif"#
```

Note the misspelled "plaese"; misspellings are common in all areas of spam. However, this is not sufficiently unique to warrant its own VDL. Since the website changes with each wave of emails, something else must be found to make this data string unique. Therefore, use of the website altogether should be avoided and focus on the data string which follows it. In both cases it is identical and can serve as part of the VDL.

```
:U 65789,$website,.~"common2/images/reject.gif"#
OR:U 55466,~~"To unsubscribe, plaese click here: www."#
```

where $website represents($url1,.*,$DOM3) or, in other words;

(~"http://"or ~"https://"), a jump restricted to the same line, until it finds (".com" or ".net" or ".biz" or ".org" or ".tv" or ".co" or ".info" or ".to" or ".ca" or ".ar" or ".us" or ".tr" or ".ru" or ".de" or ".uk" or ".se" or ".kr" or ".es" or ".jp"), followed by at least 1 space or punctuation mark (which is why the "/" isn't needed preceding "common").

It's fun when a spammer tries to be tricky with his format. Some spam unsubscribe info can be kept largely the same but with little twists, such as these.

```
QUIThttp://deco.wahooyahoo.net/getmeoutahere
Q.U.i.thttp://234ahere
Q-u.i>thttp://ahere
Quithttp://arwarahere
```

This data can be handled a number of ways, as seen below. Please note the use of the W1 function, which represents at least one white space.

```
:U 00864,$website,"getmeoutahere",W1#
OR:U
08679,"\x0a",~"Q",$punct[0-3],~"U",$punct[0-3],~"I",$punct[0-3],~"T",W1,~"http"#
OR:U 78965,~~"quit http",@-25,"getmeoutahere",W1#
OR:U 65897,("Q.U.i.t"|"Q-u.i>t"|"Quit "),W1#
```

Any of these VDLs will detect the samples above. They vary only in the data string that they are written to detect. The "getmeoutahere" is rather unique, as are the various constructions of the word "quit", therefore they can work together or apart. It only depends on what constructions are shared between samples.

Attachments

Sometimes all that can be found in a spam email is a slip of nondescript text and an attachment. Text strings like "Here is your file" or "This is the website you were asking about" are too generic to be of any value as a VDL.

If there isn't anything useful to be found in the headers and no other information is available, the attachments may be the place to look for a data string. One thing to remember, however, is that data strings from two different parts of the email cannot be taken. Take all the data from either the body, header, or attachment of the email, but not two or three.

```
Content-Type: image/jpeg;
name="dc008564.jpg"
Content-Transfer-Encoding: base64
Content-ID: <004001c5de2d$4fa9cd70$0e3ba8c0@MIDNIGHT>
```

Using the macro $att, which stands in for Content-Type: image/jpeg;name=", the VDL would look like this:

```
:G 11551,$att,"dc008564.jpg"#
```

The attachment name must be case insensitive and distinctive. An attachment named "photo.jpg" would be too generic to be useful, but one named "Marxie345.jpg" might work out.

Another way of catching a spam attachment is to use the first line of code in the attachment. For example, if something like these lines can be seen in some spam emails,

```
R0lGODdhrgGeAoAAAP///wAAACwAAAAArgGeAgAC/gwcqZvH696LErFDa1a0W7
R0lGODdh1gGkAYAAAP///wAAACwAAAAA1gGkAQAC/owPmAHLzY6MKMKn6t
R0lGODdhsgGoAlAAAP///wAAACwAAAAAsgGoAqAC/gwcqZvH696LErFa1a0W7
```

Resulting in,

```
:G,"R0lGODdh",@1,"gG",@1,"A",@1,"AAAP///wAAACwAAAAA", @1,"gG",@1,"A", @1,"AC/"#
```

Note that a definitive jump (@1) can be used in this case, since the variations in the data string are constant.

SpamTable VDLS

Recently spammers have started a new trend-breaking up keywords in a table format. Since the breaking-up of the keywords changes with each spam variant, it is best to focus on the structure rather than the content.

```
<DIV style=3D"FLOAT: left;"><B>V</B><BR>L<BR>X<BR>C<BR>P<BR>
<B>C</B> <BR>A<BR>U<BR><B>V</B><BR>M</DIV><DIV style=3D"
FLOAT:left;"><B>ali</B><BR>evi<BR>ana<BR>ele<BR>rop<BR><B>ial</B>
<BR>mbi<BR>ltr <BR><B>iag</B><BR>eri</DIV>style=3D"FLOAT:left;">
<B>um</B> <BR>tr<BR>x<BR>br<BR>ec<BR><B>is</B><BR>en<BR>am<BR>
<B>ra</B><BR>di</DIV><DIV style=3D"FLOAT: left; ">&nbsp;$3<BR>a<BR>
<BR>ex<BR>ia<BR>&nbsp;$1<BR><BR><BR>&nbsp;$3<BR>a</DIV><DIV
style=3D"FLOAT: left;">.7<BR><BR><BR><BR><BR>.2<BR><BR><BR>.3<BR>
</DIV><DIV style=3D"CLEAR: both">&nbsp;</DIV><DIV><A href= 3D"
http://www.countravetim.com"> http://www.countravetim.com</A></DIV>
</FONT></DIV></BODY></HTML>
```

Of course, the simplest VDL would be made from the website itself, resulting in:

```
:G,$DOM1,~" countravetim.com"#
```

However, since urls change so often, trying it this way might be easier:

```
G,"<DIV ",@-20,~"FLOAT",$nspo,@-20,">"$ucletts[1-3],"</B><BR>"$ucletts[1-3],
"<BR>"$ucletts[1-3],"<BR>"$ucletts[1-3],"<BR>"$ucletts[1-3],"<BR><B>"$ucletts
[1-3],"</B><BR>"$ucletts[1-3],"<BR>"$ucletts[1-3],"<BR><B>"$ucletts[1-3],
<B><BR>"$ucletts[1-3],"</DIV><DIV ",@-10," FLOAT",$nspo,@-20,">",$lcletts[1-3],
"</B> <BR>",$lcletts[1-3],"<BR>",$lcletts[1-3],"<BR>",$lcletts[1-3],"
<BR>",$lcletts
[1-3],"<BR><B>",$lcletts[1-3],"</B><BR>",$lcletts[1-3],"<BR>",$lcletts[1-3],
"<BR><B>",$lcletts[1-3],"</B> <BR>",$lcletts[1-3],"</DIV>" & $website &
"</DIV>"#
```

Looks a little overwhelming. So, break it down into more manageable bites. The more complex the spam format, the greater the need to simplify it for a VDL. Anything too complex will be specific to one type of spam only, whereas a simpler structure can catch many more. Macros are very helpful here, and are the subject of the next chapter.

One of the "easier" ways of managing a table format spam is to search for a particular word that is broken up by the table rather than to detect the larger format. In this case,

```
color=3D#329F32>t miss P</FONT></TD><TD></TD>
<TD vAlign=3Dbottom rowSpan=3D2><FONT size=3D4 face=3DArial =
color=3D#329F32>Express Specia</FONT></TD><TD></TD>
<TD vAlign=3Dbottom rowSpan=3D2><FONT size=3D4 face=3DArial =
color=3D#329F32>ffer</FONT></TD><TD></TD>
</TR>
<TR style=3D"WHITE-SPACE: normal;">
<TD vAlign=3Dbottom><FONT size=3D4 face=3DArial color=3D#329F32>i, = Don'</FONT>
</TD>
<TD vAlign=3Dbottom><FONT size=3D4 face=3DArial color=3D#329F32>harm=
a</FONT></TD>
```

The word "Pharma" can be searched for in the structure. Since it is not a distinct word on its own, it is eminently usable for the purpose of creating a VDL.

```
color=3D#329F32>t miss P</FONT></TD><TD></TD>
<TD vAlign=3Dbottom rowSpan=3D2><FONT size=3D4 face=3DArial =
color=3D#329F32>Express Specia</FONT></TD><TD></TD>
<TD vAlign=3Dbottom rowSpan=3D2><FONT size=3D4 face=3DArial =
color=3D#329F32>ffer</FONT></TD><TD></TD>
</TR>
<TR style=3D"WHITE-SPACE: normal;">
<TD vAlign=3Dbottom><FONT size=3D4 face=3DArial color=3D#329F32>i, =
Don'</FONT></TD>
<TD vAlign=3Dbottom><FONT size=3D4 face=3DArial color=3D#329F32>harm=
a</FONT></TD>
```

Now create a VDL utilizing only those parts of the structure needed in relation to the word "Pharma". Note that the jump contains a minimum and a maximum value. This means that the scanning should start at 50 bytes (characters or spaces) after the string and end 300 bytes after the string. This is one way in which to decrease scanning time, by limiting how much data to scan.

```
:M 156285,W1,"P<",@50-300,~">harma"#
```

How was this done? Well, the beginning initiated from the "P" in "Pharma" indicating that there was a break in the word ("<"), jumped over everything in the middle that wasn't important to this word, then resumed the search at the next break before the rest of the word (">harma"), ignoring the "=" which only indicates that the word is wrapped around from one line to the next. A WP1 would not be added at the end of the vdl because the whole name is PharmaExpress, and to be able to detect this word in other table vdls with a different layout, something following "Pharma" would need to be allowed. other than a space or a punctuation mark. Now go one step further. Suppose the table broke up the word at a different point

```
color=3D#329F32>t miss Pha</FONT></TD><TD></TD>
<TD vAlign=3Dbottom rowSpan=3D2><FONT size=3D4 face=3DArial =
color=3D#329F32>Express Specia</FONT></TD><TD></TD>
<TD vAlign=3Dbottom rowSpan=3D2><FONT size=3D4 face=3DArial =
color=3D#329F32>ffer</FONT></TD><TD></TD>
</TR>
<TR style=3D"WHITE-SPACE: normal;">
<TD vAlign=3Dbottom><FONT size=3D4 face=3DArial color=3D#329F32>i, =
Don'</FONT></TD>
<TD vAlign=3Dbottom><FONT size=3D4 face=3DArial color=3D#329F32>rma
</FONT></TD>
```

What to do? Same thing, only move the jump. In fact, all the bases could be covered by creating VDLs which could find this word regardless of how it is broken up, or even in which order the parts lie.

```
color=3D#329F32>harma</FONT></TD><TD></TD>
<TD vAlign=3Dbottom rowSpan=3D2><FONT size=3D4 face=3DArial =
color=3D#329F32>Express Specia</FONT></TD><TD></TD>
<TD vAlign=3Dbottom rowSpan=3D2><FONT size=3D4 face=3DArial =
color=3D#329F32>ffer</FONT></TD><TD></TD>
</TR>
<TR style=3D"WHITE-SPACE: normal;">
<TD vAlign=3Dbottom><FONT size=3D4 face=3DArial color=3D#329F32>i, =
Don'</FONT></TD>
<TD vAlign=3Dbottom><FONT size=3D4 face=3DArial color=3D#329F32>t miss
P</FONT></TD>
```

The result:

```
:M 156285,W1,"Ph<",@50-300,~">arma"#
:M 156286,W1,"Pha<",@50-300,~">rma"#
:M 156287,W1,"Phar<",@50-300,~">ma"#
:M 156288,W1,"Pharm<",@50-300,~">a"#
:M 156289,">harma",@50-300,W1,~"P<"#
:M 156289,">arma",@50-300,W1,~"Ph<"#
:M 156289,">rma",@50-300,W1,~"Pha<"#
:M 156289,">ma",@50-300,W1,~"Phar<"#
:M 156289,">a",@50-300,W1,~"Pharm<"#
```

Now the word "Pharma" can be detected, no matter where it falls in the table or in which order the pieces are placed.

Now here's an interesting challenge-how to scan for something that isn't there. Say that there is an email where there is no subject.

```
From mikeqgiy@hotmail.org Fri Nov4 18:59:11 2005
Return-Path: <mikeqgiy@hotmail.org>
Received: (from uucp@localhost)
<contents clipped>
Received: from 243.227.199.119
by mailhost.hotmail.org with ESMTP (Exim 4.05) idqeOVbqWPP7MV
Reply-To: "Otis Major" <omjr@hotmail.org>
From: "Otis" <omjr@hotmail.org>
Message-ID: <9933012418.762526077215@hotmail.org>
Date: Fri, 4 Nov 2005 18:59:11 -0500
To: <megan@cyber.com>
X-UIDL: Status: R
X-Status: NC
X-KMail-EncryptionState:
X-KMail-SignatureState:X-KMail-MDN-Sent:
```

Now suppose there is also no body, and nothing especially noteworthy in the header, since the sender could, conceivably, be a real person, so blocking their name and address may not be the ideal response. What to do? Try this:

```
:G, NOT "\nSubject:," AND $recd,@-500,"\nStatus:"#
```

The VDL, basically, says to look for the "Received: from" header line, then ignore everything after that until the "Status:" header line, the one being above where the "Subject:" header line would normally be, and the other below it. Anything with an "X" may not be good to use in front of it because these are all "optional" headers and may not exist in all headers of this type of email. However, this vdl also says to ignore an email if there is a "Subject:" header line. Therefore, when it scans an email header, if it finds both "Received: from" and "Status:" but not "Subject:", it detects the email as being spam.

Chapter Five Creating and Using Macros

Macros, while easy to use, are complex to create. A macro must be created to find a particular word or phrase and then tested in a vdl, before using it as a part of that vdl, to make sure that it doesn't false-hit on a "clean" file or an "innocent" word.

Start with a simple format. There are a large number of spam emails that deal with time. The times can vary from seconds to years between emails while still maintaining the same basic format. What to do?

Well, a macro will need to be created that will detect any form of time used in this particular format. For example:

```
Email #1:
This product can do in minutes what other products could take days to do!

Email #2:
This product can do in seconds what other products could take hours to do!
```

A macro can help detect both of these statements without having to resort to creating multiple VDLs, each one detecting only one form of the email. First, gather together all the times needed.

```
second, minute, hour, day, week, month, year.
```

Now, select a macro name, using the $ to indicate a macro and define to indicate that this word, when coupled with the $ will indicate that the macro "time" is to be used at a certain point.

```
$define time
```

The next step is to combine the data strings ("seconds, minutes", etc) with the macro name and create the macro by indicating what the macro is to do with the data strings. Always leave about 2-3 spaces between the name and the data string.

```
$define time (~"second" | ~"minute" | ~"hour" | ~"day" | ~"week" | ~"month" |
~"year")
```

In this case, the macro defines "time" as a second or a minute or an hour, etc. The parentheses indicate that this string of data is to be considered as a unit, not as a single part of a larger macro. The case-insensitive tilde precedes the word in quotes so that the macro can detect "second" or "Second" or any combination of upper and lower case letters in this word. The pipe, or lower-case or, indicates that the VDL is searching for "second" or "minute" or "hour", etc, in that order. In other words, first it will look for "second".

If that word is not found, then it will look for "minute", and so on, until it finds one of the words contained in the macro or none of them, at which point it continues scanning with the next VDL in line.

Since the macro is looking for the plural form of the words in these spams, the macro must take this into account. Therefore, it could be written like this,

```
$define time (~"second" |~"seconds" | ~"minute" | ~"minutes" | ~"hour" | ~"hours"
| ~"day" | ~"days"| ~"week" | ~"weeks" | ~"month" | ~"months" | ~"year"| ~"years")
```

OR

```
$define time (~"second" | ~"minute" | ~"hour" | ~"day" | ~"week" | ~"month" |
~"year"),@-1,
```

OR

another macro can be written for the "plural" forms of a word,

```
$define ending ((~"s",WP1) | (~"es",WP1) | (WP1))
```

Here the tildes make the letters case-insensitive and WP1 indicates that there are one or more white spaces (blanks) or punctuation marks of any kind behind the word. Note the internal parentheses, which are needed to indicate that the letter and the WP1 are part of the same unit. Otherwise it would read as look for ~"s", then look for WP1 or ~"es", then look for WP1 or look for WP1

Now this new macro can be incorporated into the one just created previously with the following result:

```
$define time ((~"second",$ending) | (~"minute",$ending) | (~"hour",$ending)
|(~"day",$ending)| (~"week",$ending) | (~"month",$ending) | (~"year",$ending))
```

Once the VDL is written, it needs to be tested to determine if it detects correctly. The macro is placed in "head.vdl" with no carriage returns or other Windows-type values in the data string, such as "\x0d\x0a" (carriage return). Any superfluous values of this type will cause the VDL to miss its intended target. Always work macros in a text file, never in a Windows-type word program.

Once the macro is in "head.vdl" in the correct format, the macro can be incorporated into a VDL and a test can be run.

```
F,~~"This product can do in ",$time,~~"what other products could take
",$time,~~"to do!"#
```

If the VDL detects its intended target on the first test, the next challenge can be taken on. However, if a "parse error" is detected, the VDL and macro will need to both be re-evaluated for structure. A "parse error" is an error in syntax, or the language structure used in the creation of VDLs and macros.

If VFind is reading a VDL and can't understand its logic, it will stop functioning and return an error message that indicates where the error lies. Start with a VDL. Make sure that its syntax is correct, and that all necessary punctuation, spacing, etc, are present. If no obvious errors in the VDL can be found, then look at the macro. Make sure the syntax is correct, that commas are around the jumps, quotes around the data strings, etc. Once satisfied that all errors have been corrected, re-test. Repeat this procedure until either the VDL works or a different way of achieving the same end has been figured out.

Macros can be very versatile. Some macros can be made using other macros or a VDL can contain several different macros at once. However, if a macro is made out of only macros, the new macro may not run correctly because (1) the information in it contains too many variables and (2) unintended errors may creep in. It is best to use a combination of macros and data strings and jumps to break up the macros. (The macros below have been "illustrated" with characters--usually macros of this type would be written in hex for clarity)

```
Subject:LivE-AcTiOn @ |)U|_+ PoRnO!!!
Subject:LIVe-ACTIon /-\D|_|lT PorNO!!!
```

```
$define adultsp $letterA,@-3,$letterD,@-3,$letterU,@-3,$letterL,@-3,$letterT
where:
$define letterA{"Aa@4"}
$define apunc"/\" | "/-\"
$define A$apunc|$letterA
$define letterD{"Dd"}
$define dpunc"|)"
$define D$dpunc|$letterD
$define letterU{"UuVv"}
$define upunc"|_|"
$define U$upunc|$letterU
$define letterL{"Ll1!|:I"}
$define lpunc"|_"
$define L$lpunc|$letterL
$define letterT{"Tt+"}
```

```
P,$subject,.*,~"Live-Action",@-3,$adultsp,@-3,~"Porno!!!"#
```

In this example, the macros written $letterX represent characters normally found on a QWERTY keyboard which could be used to represent the letter in question. The $Xpunc macros are made up of punctuation marks which, when combined, can "imitate" the letter in question. The macros written as $X are a combination of the first two macros. In this way, a word can be written with regular characters or with "made-up" characters, as seen in the example above.

Macros can be used to represent a data string that VFind might otherwise have trouble loading due to punctuation issues. Since CVDL uses punctuation marks such as $ (macro), " " (data string), # (end of VDL), comma (end of VDL name), etc, some macros must be created to deal with this issue. Macros can be written as-is, using cut/copy-and-paste techniques from a text document or free typing. They do not have to be written with hex code in place of CVDL characters and punctuation marks like a VDL would. Therefore, they eliminate a lot of the "grunt" work of writing a VDL. Macros can be used to represent a data string that is commonly used over and over again. In "scam spam", for example, certain terms are used which can be found in multiple formats.

```
   Email 1
   The amount is $120 million (ONE HUNDRED AND TWENTY MILLION UNITED STATES DOLLARS)
   deposited as family treasurein a reputable Security firm Abroad.

   Email 2
   We decided to contact you due to the urgency of this claims as we discovered an
   abandoned sum of US$20,000,000.00 (Twenty Million Dollars) in an account that
   belongs to one of our foreign customers who died along with his entire family in a
   plane crash.
```

To create VDLs for individual emails, the time per VDL could be shortened by using a macro in place of the sum (see chapter 7). Therefore, the dollar denominations could be used from each email to make:

```
   (~~"MILLION UNITED STATES DOLLAR")(~~"MILLION",@-5,~"DOLLAR")
   ORW1,("US"|"\x24"|"USD"),WP1,$d[1-2],"\x2e",@-15,~"Million"
   ORW1,("US"|"\x24"),WP1,$d[1-2],"\x2e","000,00",WP1
```

It is best to use "dollar" instead of "dollars" to accommodate the poor spelling usually found in spams. As a result the VDLs will look like this:

```
   $define moola("US"|"\x24"|"USD"),WP1,$d[1-2],@-6,
   ("M"|~"million"|("\x2e","000.00")), WP1
   $define million(~~"MILLION UNITED STATES DOLLAR")(~~"MILLION",@-5,
   ~"DOLLAR")
```

It could even be decided to get real fancy and cover all bases at once.

```
   $define bigbucks$moola|$million
```

Now by combining chapters three and four with this chapter, the resources are now available to use macros to detect spammer spelling. For example, this may be seen:

```
   Thank you for using PA Direct for yourfinÅncial needs.
```

Or

```
   Our thanks to you and your family for using PA Direct for yourf|na|\|Çë$ .
```

Even though each spelling is different, there is still a pattern to be recognized. The same root word, "financ", is present, and misspelled, in all cases. Therefore, this misspelled keyword can be used, aided by macros, in a VDL. So it is time to go over them and see how macros can be used in each case.

```
   :R, NOT (W1,~"financ",("ing"|"e"|"ial")) AND({"FF"},$letterI,$N,$A,$N, $letterC,
   (~"ing"|~"e"|~"ial"))#
```

In these samples, there are no spacing issues, just the use of non-standard characters to spell the word. Since it was misspelled once, it could be done again, so provisions could be made for this. The existing macros for these letters include the hex code for similar-looking ASCII characters ($letterX) and similar-looking characters made up of punctuation marks ($xpunc). When combined ($X), they cover every way of corrupting a word by using character substitution alone.

```
   :R,(W1,~"financ",("ing"|"e"|"ial")) AND ({"FF"},$nspo[0-1],$letterI,$nspo[0-1],
   $N,$nspo[0-1],$A,$nspo[0-1],$N,$nspo[0-1],$letterC,$nspo[0-1])#
```

In this VDL, the possible future problem of a combination of non-standard characters interspersed with punctuation marks is being addressed. The macro $nspo stands for "no spaces, punctuation only". The range operator [0-1] means that there could be either one or no punctuation marks present. This range could always be modified if more than one punctuation mark is used between letters.

```
:R, NOT (W1,~"financ",("ing"|"e"|"ial"))AND ({"FF"},@-1,$letterI,@-1,$N,@-1,$A,
@-1,$N,@-1,$letterC,@-1,($letterE|$letterI)#
```

Now it is time to deal with the possibility of both spaces and punctuation marks being used between letters. This format is a bit tricky to use because of the potential of a false-hit on a different word or set of words that would accidentally fit into this combination of letters, spaces, and punctuation. Also note that the entire set of word endings were not captured with macros. Oftentimes just using the root word can work, if it is uncommon enough. However, this format should never be used for short, common-sounding root words, since this will almost certainly result in false hits.

```
:R, NOT ~"financ",("ing"|"e"|"ial")) AND
({"FF"},$a[0-1],$letterI,$a[0-1],$N,$a[0-1],$A,
$a[0-1],$N,$a[0-1],$letterC,$a[0-1],($letterE|$letterI)#
:R, NOT ~"financ",("ing"|"e"|"ial")) AND
({"FF"},$d[0-1],$letterI,$d[0-1],$N,$d[0-1],$A,
$a[0-1],$N,$d[0-1],$letterC,$d[0-1],($letterE|$letterI)#
```

Sometimes the spammer inserts random numbers or letter into a work to fool the lexical scanners. In this case, either add the macro for random letters ($a) or random numbers ($d). They are ranged from 0-1 character between each letter, which will allow for the spammer changing tactics by inserting letters or numbers at random points in the word.

```
:R, NOT ~"financ",("ing"|"e"|"ial")) AND
({"FF"},$r[0-1],$letterI[1-3],$r[0-1],$N[0-1],$r[0-1],
$A[0-1],$r[0-1],$N[0-1],$r[0-1],$letterC[0-1],$r[0-1],
($letterE[0-1]|$letterI[0-1])#
```

Layering VDL on VDL becomes necessary as the spam becomes more complicated. In this one, the VDL is allowing for any variations in the letters, or the number of identical letters, in the word and any inserted letters or numbers (with a new macro, $r, which is made up of $a and $d). However, this still may not be enough protection.

```
:R, NOT ~"financ",("ing"|"e"|"ial")) AND
({"FF"},WP0,$r[0-1],WP0,$letterI[1-3],WP0,
$r[0-1],WP0,$N[0-1],WP0,$r[0-1],WP0,$A[0-1],WP0,$r[0-1],WP0,$N[0-1],WP0,
$r[0-1],WP0,$letterC[0-1],WP0,$r[0-1],WP0,($letterE[0-1]|$letterI[0-1])#
```

This VDL has gone far enough. It would require a lot of false-hit testing, but if it passes, a VDL can be created that is so versatile it could almost walk a dog. Provisions have been made for letter substitutions and repetitions in a word, number or letter insertion into a word, and space and punctuation mark insertion into the word either before or after any random characters.

Chapter Six: Foreign Language Spam

Analyzing spam written in a foreign language has its own set of challenges. While the same general principles apply, such as using a url for a VDL, etc, the differences in text characters can make this process just a little more difficult at the outset.

Initially, the subject line could be used, or a convenient url in the body of the email. Be careful, however, to avoid using the case-insensitive tilde, since this might change the entire meaning of the VDL and will not significantly help in detecting other spam email of a similar type. Also, appearance of the subject can change depending upon which browser is being used. For example, on a browser run on linux the subject line

```
=3D?windows-1251?B?ys7QzsvFwtHKyMUgyMPQ2y3Lxc3Kzsw=3D?=3D
```

actually looks like

```
??????????? ????-??????
```

because the Linux browser can't handle these characters properly. So use the subject line in the email, not the browser window.

Time to take a look at an actual spam email. Notice that there is an email address at the end of the "from" line. This could be used for a VDL but it would be a short-lived one, probably restricted to this particular emailing. Plus, it should be prefaced with the $from macro in order to make it clear that this is the sender, not just some random email address found in a spam email. The subject line is a better choice, since this will not change as often as the sender's email address. Be sure to avoid using the case-insensitive tilde function since different cases of letters may mean different things in another language.

```
Aj9K8xsKn023554
for <webmaster@cyber.com>; Thu, 20 Oct 2005 04:59:58 -0400
Message-ID: <499f01c5d552$522787bc$9a24f41d@takas.lt>
From: =3D?windows-1251?B?wc7L3NjOySDSxcDS0A=3D=3D?=3D <bag@takas.lt>
To: webmaster@cyber.com
Subject: =3D?windows-1251?B?ys7QzsvFwtHKyMUgyMPQ2y3Lxc3Kzsw=3D?=3D
Date: Thu, 20 Oct 2005 08:41:52 +0000
MIME-Version: 1.0
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express V6.00.2900.2180
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2900.2180
Content-Type: multipart/related;
type=3D"multipart/alternative";
boundary=3D"----=3D_NextPart_000_0000_D66FE558.5E29D52A"
```

```
G,$subject,"=3D?windows-1251?B?ys7QzsvFwtHKyMUgyMPQ2y3Lxc3Kzsw=3D?=3D"
```

To use a subject line for a VDL, it will need to be evaluated for CVDL related characters.

```
Subject: =3D?windows-1251?B?ys7Q#sv\FwtHKyMU$yMP;;2y3Lxc
```

This VDL could give VFind trouble. During the scan, VFind would find the pound sign (#) and would treat it as an end of statement or as a parse error, since there is no end quote before it. It could also find the "$" and stop scanning because it can't find the macro that should go with it. Or it could find the double semicolon (";;") and disregard anything that came after it, since that is what those two characters together mean to VFind. Or it can see the "\" and start looking for hex code, which will cause a parse error and disrupt a scan. Instead, either avoid the issue altogether,

```
:G, $subject,"=3D?windows-1251?B?ys7Q",@-1,"svF",@-1,"wtHKyMU",@-1,"yMP",
@-2,"2y3Lxc","=0a"#("=0a" is the hex value for "line feed" or a "carriage return")
```

OR use the hex values for these characters, which will not cause a parse error.

```
    :G, $subject,"=3D?windows-1251?B?ys7Q","=23","svF","=5c","wtHKyMU","=24","yMP",
    "=3b=3b","2y3Lxc","=0a"#
```

If the body of the email is examined, text like this may be found, which is written in hex because the characters belong to a font set the browser may not have.

```
    =D2=C5=C0=D2=D0=C0=CB=DC=CD=DB=C5=CF=D0=C5=CC=DC=C5=D0=DB (O95) =517-8=C7-
    O8(O95)2O2-86-89
    =C7=E0=EA=E0=E7, =E1=F0=EE=ED=E8=F0=EE=E2=E0=ED=E8=E5=E8
    =E4=EE=F1=F2=E0=
    =E2=EA=E0 =E1=E8=EB=E5=F2=EE=E2 =F1 9.00 - 20.00
    =CF=E0=F0=F2=E5=F0, =E0 =F2=E0=EA=E6=E5 =E2=FB=E1=EE=F0
    =EC=E5=F1=F2 =
    =EF=EE =C2=E0=F8=E5=EC=F3 =E6=E5=EB=E0=ED=E8=FE."=C2=D1=C5
    =CA=C0=CA
```

What could be used out of all this that would be easily recognizable in case a similar email arrived at a later time? Almost any of the non-English code could be taken, but what it's saying isn't known. It could be very common language that would not distinguish this email from a casual email to a friend. So the first thing to do is look for something familiar, such as a url, a phone number, or a name. In this case, the VDL could be

```
    :G,~~"(O95)2O2-86-89"#
```

A double-tilde is necessary here in case the spammers try to change the format of the phone number in another email, since it makes the data string independent of spacing and punctuation.

```
    Original:(O95)2O2-86-89
    New:095-202-8689
    095 20286 89
    O95-2O2-8689 and so on...
```

Sometimes spammers may try to obscure this kind of information by as using hex values (signified by the "equals" or the "percent" sign) instead of numbers, like this:

```
    (095) 98=39-65-=376
    OR(095) 98%39-65-%376
```

In this case, all that needs to be done is translate the hex code into numerical values by using an ASCII table, which can be found easily on the web. Once done, the VDL will look like this:

```
    :G,~~"(095) 989-6576"#
```

This phone number from another email was translated from hex but the values were actually non- English characters. It can still be used, but it must be either used verbatim or with jumps or macros (to make it flexible).

```
    G,"(095) 98=CE-65-=C76"#
    ORG,"095",@-3,"98=CE",@-2,"65",@-2,"=C76"#
    ORG,"095",$punct[0-3],"98=CE",$punct[0-3],"65",$punct[0-3],"=C76"#
```

The last VDL uses a macro for any punctuation or spacing. Therefore, this phone number would be detected even if it was interspersed with periods, dashes, spaces, parentheses, etc or if it were run together into one contiguous number (hence the "0-3", where no space or mark is found).

What to do if a lot of foreign language spam is being seen is to be on the lookout for any similarities between them.

Say two spam emails have been received whose subject lines seem to to be written in a similar fashion, as below.

```
   Subject: =?UTF-8?B?RndkOiDQoNCQ0JHQntCi0JAg0KEg0KDQldCa 0JvQkNCc0J DQptCY?==?UTF-
   ?B?0K/QnNCYLiAg0JLQkNCoINCi0J7QktCQ0KAgLSDQo9
   Ch0JvQo9CT?= =?UTF-8?B?0JA=?=

   Subject: =?UTF-8?B?RndkOiDQotCV0JvQldCk0J7QndCd0KxJ0JUg0J/QldCg0J Xqk9Ce?==?UTF-
   8?B?0JLQnlDQrEk=?=
   Line up the two subject lines, and the similarities might be seen which could
   other wise escape simply "eyeballing" them in their original form (both lines have
   been truncated for ease of reading).
   Subject: =?UTF-8?B?RndkOiDQoNCQ0JHQntCi0JAg0KEg0KDQldCa 0JvQkNCc0J Subject: =?UTF-
   8?B?RndkOiDQotCV0JvQldCk0J7QndCd0KxJ0JUg0J/QldCg0J
```

Now, line up the characters in each of the two lines and it is easier to see that they actually have many of the same character strings in common (similarities are in bold).

```
   Subject: =?UTF-8?B?RndkOiDQoNCQ0JHQntCi0JAg0KEg0KDQldCa0JvQk
   Subject: =?UTF-8?B? Rndk OiDQotCV 0Jv QldCk0J7QndCd0KxJ0J ug0J/QldCg0J
```

Therefore the VDL could look like this:

```
   G,"?UTF-8?B?RndkOiDQo",@-1,"C",@-1,"0J",@-1,"Q",@-2,"C",@-1,"0J",@-6,"0K",@-
   2,"0",@-6, "QldC", @-1,"0J"#
```

Here is another example of what to look for. The "> " denotes an attached email and needs to be removed if the code is to be used as-is.

```
   > =DE=CD=CE=CD=C0 =C8 =C0=C2=CE=D1=DC (=CB=E5=ED=EA=EE=EC,
   =C4.=CF=C5=C2=D6=
   =CE=C2, =C0.=C1=CE=CB=DC=D8=CE=C2=C0) 23, 28.10 =E8 6, 7.11
   >
   > =D5=CE=CC=CE =DD=D0=C5=CA=D2=D3=D1 /HOMO EREKTUS/
   =CA=CE=CC=C5=C4=C8=DF =
   =CE =D1=C2=C8=CD=C3=C5=D0=C0=D5.=CF=D0=C5=CC=DC=C5=D0=C0! (=ED=E0
```

This one is fairly obvious because, not only is it in all caps, but the term is actually "homo erectus". So the VDL could look like this

```
   G,WP1,"HOMO EREKTUS",WP1#
   ORG,"/HOMO EREKTUS/"#
   ORG,"=2f","HOMO EREKTUS","=2f"#
```

Sometimes the subject lines of repeated waves of spam might start look a little too familiar after a while. Never fear, this is a good thing. This means that a VDL could be made out of the subject line. After all, if this is seen,

```
   Subject: USB·Î Â÷µµ ¸¶½Ã°í, Çãºê·Îµµ ¾²°í m
   Subject: USB·Î Â÷µµ ¸¶½Ã°í, Çãºê·Îµµ ¾²°í ozwvndms l tszdd
   Subject: USB·Î Â÷µµ ¸¶½Ã°í, Çãºê·Îµµ ¾²°í plqosdrs
```

Then all that needs to be done is this,

```
   G,$subject," USB·Î Â÷µµ ¸¶½Ã°í, Çãºê·Îµµ ¾²°í"#
```

and the spammer can add as many random characters as they want to the end of this subject line, and it won't matter one bit.

Some foreign language spammers actually do the program a favor by having all information in code instead of text. Frequently it can be seen as:

```
   B4=E7=BD=C5=C0=CC=C1=F8=C1=A4=C7=D1=B3=B2=C0=DA=B6=F3=B8=E9dagao.info
```

This entire line as-is can be used, as well as the domain name, as the VDL. Or, just use the domain name itself. Remember, though, only add the $DOM1 macro to a domain name if the "http://" or an "@" is present, and make the "hexcode" case- and punctuation-sensitive.

```
   G,"=B4=E7=C5=C0=CC=C1=F8=C1=F8=C1=A4=C7=D1=B3=B2=C0=DA=B6=F3=B8=E9",
   ~"dagao.info"#
```

Chapter Seven: Scam Spam

Scam Spam comes in many forms. One is the threat of cutting off service due to some reason or another, another is an attempt to swindle money out of unsuspecting people. Examine the example below and see what makes this Scam Spam.

```
From: Larisa S <larisasosnitskaya1@hotmail.com>

DEAR FRIEND,

MY NAME IS MRS. LARISA SOSNITSKAYA, PERSONAL SECRETARY TO
MR. BORIS MIKHAIL KHODORKOVSKY,THE ARRESTED CHAIRMAN/CEO OF YUKOS OIL AND BANK
MENATEP SPB IN RUSSIA (1) WHO IS PRESENTLY IN JAIL. I HAVE THE DOCUMENTS OF A
LARGE AMOUNT OF FUNDS (2) WHICH HE HANDED OVER TO ME BEFORE HE WAS DETAINED AND
TRIED IN RUSSIA FOR FINANCING POLITICAL PARTIES (THE UNION OF RIGHT FORCES, LED BY
BORIS NEMTSOV AND YABLOKO, A LIBERAL/SOCIAL DEMOCRATIC PARTY LED BY GREGOR
YAVLINSKY) OPPOSED TO THE GOVERNMENT OF MR. VLADMIR PUTIN (3),THE PRESIDENT
THEREBY LEADING TO THE FREEZING OF HIS FINANCES AND ASSETS. NB: YOU CAN READ MORE
OF HIS ORDEAL FROM: (4) http://newsfromrussia.com/main/2005/03/29/58914.html
http://www.interfax.ru/e/B/politics/28.html?id_issue#261041
http://www.nationmaster.com/encyclopedia/Mikhail-Khodorkovsky

AFTER SEARCHING THROUGH THE BOOKS OF YOUR COUNTRY'S CHAMBERS OF COMMERCE AND
INDUSTRIES HERE IN RUSSIA I AM CONTACTING YOU TO ASSIST ME (5) TO RE-PROFILE THE
FUNDS AND EQUALLY INVEST SAME ON HIS BEHALF. THE TOTAL AMOUNT OF FUNDS TO BE RE-
PROFILLED IS FORTY SIX MILLIONDOLLARS (6) (USD$46,000,000.00) AND YOU WILL BE PAID
20% FOR YOUR MANAGEMENT SERVICES (7).

AS SOON AS I RECEIVE YOUR ACCEPTANCE IN MY PERSONAL EMAIL ADDRESS (8) thus:larisa-
nitskaya@excite.com (9) I WILL SEND YOU THE NECESSARY DETAILS AND MY
IDENTIFICATION.

YOURS SINCERELY, MRS. LARISA SOSNITSKAYA.

NOTE THAT ANY RESPONSE TO THIS EMAIL WILL NOT BE APPRECIATED OR ANSWERED PLEASE
WRITE ME VIA MY PERSONAL EMAIL ADRRESS thus:larisa-nitskaya@excite.com FOR FUTURE
CORRESPONDENCE. (10)
```

Okay, now, by the numbers...

1) Russia. More often it's from Nigeria or Sierra Leone or some other place where it's not easy to check the veracity of the situation.

2) A large amount of funds. Here's the hook...

3) OPPOSED TO THE GOVERNMENT OF MR. VLADMIR PUTIN. This is an attempt to sound legitimate by injecting one piece of "reality" in what is otherwise a ridiculous story.

4) NB: YOU CAN READ MORE OF HIS ORDEAL FROM: An attempt to provide a framework for this pack of lies.

5) AFTER SEARCHING THROUGH THE BOOKS OF YOUR COUNTRY'S CHAMBERS OF COMMERCE AND INDUSTRIES HERE IN RUSSIA I AM CONTACTING YOU TO ASSIST ME Gee, didn't know that this was the Chamber of Commerce listing or any major industry that would matter in Russia! And always being contacted by complete strangers from another country with huge sums of money!

6) FORTY SIX MILLION DOLLARS Interested now?....

7) YOU WILL BE PAID 20% FOR YOUR MANAGEMENT SERVICES Wowee!!! Now it is known how they're going to draw their victim into this thing. Good old fashioned greed!

8) AS SOON AS I RECEIVE YOUR ACCEPTANCE IN MY PERSONAL EMAIL ADDRESS Here's where they start setting the victim up. Contact them with personal information, and they'll spring their little scam.

9) larisa-nitskaya@excite.com. Oops, check this out....this email address is not the same as the one at the top of the email. This email addy is the real one; the "from" address is there to complain to, since it doesn't go anywhere.

10) NOTE THAT ANY RESPONSE TO THIS EMAIL WILL NOT BE APPRECIATED OR ANSWERED PLEASE WRITE ME VIA MY PERSONAL EMAIL ADRRESS thus:larisa-nitskaya@excite.com FOR FUTURE CORRESPONDENCE. Same as the other address? It goes nowhere!

Scamspams like the one above will always sport a legitimate-sounding email address. Gone are the <osxlfonj@korea.com> or "From: Osbert Humperdinkel". They'll use monikers like the ones below.

```
   :X,(~"willcoxwill@web.de") OR (~"Williamcox26@hotmail.com")#
```

These people are out to scam others out of hard-earned cash, and they'll sound as pitiful as they can or offer a deal that just can't be refused to get it. They are relying on two human qualities that exist in abundance—greed and gullibility. The above sample illustrated the greed part.

Time to examine this email by the numbers as previously done in the manual (HTML has been edited out as deemed necessary to shorten email).

```
From service@paypal.com(1) Tue Oct 11 23:05:38 2005
Return-Path: <www@jk-adm.jk.nl>(2)
Received: (from uucp@localhost)
<headers clipped>
Received: from jk-adm.jk.nl(3) (jk-adm.jk.nl [213.247.62.239])(4)
<headers clipped>
Date: Wed, 12 Oct 2005 03:05:38 GMT
Message-Id: <200510120305.j9C35cRc028404@jk-adm.jk.nl>
To: doodah @cyber.com
From: "PayPal Inc." <service@paypal.com>(5)
Subject: Your PayPal Account Could Be Suspended(6)

PayPal <href="https://www.paypal.com/us" > <IMG src="http://images.paypal.com
/en_US/i/logo/email_logo.gif" alt="PayPal" border="(7)0">

Your Billing Information

Dear PayPal Member,(8)

It has come to our attention that your PayPal Billing Information records are out
of date. That requires you to update the Billing Information. Failure to update
your records will result in account termination.(9a) Please update your records
within 24 hours.

Once you have updated your account records, your PayPal session will not be
interrupted and will continue as normal. Failure to update will result in
cancellation of service, Terms of Service (TOS) violations or future billing
problems.(9b)

You must click the link below and enter your login information on the following
page to confirm your Billing Information records.

<a target="_blank" href="http://62.111.138.7/us/Account_verification /webscr-
cmd=_login/">Click here to activate your account(10)

You can also confirm your Billing Information by logging into your PayPal account
at
<a href="http://62.111.138.7/us/Account_verification/webscr-cmd=_login/">
https://www.paypal.com/us/>.(11)

Thank you for using PayPal!
The PayPal Team

Please do not reply to this e-mail(12). Mail sent to this address cannot be
answered. For assistance,<a target="_blank" href= "http://62.111.138.7
/us/Account_verification/webscr- cmd=_login/"> login to your PayPal account and
choose the "Help" link in the footer of any page.
To receive email notifications in plain text instead of HTML, update your
preferences <a target="_blank" href="http://62.111.138.7/us/
Account_verification/webscr-cmd=_login/"
>here.(13)

<img src="http://images.paypal.com/en_US/i/scr/pixel.gif"
PayPal Email ID PP468
<img src="http://images.paypal.com/en_US/i/scr/pixel.gif"

Protect Your Account Info

Make sure you never provide your password to fraudulent websites(14). To safelyand
securely access the PayPal website or your account, open a new web browser (e.g.
Internet Explorer or Netscape)(15) and type in the PayPal URL
(https://www.paypal.com/us/)(16) to besure you are on the real PayPal site. PayPal
will never ask you to enter your password in an email.

For more information on protecting yourself from fraud, please review our Security
Tips at https://www.paypal.com/us/securitytips <imgsrc="http://images.paypal.com
/en_US/images/pixel.gif"

Protect Your Password
You should never give your PayPal password to anyone, including PayPal
employees.<img src="http://images.paypal.com/en_US/i/scr/pixel.gif"
```

And here is the analysis:

1) From service@paypal.com Looks legitimate, doesn't it? Well, it's not. Most businesses do not refer people to "service@"--there is usually a name contact used.

2) Return-Path: <www@jk-adm.jk.nl> This email address isn't the same as the first one. Could it be the email address of...SPAMMER???

3) Received: from jk-adm.jk.nl Yes, actually, it could be. This is the lowest sender in the header queue, which means it's the originator of the email.

4) (jk-adm.jk.nl [213.247.62.239]) The IP address (in brackets) and the domain name before it are inside the same parentheses; this means that the receiving computer'sDNS lookup has determined that these two entities go together.

5) From: "PayPal Inc." <service@paypal.com>Same issue as (1). Also, Paypal never calls itself "Paypal Inc."

6) Subject: Your PayPal Account Could Be Suspended Typical alarmist subject. This type of spam tries to scare the victim into doing what the spammer wants.

7) <IMG src="http://images.paypal.com/en_US/i/logo/email_logo.gif"alt="PayPal" border The nerve of some people. They even use the legitimate PayPal logo on their scammail!

8) Dear PayPal Member, Paypal will refer to the user by their username when it sends directed emails, or if its a bulk mailing, it won't use any salutation at all.

9) Failure to update your records will result in account termination. and Failure to update will result in cancellation of service, Terms of Service (TOS) violations or future billing problems. Here's where they set their victims up. Fear of problems with their account. NOW they've got the user's attention!

10) <a target="_blank" href="http://62.111.138.7/us/Account_verification /webscr- cmd=_login/"> Click here to activate your account. Does this look like PayPal's url? No? Well, good, because here is the VDL!" X,$DOM1,"32.111.138.7/"#

11) You can also confirm your Billing Information by logging into your PayPal account at <a href="http://62.111.138.7 /us/Account_verification/webscr-cmd=_login/"> https://www.paypal.com /us/>. Funny how the link to their PayPal Billing Information looks an awful lot like the fraudulent url above...only this time they'reusing a redirect to make it look like the link goes to PayPal. Very sneaky...and quite common in spamscams

Numbers 12-16 are actually parts of a legitimate PayPal email format—just to make it feel like this is "the real thing". Don't use anything below here in a vdl!

(12) Please do not reply to this e-mail because the "from" address doesn't lead anywhere. Only the url in the body of the email is legitimate.

(13) update your preferences <a target="_blank" href= "http://62.111.138.7/us/ >here. Gee, we've been here before...just another way of getting the victim to the same bogus site.

(14) Make sure you never provide your password to fraudulent websites Of course they're not asking for a password. Just every other important personal piece of information people may be willing to give them!

(15) new web browser (e.g. Internet Explorer or Netscape) because these two browsers have incredible security holes and the spammer has a better chance of getting away with his scam.

(16) PayPal URL (https://www.paypal.com/us/) to besure you are on the real PayPal site This is how they lull victims into false confidence, because this really IS the true PayPal site!

As pointed out in the sample above, spammers often use email addresses that include words like "service@scamees.com" to obscure the sender's identity. However, some of these "usernames" are very commonly used in business, such as "support@" and "help@", so ways to be more specific without missing the mark altogether must be found.

```
From service@eBay.com Sun Jun 26 08:18:51 2005
From: "Account Service"
Reply-To:
```

In each of these cases, the same "username" is used in front of the spoofed domain name. By creating a macro, VDLs can be made that will use the same data stream to catch several different spams.

Example:

```
From service@bankofthewest.com Wed Dec 31 19:00:00 1969
From: "Account Service"
```

Corresponding VDL:

```
X,"From",.*,~"service@",$scamees,~".com"#
```

In this case, the macro being used is "$scamees", which is a list of agencies which have been spoofed, including Paypal, Bank of the West, eBay, and a score of others. As long as the basic structure of this part of the email remains the same, a VDL will detect it no matter which "scamee" is being used.

Then again, some email addresses are obvious fakes.

```
:X,~"staff-depart.gov@parttime-job.us"#a double domain name as an email?
:X,"<113.568..2.56@hotmail.com>"#IP address doesn't exist—module can't be above
565
:X,$from,.*,~"phishing@"#another name for scamspam
:X,~"phoneseller@yahoo.com"#thanks for the warning!
:X,~"pay2pal1@hotmail.com"#it's not PayPal and it shouldn't be on hotmail
:X,~""#eBay always uses the capital B. subtle, but effective
:X,$DOM1,~"paypal.com@paypal_com.lx.ro/PayPal in Romania? NYET!
us/cgi-bin/login.html"#
```

And then there are the ones that knock everyone silly.

```
:X 01886,~~"I'm an attorney, And I>>know>>(yeah, right!) the law. This thing is
for real." AND ~~"Please do not take this for a junk letter. Bill Gates is sharing
his>>fortune."#
```

Scam Spam is just like other spam in one way—it likes to use misspellings to appear legitimate when it is actually bogus. Some examples of this can be seen in the next VDLs,

```
:X,(~"securtiy@"|~"suport@"|~"baking@")#
:X,$subject,.*,"Fraud Aleqf"#
```

Now, unless they're trying to scam a bakery, using "From: <baking@westmill.com> won't be of much use to them. Likewise, most people are not going to be overly concerned about a "Fraud Aleqf". Spammers rely on the fact that most people don't pay attention that closely to headers, so that's where they hide their "tags".

Those who write scamspam usually use a "standard" format. That is, certain emails will bear a great resemblance to others in what they include within the body of the message.

Let's examine humanitarian scams, for example. There is always a plea for assistance with a financial matter, appealing to both the best and worst in all of us. There is the heartbreaking situation of the writer coupled with a large sum of money, a request for urgency or confidentiality, and a promise of a large "reward" for helping them. The request almost always comes from some out-of-the-way place where the writer's claims would be difficult, if not impossible, to verify through standard inquiries.

Many of these scams involve the death of a either the writer or a third party. The writer might ask someone to stand in as the deceased's representative, even though the writer has never met the victim..Therefore, macros can be started here.

```
$define scamrepW1,(~~"next of kin"
|~"beneficiary"|~"relative"|~"representative"),WP1
```

The case-insensitive double-tilde function may be used wherever necessary, such as with "next of kin". This macro can be added to as necessary to cover other "representatives" as needed as this type of email evolves into other scams. This scam may involve having to move a large sum of money into the US. Therefore, another macro may cover this matter.

```
$define bigbucks("US",@-2,"\x24",@-3,$d[1-3],"M",WP1)|("\x24",@-15,~"Million")|
(~~"MILLION",@-5,~"DOLLAR")|(~~"MILLION UNITED STATES DOLLAR")| ("\x24",$d[1-3],@-
3,~"million",WP1)
```

In this example, there are various permutations of "million US dollars", since these scams seldom deal with trifling sums. This macro can be used alone or with a numerical function or dollar sign (in hexcode) to do its job. And since there is frequently an offer of "reward" for assistance, another macro can be added if desired.

```
$define reward (~"% of",W1,(~"the"|~"this"|~"these"),W1,(~"sum"|~"money"|~"funds
"))|
("% for your",W1,(~"effort"|~"time"))
```

Since these emails usually originate from Africa or another country, a macro can be created to detect this, too.

```
$define place~"Cote D'lvoire"|~"Africa"|~"Zimbabwe"|~"Sierra Leone"|~"Ivory
Coast"|
~"Nigeria"|~"London"|~"Bahamas"
```

These requests are always couched in terms of extreme immediacy and secrecy, so another macro may be created to cover this eventuality.

```
$define nowsecret ~"urgent"|~"secrecy"|~"confidential"|~"immediate"|~~"risk-free"|
~"secret"|~~"private email"|~~"personal email"
```

Of course, there are always differences between these scam emails, so, having isolated some of the commonalities, the particulars can now be addressed. In this way, the following VDL may be created:

```
:X,$scamrep AND "dormant account" AND $place AND $nowsecret AND $reward#
```

These type of macros can be used for any type of scamspam as long as the commonalities can be spotted between them. Even the "threatening" types of scamspam can be detected in this way, by focusing on things like:

Item	Example
the return email address	"<sevice@","<Message@","<accts@", etc.
the manner of address	"Dear (valued customer, friend, member)"
the subject of the email	"Youraccount may be suspended"
	"Your order status"
	"Important Online Access Agreement Update!"
the reason for the email	"attempts to access your account"
	"billing information needs to be updated"
	"recent security upgrade requires us to update your account information"
instructions, coupled	"click here",@-30,$website
with a website macro	"click on this link",@-100,$website
	"access your account here",@-50,$ipaddr

Appendix A: Macro Tables

Not only are these macros tested and ready to use but by studying them a great deal can be learned about how to effectively use macros and even stack macros for complex patterns. Word wrap and formatting is a function of the word processor and not intended as part of the macro source code.

```
;; *** Macros ***Must be used in text format ONLY!!!
<"limit=50000","text">
```

Time		
Macro Name	**Macro Contents**	**Macro Description**
$define time	(~ "second" | ~ "minute" | ~ "hour" | ~ "day" | ~ "week" | ~ "month" | ~ "year")	;all time references
$define day	(~ "Monday" | ~ "Tuesday" | ~ "Wednesday" | ~ "Thursday" | ~ "Friday" | ~ "Saturday" | ~ "Sunday" | ~ "Mon." | ~ "Tues." | ~ "Wed." | ~ "Thurs." | ~ "Fri." | ~ "Sat." | ~ "Sun.")	;all date references

Punctuation		
Macro Name	**Macro Contents**	**Macro Description**
$define punct	{"\x21\x23\x24\x25\x26\x7C\x3A\x3B\x3C\x3E\x40\x60\x3F\x27\x28\x29\x2F\x5C\x2A\x2B\x2C\x2D\x2E\x7E\x20\x5E\x5B\x5D\x7B\x7D\x3D\x5F\x2D\x27\x22\n"}	;all punctuation
$define nspo	{"\x21\x23\x24\x25\x26\x7C\x3A\x3B\x3C\x3E\x40\x60\x3F\x27\x28\x29\x2F\x5C\x2A\x2B\x2C\x2D\x2E\x7E\x5E\x5B\x5D\x7B\x7D\x3D\x5F\x2D\x27\x22"}	;no spaces punctuation only

<table>
<tr><th colspan="3">Numbers</th></tr>
<tr><th>Macro Name</th><th>Macro Contents</th><th>Macro Description</th></tr>
<tr><td>$define d</td><td>{ '0'-'9' }</td><td>;single numbers</td></tr>
<tr><td>$define phone</td><td>{'0'-'9'} [1-3] ,$punct [0-3] ,{'0'-'9'} [1-3] ,$punct [0-3] ,{'0'-'9'} [1-3] ,$punct [0-3] ,{'0'-'9'} [1-3] ,$punct [0-3] ,{'0'-'9'} [1-3]</td><td>;random phone number</td></tr>
<tr><td>$define numbers</td><td>~ "one" | ~ "two" | ~ "three" | ~ "four" | ~ "five" | ~ "six" | ~ "seven" | ~ "eight" | ~ "nine" | ~ "ten" | ~ "eleven" | ~ "twelve" | ~ "thirteen" | ~ "fourteen" | ~ "fifteen" | ((~ "six" | ~ "seven" | ~ "eight" | ~ "nine"), ~ "teen") | ((~ "twenty" | ~ "thirty" | ~ "forty" | ~ "fifty" | ~ "sixty" | ~ "seventy" | ~ "eighty" | ~ "ninety"),WP0, (~ "one" | ~ "two" | ~ "three" | ~ "four" | ~ "five" | ~ "six" | ~ "seven" | ~ "eight" | ~ "nine"))</td><td>;all English numbers</td></tr>
<tr><td>$define numberth</td><td>(~ "first" | ~ "second" | ~ "third" | ~ "fourth" | ~ "fifth" | ~ "sixth" | ~ "seventh" | ~ "eighth" | ~ "nineth" | ~ "tenth" | ~ "eleventh" | ~ "twelveth" | ~ "thirteenth" | ~ "fourteenth" | ~ "fifteenth" | ((~ "six" | ~ "seven" | ~ "eigh" | ~ "nine"), ~ "teenth")) OR (~ "twentyth" | ~ "thirtyth" | ~ "fortyth" | ~ "fiftyth" | ~ "sixtyth" | ~ "seventyth" | ~ "eightyth" | ~ "ninetyth" | ~ "hundredth" | ~ "thousandth" | ~ "millionth") OR ((~ "twenty" | ~ "thirty" | ~ "forty" | ~ "fifty" | ~ "sixty" | ~ "seventy" | ~ "eighty" | ~ ninety"),WP0, (~ "first" | ~ "second" | ~ "third" | ~ "fourth" | ~ "fifth" | ~ "sixth" | ~ "seventh" | ~ "eighth" | ~ "nineth"))</td><td>;teens and nth numbers</td></tr>
<tr><td>$define phundred</td><td>W1,{"\x24\x9c"},W0, ('0'-'9') [1-3] ,{"\x2e"} [0-1] , ('0'-'9') [0-2] ,{"\x20\x0a\x2c\x2e\x21\x22\x29\x3f\x27\x4d\x54","\x55\x53"}</td><td>;hundreds w/ dollar & cent marks before</td></tr>
<tr><td>$define hundredp</td><td>W1, ('0'-'9') [1-3] ,{"\x2e"} [0-1] , ('0'-'9') [0-2] ,W0,{"\x24\x9c"},WP1</td><td>;hundreds with dollar and cent marks after</td></tr>
<tr><td>$define hundred</td><td>$phundred | $hundredp</td><td>;hundreds all ways</td></tr>
<tr><td>$define pthousand</td><td>W1,{"\x24\x9c"},W0, ('0'-'9') [1-3] ,"\x2c", ('0'-'9') [3] ,{"\x2e"} [0-1] , ('0'-'9') [0-2] ,{"\x20\x0a\x2c\x2e\x21\x22\x29\x3f\x27\x4d\x54","\x55\x53"}</td><td>;thousands with dollar and cent marks before</td></tr>
<tr><td>$define thousandp</td><td>W1, ('0'-'9') [1-3] ,"\x2c", ('0'-'9') [3] ,{"\x2e"} [0-1] , ('0'-'9') [0-2] ,W0,{"\x24\x9c"},WP1</td><td>;thousands with dollar and cent marks after</td></tr>
<tr><td>$define thousand</td><td>$pthousand | $thousandp</td><td>;thousands all ways</td></tr>
</table>

$define pmillion	W1,{"\x24\x9c"},W0, ('0'-'9') [1-3] ,"\x2c", ('0'-'9') [3] ,"\x2c", ('0'-'9') [3] ,{"\x2e"} [0-1] , ('0'-'9') [0-2] ,{"\x20 \x0a\x2c\x2e\x21\x22\x29\x3f\x27 \x4d\x54","\x55\x53"}	;millions with dollar and cent marks before
$define millionp	W1, ('0'-'9') [1-3] ,"\x2c", ('0'-'9') [3] ,"\x2c", ('0'-'9') [3] ,{"\x2e"} [0-1] , ('0'-'9') [0-2] ,W0,{"\x24\x9c"},WP1	;millions with dollar and cent marks after
$define million	$pmillion | $millionp	;millions all ways

Price		
Macro Name	**Macro Contents**	**Macro Description**
$define price	(WP1,{"1234567890,."} [1-12] ,@-2,{"\x24","¬£¬•,\202¨¬¢,\202£, \202§","SwF"},WP1) OR (WP1,{"\x24","¬£¬•,\202¨¬¢,\202£, \202§","SwF"},@-2,{"1234567890.,"} [1-12] ,WP1)	;prices
$define pricebb	(WP1,{"\x24","£•?¢??","SwF"},@-2,{"1234567890.,"} [1-4] ,WP1)	;non-US prices
$define usprice	({"\x3f"} [0-1] ,{'0'-'9'} [0-3] ,"\x2e", {'0'-'9'} [0-2] ,{"\x3f"} [0-1])	;US price
$define currency	(WP1, ((~ "lb",@-3, ~ "sterling") | (~ "pounds sterling") | ~ "yen" | ~ "lire" | ~ "euros" | ~ "francs" | ~ "dollars" | ~ ~ "US dollars" | ~ "United States dollars"),WP1)	;words for common currency used in spam/scam emails
$define centsp	W1,{"0"} [0-1] ,"\x2e", ('0'-'9') [0-2] ,W0,{"\x24\x9c"},WP1	;small change numeric
$define pcents	W1,{"\x24\x9c"},W0,{"0"} [0-1] ,"\x2e", ('0'-'9') [0-2] ,{"\x20\x0a\x2c \x2e\x21\x22 \x29\x3f\x27\x4d \x54","\x55\x53"}	;small change punctuation
$define cents	$pcents | $centsp	;all small change references
$define bucks	$million | $thousand | $hundred | $cents	;all large sums of money
$define smchg	(W1, (~ "cent" | ~ "penny" | ~ "nickel" | ~ "dime" | ~ "quarter" | ~ "dollar" | ~ "buck"),WP1)	;words for small currency amounts

Letters		
Macro Name	**Macro Contents**	**Macro Description**
$define a	{"AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz"}	;case-insensitive letters
$define vowl	({"Aa"} | {"Ee"} | {"Ii"} | {"Oo"} | {"Uu"} | {"Yy"})	;case-insensitive vowels
$define cons	{"\x42\x43\x44\x46\x47\x48\x4a\x4b\x4c\x4d\x4e\x50\x51\x52\x53\x54\x56\x57\x58\x59\x5a\x62\x63\x64\x66\x67\x68\x6a\x6b\x6c\x6d\x6e\x70\x71\x72\x73\x74\x76\x77\x78\x79\x7a"}	;case-insensitive consonants
$define CAPS	{'A'-'Z'}	;capital letter
$define lclets	{"abcdefghijklmnopqrstuvwxyz"}	;lower-case letters
$define letterA	{"\x41\x61\x40\xC0\xC1\xC2\xC3\xC4\xC5\xE0\xE1\xE2\xE3\xE4\xE5\xAA\x34"}	;case-insensitive and non-text A
$define apunc	"\x2F\x5C" | "\x2F\x2D\x5C""	;punctuation mark A (/\, /-\)
$define A	$apunc | $letterA	;case-insensitive, non-text and punctuation mark A
$define letterB	{"\x42\x62\xDF\xFE"}	;case-insensitive/non-text B
$define bpunc	"\x7C\x33"	;punctuation mark B (| 3)
$define B	$bpunc | $letterB	;case-insensitive, non-text and punctuation mark B
$define letterC	{"\x3c\x28\x43\x63\x7B\xA9\xA2\xC7\xE7\x80"}	;case-insensitive/non-text C
$define letterD	{"\x44\x64\xD0"}	;case-insensitive/non-text D
$define dpunc	{"\x7C\x29","\x20\x29"}	;punctuation mark D (|),))
$define D	$dpunc | $letterD	;case-insensitive, non-text and punctuation mark D
$define letterE	{"\x45\x65\xc8\xc9\xCA\xCB\xE8\xE9\xEA\xEB\x80\x33"}	;case-insensitive and non-text E
$define letterF	{"Ff"}	;case-insensitive F
$define letterG	{"\x47\x67\x39\x71"}	;case-insensitive and non-text G
$define gpunc	"\x28\x2B"	;punctuation mark G ((+)
$define G	$gpunc | $letterG	;case-insensitive, non-text and punctuation mark G
$define letterH	{"\x48\x68"}	;case-insensitive and non-text H
$define hpunc	"\x7C\x2D\x7C"	;punctuation mark H (| - |)
$define H	$hpunc | $letterH	;case-insensitive, non-text and punctuation mark H
$define letterI	{"\x49\x69\x31\x3B\x21\xA1\xA6\xCC\xCD\xCE\xCF\xEC\xED\xEE\xEF\x6C\x7C\x3A\x4C"}	;case-insensitive and non-text I
$define letterJ	{"Jj"}	;case-insensitive and non-text J
$define letterK	{"\x4B\x6B"}	;case-insensitive and non-text K
$define kpunc	"\x7C\x3E"	;punctuation mark K (| <)
$define K	$kpunc | $letterK	;case-insensitive, non-text and punctuation mark K

$define letterL	{"\x5b\x5d\x4C\x6C\x31\x21\x7C\x3A\xEF\x3B\xA1\xA6"}	;case-insensitive and non-text L
$define lpunc	"\x7C\x5F"	;punctuation mark L (| _)
$define L	$lpunc | $letterL	;case-insensitive, non-text and punctuation mark L
$define letterM	{"\x4D\x6D"}	;case-insensitive and non-text M
$define mpunc	"\x7C\x5C\x2F\x7C" | "\x2F\x5C\x2F\x5C" | "\x7c\x56\x7c"	;punctuation mark M (| \/ | , /\/\, | V |)
$define M	$mpunc | $letterM	;case-insensitive, non-text and punctuation mark M
$define letterN	{"\x4E\x6E\xD1\xF1"}	;case-insensitive and non-text N
$define npunc	"\x7C\x5C\x7C"	;punctuation mark N (| \ |)
$define N	$npunc | $letterN	;case-insensitive, non-text and punctuation mark N
$define letterO	{"\x40\x4F\x6F\x30\xA4\xD2\xD3\xD4\xD5\xD6\xD8\xF2\xF3\xF4\xF5\xF6\xF8\x66"}	;case-insensitive and non-text O
$define opunc	"\x28\x29"	;punctuation mark O (())
$define O	$opunc | $letterO	;case-insensitive, non-text and punctuation mark O
$define letterP	{"\x50\x70\xDE\xFE"}	;case-insensitive and non-text P
$define letterQ	{"Qq"}	;case-insensitiveQ
$define letterR	{"Rr"}	;case-insensitiveR
$define letterS	{"\x53\x73\x24\xA7\x32\x8A\x9A\x5A\x7A"}	;case-insensitive and non-text S
$define letterT	{"\x54\x74\x86\x2B"}	;case-insensitive and non-text T
$define letterU	{"\x55\x75\xB5\xD9\xDA\xDB\xDC\xF9\xFA\xFB\xFC\x76"}	;case-insensitive and non-text U
$define upunc	"\x7C\x5F\x7C"	;punctuation mark U
$define U	$upunc | $letterU	; case-insensitive, non-text and punctuation mark U
$define letterV	{"\x56\x76\xA5"}	;case-insensitive and non-text V
$define vpunc	{"\x5C\x2F","\x2f"}	;punctuation mark V (\/, /)
$define V	$vpunc | $letterV	;case-insensitive, non-text and punctuation mark V
$define letterW	{"\x57\x77"}	;case-insensitive and non-text W
$define wpunc	"\x5C\x2F\x5C\x2F" | "\x56\x56" | "\x76\x76" | "\x2f\x2f" | "\x7c\x2f\x5c\x7c"	;punctuation mark W (| /\ | , VV, //, vv, \/\/)
$define W	$wpunc | $letterW	;case-insensitive, non-text and punctuation mark W
$define letterX	{"\x58\x78"}	;case-insensitive and non-text X
$define xpunc	"\x3E\x3C"	;punctuation mark X (><)
$define X	$xpunc | $letterX	;case-insensitive, non-text and punctuation mark X
$define letterY	{"\x59\x79\xA5\xFF\xFD\x9F\xDD"}	;case-insensitive and non-text Y
$define letterZ	{"Zz"}	;case-insensitive Z

<table>
<tr><th colspan="3">Grammar</th></tr>
<tr><th>Macro Name</th><th>Macro Contents</th><th>Macro Description</th></tr>
<tr><td>$define capswd</td><td>WP1,{"ABCDEFGHIJKLMNOPQRSTUVWXYZ"} [3-15] ,WP1</td><td>;capitalized words</td></tr>
<tr><td>$define capswds</td><td>($capswd,@-500,$capswd,@-500,$capswd) | ($capswd,@-500,$capswd,@-500,$capswd,@-500,$capswd) | ($capswd,@-500,$capswd,@-500,$capswd,@-500,$capswd,@-500,$capswd,@-500,$capswd)</td><td>;multiple capitalized words</td></tr>
<tr><td>$define nounend</td><td>$a [0-1] , (~ "es" | ~ "er" | ~ "ers" | ~ "er's" | "" | ~ "y" | ~ "ys" | ~ "y's" | ~ "ey" | ~ "eys" | ~ "ey's" | ~ "ie" | ~ "ies" | ~ "ie's" | ~ "ing" | ~ "ings" | ~ "ing's"),WP1</td><td>;noun endings</td></tr>
<tr><td>$define verbend</td><td>$a [0-1] , (~ "s" | ~ "es" | ~ "ed" | ~ "ing" | "" | ~ "e"),WP1</td><td>;verb endings</td></tr>
<tr><td>$define wordend</td><td>$verbend | $nounend</td><td>;all noun and verb endings</td></tr>
<tr><td>$define superla</td><td>(~ "e" | ~ "r" | ~ "er" | ~ "st" | ~ "est" | ""),WP1</td><td>;adjective superlatives</td></tr>
<tr><td>$define badadj</td><td>(~ "fucking" | (~ "damn", (~ "ed" | "")) | ~ "stupid" | (~ "friggin", ("" | ~ "g" | """)) | ~ "a bunch of" | ~ "a batch of" | ~ "such" | ~ "rotten" | ~ "shitty" | (~ "shit",$punct, ~ "head", ("" | ~ "ed")) | ~ "crappy" | ~ "bastard" | (~ "shit",$punct, ~ "head", ("" | ~ "ed")) | (~ "ass",$punct, ~ "hole")),WP1</td><td>;common profane adjectives</td></tr>
<tr><td>$define preverbsoon</td><td>W1, (~ "be" | ~ "could" | ~ "would" | ~ "should" | ~ "go" | ~ "to" | ~ "will" | ~ "can")</td><td>;future-tense verb modifiers</td></tr>
<tr><td>$define preverbpres</td><td>W1, (~ "being" | ~ "is" | ~ "am" | ~ "are"),W1</td><td>;present-tense verb modifiers</td></tr>
<tr><td>$define preverbthen</td><td>W1, (~ "have" | ~ "been" | ~ "had" | ~ "was" | ~ "were"),W1</td><td>;past-tense verb modifiers</td></tr>
<tr><td>$define preverb</td><td>$preverbsoon | $preverbpres | $preverbthen</td><td>;all verb modifiers</td></tr>
<tr><td>$define mypronoun</td><td>W1, (~ "my" | ~ "your" | ~ "his" | ~ "her" | ~ "its" | ~ "our" | ~ "their"),WP1</td><td>;possessive pronouns</td></tr>
<tr><td>$define mepronoun</td><td>W1, (~ "me" | ~ "you" | ~ "him" | ~ "her" | ~ "us" | ~ "them" | ~ "a" | ~ "that" | ~ "those"),W1</td><td>;object pronouns</td></tr>
<tr><td>$define ipronoun</td><td>W1, (~ "I" | ~ "you" | ~ "he" | ~ "she" | ~ "it" | ~ "we" | ~ "they" | ~ "what"),WP1</td><td>;subject pronouns</td></tr>
<tr><td>$define pronoun</td><td>$mypronoun | $mepronoun | $ipronoun</td><td>;ultimate combination of all pronouns</td></tr>
<tr><td>$define article</td><td>WP1, (~ "the" | ~ "a" | ~ "some" | ~ "any" | ~ "all" | ~ "every" | ~ "no"),WP0</td><td>;random articles</td></tr>
</table>

$define ppphrase	WP1, (~ "on" | ~ "to" | ~ "of" | ~ "in" | ~ "into" | ~ "inside" | ~ "within" | ~ "without" | ~ "with" | ~ "after" | ~ "before" | ~ "under" | ~ "over" | ~ "atop" | ~ "around" | ~ "beneath" | ~ "beside" | ~ "as" | ~ "through"),$article	;prepositional phrases
$define gibber	(0x00-0x40), (0x5b-0x60), (0x6b-0xff)	;gibberish found in spam
$define plural	(~ "s" | "" | ~ "'s" | ~ "es")	;case-insensitive plural endings for words

Pronoun Action Endings		
Macro Name	**Macro Contents**	**Macro Description**
$define pronoun	(~ "my" | ~ "his" | ~ "her" | ~ "our" | ~ "that" | ~ "a" | ~ "those" | ~ "your" | ~ "their"),W1	;pronouns
$define action1	(~ "touch" | ~ "suck" | ~ "feel" | ~ "grab" | ~ "lick" | ~ "stroke" | ~ "spank" | ~ "fluff" | ~ "eat" | ~ "suck"),$verbend,W0	;actions
$define action2	(~ "poke" | ~ "ram" | ~ "bang" | ~ "work" | ~ "fuck" | ~ "frigging" | ~ "nail" | ~ "kiss" | ~ "fuck"),$verbend,W0	;actions
$define action3	(~ "ate" | ~ "frigged" | ~ "poking" | ~ "stroking" | ~ "felt" | ~ "look at" | ~ "rub"),$verbend,W0	;actions
$define totalact	($action1 | $action2 | $action3)	;all actions
$define ending	((~ "s",WP1) | (~ "es",WP1) | (WP1))	;endings
$define wordend	((~ "ing",WP1) | (~ "e",WP1) | (~ "ed",WP1) | (~ "er",WP1) | (~ "'s") | $ending)	;endings

Characters		
Macro Name	**Macro Contents**	**Macro Description**
$define r	{"AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz1234567890"}	;single random characters
$define randchar	{"bcdf","BCDF","ghjk","GHJK","lmnp","LMNP","qrst","QRST","vwxyz","VWXYZ"} [5-30]	;groups of random letters
$define randcharend	{"bcdf","BCDF","ghjk","GHJK","lmnp","LMNP","qrst","QRST","vwxyz", "VWXYZ"} [5-30] ,@-30,EOD	;random letters found at end of spam email
$define rand	{"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39"}	;upper/lwrcase letters and numbers
$define allrand	{"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52 \x53\x54\x55\x56\x57\x58\x59\x5a\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2A\x2B\x2C\x2D\x2E\x2F\x3A\x3B\x3C\x3D\x3E\x3F\x40\x5B\x5C\x5D\x5E\x5F\x60\x7B\x7C\x7D\x7E"}	;upper/lwrcase letters, numbers, punctuation
$define allrandsp	{"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2A\x2B\x2C\x2D\x2E\x2F\x3A\x3B\x3C\x3D\x3E\x3F\x40\x5B\x5C\x5D\x5E\x5F\x60\x7B\x7C\x7D\x7E"}	; upper/lwrcase letters, numbers, punctuation,spaces
$define anumdashper	{'a'-'z','A'-'Z','0'-'9','-','.' }	;letter number/dash/period used in spammer-spelling
$define multibang	({"!"} [1-3] ,@-1000,{"!"} [1-3] ,@-1000,{"!"} [1-3] ,@-1000,{"!"} [1-3])	;multiple exclamation points used at intervals in a spam email

<table>
<tr><th colspan="3">Contest</th></tr>
<tr><th>Macro Name</th><th>Macro Contents</th><th>Macro Description</th></tr>
<tr><td>$define lottotalk</td><td>(~ "CONGRATULATIONS" | (~ "Promotion",@-5, ~ "Program") | ~ "cash prize" | ((~ "computer" | ~ "internet-based"),@-5, ~ "ballot") | ((~ "WINNING" | ~ "award"),@-3, ~ "NOTIFICATION") | ((~ "lotto" | ~ "lottery"),@-5, ~ "international") | ~ "Yours in service," | ~ "pay-point bank" | ~ "email lottery winning" | ~ "prize winning")</td><td>;phrases common in lottery/contest scams and spams</td></tr>
<tr><td>$define contest</td><td>WP1, (~ "lotto" | ~ "lottery" | ~ "contest" | ~ "e-games" | ~ "sweepstake" | ~ "sweepstakes" | ~ "Loterij" | (~ "promotion",@-3, ~ "program")),WP1</td><td>;words/phrases used for contest spams and scams</td></tr>
<tr><td>$define ticket</td><td>(((~ "lucky" | ~ "serial" | ~ "reference" | ~ "ticket" | ~ "batch" | ~ "ref"),@-15, ~ "number") OR (~ "security code"))</td><td>;contest/lottery ticket catch phrases</td></tr>
</table>

<table>
<tr><th colspan="3">Gamble</th></tr>
<tr><th>Macro Name</th><th>Macro Contents</th><th>Macro Description</th></tr>
<tr><td>$define gamble</td><td>(~ "casino" | ~ "gamble" | ~ "gambling" | ~ "poker" | ~ W"slot machine" | ~ "lotto" | ~ "bingo" | ~ "blackjack")</td><td>;case-insensitive words relating to gambling</td></tr>
<tr><td>$define gamez</td><td>$letterG,@-3,$letterA,@-3,$letterM,@-3,$letterE,@-3,($letterS | {"Rr"},@-3,$letterS | {"Rr"})</td><td>;case-insensitive and spammer-spelling games</td></tr>
</table>

Domains		
Macro Name	**Macro Contents**	**Macro Description**
$define http	(~ "http",{"sS"} [0-1] ,"://")	;regular & secure http domains w/ spammer case tricks
$define DOM1	((~ "http" | "@" | ~ "www."), .*)	;domain name preceders with jump used before websites and email addresses
$define DOM2	(0x00-0x2d | 0x2f | 0x3a-0x3c | 0x3e-0x40 | 0x5b-0x60 | 0x7b-0xff)	;random character for domain names
$define DOM3	"\x2e", ("ae" | "ar" | "at" | "au" | "be" | "bg" | "bh" | "biz" | "br" | "c" | "ca" | "ch" | "cl" | "cn" | "co" | "com" | "cz" | "de" | "dk" | "dn" | "dp" | "edu" | "es" | "fi" | "fm" | "fr" | "gov" | "gr" | "gu" | "hk" | "hu" | "id" | "ie" | "il" | "in" | "info" | "it" | "jp" | "kr" | "kz" | "lb" | "my" | "ne" | "net" | "nl" | "no" | "nz" | "or" | "org" | "pk" | "pl" | "pt" | "ro" | "ru" | "se" | "sh" | "sk" | "tc" | "to" | "tr" | "tv" | "tw" | "ua" | "uk" | "us" | "yu" | "za"),WP1	;all domain site enders, including countries
$define url1	(~ "http://" | ~ "https://")	;regular and secure http domains
$define urlp	"\x2e\x2f\x3a" | ~ ~ " dot " | ~ ~ " slash " | ~ ~ " colon "	;punctuation preceding email address or website, along w/ words used to obfuscate email address or website
$define website	($url1,.*,$DOM3)	;random website found within spammer emails
$define randcharsite	~ "http://www.",$randchar AND ~ ".com"	;random characters preceding domain name
$define ipaddr	$DOM1,{'0'-'9'} [1-3] ,".",{'0'-'9'} [1-3] ,".",{'0'-'9'} [1-3] ,".",{'0'-'9'} [1-3]	;domain name preceders plus random IP address
$define ipaddr2	$DOM1,{'0'-'9'} [1-3] ,".",{'0'-'9'} [1-3] ,".",{'0'-'9'} [1-3] ,".",{'0'-'9'} [1-3] ,"\x3a",{'0'-'9'} [1-4]	;domain name preceders + random IP address + colon and 1-4 numbers
$define badip	("25",{'5'-'9'}) | ("2",{'6'-'9'},{'0'-'9'}) | ({'3'-'9'},{'0'-'9'},{'0'-'9'})	;fake IP used by spammers (numbers greater than 255)
$define geocities	("http://", .*, ~ "geocities.com") | ("http://", ~ "geocities.yahoo.com")	;fake geocities.com web address
$define spamgeo	($http,{'a'-'z'} [2-3] , ("." | ""),"geocities.",@-8, ("." | ""),"com/")	;fake geocities.com web address
$define spammsn	~ "http://",$allrand [2-10] , ~ "msn.com/members/",$allrand [2-30] ,"/"	;fake msn.com web address
$define spamyahoo	~ "http://",$allrand [2-10] , ~ "yahoo.com.",$a [1-2] ,"/",$allrand [2-30] ,"/"	;fake yahoo.com web address
$define purl	{"\x2e\x2f\x40"}	;punctuation preceding a url (., /, @)
$define peml	{"\x20\x3c\x3a\x5b"}	;punctuation preceding an email address (space, <, >, [ )

Email Headers

Macro Name	Macro Contents	Macro Description
$define eml	{'A'-'Z','a'-'z','0'-'9',$nspo} [3-30] ,"\x40",{'A'-'Z','a'-'z','0'-'9',$nspo} [3-30] ,$DOM3	;email address
$define reply	("\nSend reply to","\x3a\x20\x22" | "\nReply-To","\x3a\x20\x22")	;email header Reply-Tos
$define subject	"\nSubject","\x3a\x20"	;email header Subject line
$define from	"\nFrom","\x3a\x20",{"\x22"} [0-1]	;email header From line
$define headto	"\nTo","\x3a\x20",{"\x22"} [0-1]	;email header To line
$define recd	"Received",":",W1,"from","\x20"	;email header Received line
$define nuline	("\n" | "\r\n")	;email header "new line" to end contents in a given line
$define goodip	{'0'-'9'} OR ({'1'-'9'},{'0'-'9'}) OR ('1',{'0'-'9'},{'0'-'9'}) OR ('2',{'0'-'5'},{'0'-'5'})	;good IP address numbers (to be used with periods between each goodip)

Coding

Macro Name	Macro Contents	Macro Description
$define htmlcode	~ "#000080" | ~ "#ff0000" | ~ "#0000ff" | ~ "<b>" | ~ "#ff6600" | ~ "#00ff00" | ~ "#ffff00" | ~ W"COLOR: RED"	;key html code pieces
$define white	(~ "FFFFF", {0x41-0x46,0x61-0x66,0x30-0x39} [1]) | ~ ~ "color=white"	;color white for background (can be used with white font)
$define graphic	~ ".tiff" | ~ ".jpg" | ~ ".gif" | ~ ".bmp" | ~ W"Please wait for message to load, it may take a few seconds" | ~ ".jpeg" | ~ ".jpe" | ~ ".jfif" | ~ ".png"	;questionable graphics in spam emails
$define wht1	(~ "<font ",@-12, ~ "color=","\x22", ~ "White") | (~ "font ",@-12, ~ "color=","\x22", ~ "#FFFFFF")	;white font
$define red	("font ",@-12, ~ "color=","\x22", ~ "#FF0000") | ("<font ",@-12, ~ "color=","\x22", ~ "Red")	;red font
$define spamhex	("\x25" | "\x3d"), {"AaBbCcDdEeFf0123456789"}, {"AaBbCcDdEeFf0123456789"}	;hex used in spam emails

Attachments

Macro Name	Macro Contents	Macro Description
$define att	(~ "Content-Type","\x3a",@-50, (~ "file" | ~ "name"),"\x3d\x22")	;precedes attachment name
$define att2	"\nContent-Disposition"," \x3a\x20",@-20,"file name","\x3d\x22"	;precedes attachment name
$define attid	(~ "Content-ID","\x22")	;precedes attachment name

<table>
<tr><th colspan="3">People</th></tr>
<tr><th>Macro Name</th><th>Macro Contents</th><th>Macro Description</th></tr>
<tr><td>$define galsp</td><td>(~ "Posrtitute" | ~ "prostitute" | ~ "Porstitute" | ~ "housewire" | ~ "wmoen" | ~ "MILF" | ~ "Mheotr" | ~ "Mmos" | ~ "Mhoter" | ~ "Mmomy" | ~ "Weomn" | ~ "Mhoetr" | ~ "Wemon")</td><td>;mispelled females/prostitutes</td></tr>
<tr><td>$define gal1</td><td>(~ "woman" | ~ "women" | ~ "girl" | ~ "mom" | ~ "wife" | ~ "wives" | ~ "lady")</td><td>;females</td></tr>
<tr><td>$define gals</td><td>($gal1 | $galsp)</td><td>all females</td></tr>
<tr><td>$define mate</td><td>(~ "husband" | ~ "partner" | ~ "paretnr" | ~ "parnetr" | ~ "spouse" | ~ "lover" | ~ "boyfriend" | ~ "fiance")</td><td>;males</td></tr>
<tr><td>$define partner</td><td>((~ ~ " bi female", (~ "s" | WP1)) | (~ " gay ",.*, (~ "guy" | ~ "gal" | ~ "girl")) | ~ ~ "single gay")</td><td>;bi/gay person</td></tr>
<tr><td>$define female</td><td>~ "girl" | ~ "g1rl" | ~ "gurl" | ~ "g√Ærl" | ~ "woman" | ~ "female" | ~ "dame" | ~ "chick" | ~ "women" | ~ "Blondes" | ~ "Brunettes" | ~ "Babes"</td><td>;mispelled and slanf females</td></tr>
</table>

Size/Quality		
Macro Name	**Macro Contents**	**Macro Description**
$define bigger	((~ "enhanc" | ~ "enlarg"), (~ "ement" | $verbend)) \| (~ "increas",$verbend) | ((~ "larg" | ~ "big" | ~ "hug),$superla) | ~ "enormous"	;bigger references
$define quality	~ "Premium" | ~ ~ "high-quality" | ~ "best" | ~ "greatest"	;quality references
$define increase	~ "increase" | ~ "ncrease" | ~ "increasing" | ~ "increased" | ~ "enlarge" | ~ "nlarge" | ~ "bigger" | ~ "larger"	;increasing references
$define cheapsp	$letterC,@-3,$letterH,@-3,$letterE,@-3,$letterA,@-3,$letterP	;all spellings of cheap
$define cheap1	($cheapsp) | ~ "inexpensive" | ~ "low",@-5, ~ "price" | ~ ~ "less than cost price" | ~ ~ "non expensive" | ~ "bargain" | ~ "discount" | ~ "wholesale"	;low-cost references
$define cheap2	~ "deal" | ~ ~ "off the retail price" | ~ ~ "cut your expenses" | ~ ~ "less expensive" | ~ ~ "very budget-wise prices" | ~ ~ "the greatest prices" | ~ ~ "not highly priced" | (~ ~ "priced for",@1-5, ~ "workers") | ~ "goodbuy" | (~ "not",@1-10, ~ ~ "you can't afford")	;more inexpensive references
$define cheap3	~ "specials" | ~ ~ "less costly" | ~ "affordable" | ~ ~ "best values" | ~ ~ "opportune to be acquired" | ~ ~ "fantastic prices" | ~ ~ "outstanding deal" | ~ ~ "low cost" | ~ ~ "prices reasonable"	;even more inexpensive references
$define cheap	(($cheap1) \| ($cheap2) | ($cheap3))	;all inexpensive references

Credit		
Macro Name	**Macro Contents**	**Macro Description**
$define guarantee	$letterG,@-3,$letterU,@-3,$letterA,@-3,{"Rr"},@-3,$letterA,@-3,$letterN,@-3,$letterT,@-3,$letterE,@-3,$letterE	;guarantee spelling
$define credit	((~ "bad" | ~ "no" | ~ "poor" | ~ "low"),WP0, ~ "credit") OR (~ "credit",@-12, (~ "not a factor" | ~ "problems" | ~ " not an issue" | ~ " not important" | ~ " a non-issue" | ~ "will not disqualify you" | ~ "doesn't matter to us")) OR (~ "any credit accepted" | ~ "Credit history is NOT a factor")	;credit references
$define creditcard	~ W"mastercard" | ~ "visa" | ~ W"American Express" | ~ W"Gold Card" | ~ W"platinum Card" | (~ "discover",WP1) | ~ W"credit card" | ~ W"Centennial Card"	;credit cards
$define approved	$letterA,@-3,$letterP,@-3,$letterP,@-3,{"Rr"},@-3,$letterO,@-3,$letterV,@-3,$letterE,@-3,$letterD	;approved spelling
$define preapproved	$letterP,@-3,{"Rr"},@-3,$letterE,@-3,$approved	;preapproved spelling
$define bankrupt	$letterB,@-3,$letterA,@-3,$letterN,@-3,$letterK,@-3,{"Rr"},@-3,$letterU,@-3,$letterP,@-3,$letterT	;bankrupt spelling
$define creditterm	$preapproved | $approved | $bankrupt AND $guarantee	;all references to approval, bankruptcy plus guarantee
$define interest	~ W"Mortage Rate" | ~ W"low rate" | ~ W"special Rate"	;interest rate references
$define consolidate	$letterC,@-3,$letterO,@-3,$letterN,@-3,$letterS,@-3,$letterO,@-3,$letterL,@-3,$letterI,@-3,$letterD,@-3,$letterA,@-3,$letterT,@-3,$letterE	;consolidate spelling

Mortgage		
Macro Name	**Macro Contents**	**Macro Description**
$define mortgage	~ "Credit" | ~ "Home",@-25, ~ "Loan" | ~ "Mortgage" | ~ "Refinance" | ~ W"Interest Rate" | ~ "Refinancing" | ~ "Debt" | "&#106"	;references to mortgages
$define finofr	WP1, (~ "loan" | ~ "mortgage" | ~ "lending" | (~ "low",@-1, ~ "rate")),WP1	;finance offer
$define refi	(WP1,$letterR [1-3] ,WP0,$letterE [1-3] ,WP0,$letterF [1-3] ,WP0,$letterI [1-3] , WP0,$letterN [1-3] ,WP0,$letterA [1-3] ,WP0,$letterN [1-3] ,WP0,$letterC [1-3]), ($letterE [1-3] \| ($letterE [1-3] ,WP0,$letterS [1-3]) \| ($letterI [1-3] ,WP0,$letterN [1-3] , WP0,$letterG [1-3]),WP1)	;refinance/refinancing

<table>
<tr><th colspan="3">Stocks</th></tr>
<tr><th>Macro Name</th><th>Macro Contents</th><th>Macro Description</th></tr>
<tr><td>$define stocks</td><td>(~ w"seek advice from a registered" | ~ w"lose all your money" | ~ "receipt of ten thousand dollars from a third party" | ~ "paid",WP1,\d+,",", \d+,WS1, ~ "dollars")</td><td>;stock offer references</td></tr>
<tr><td>$define pennystock</td><td>($letterP,$letterE,$letterN,$letterN,$letterY,$letterS,$letterT,$letterO,$letterC,$letterK)</td><td>;pennystock spelling</td></tr>
<tr><td>$define stoktalk</td><td>((~ "CHECK OUT ",@-20, ~ "TODAY!") | ~ "Big PR Campaign" | ~ "Trade at the Top!" | ~ "get in the game" | ~ "undiscovered gem" | (~ "hot ", (~ "trading" | ~ "stock")) | ~ "highly speculative" | ~ "Get on board" | ~ "it will continue to climb" | ~ "Early Warning" | ~ "Rising Star Stock" | ~ "This Stock will rock tomorrow" | ((("G", ~ "ood") | ("H", ~ "appy")),@-3,"T", ~ "rading") | ((~ "trader" | ~ "stock"),@-3, ~ "alert") | (~ "Price Jumps ",@-5,"%") | ~ "Getting Ready To Rumble?" | ~ "solid, steady growth" | ~ "Don't miss out " | ~ "Watch this one trade " | (~ "Get in",@-3, (~ "early" | ~ "now")) | ~ ~ "this one is going to explode!" | ~ "ACT QUICK" | ~ "Watch this one go higher and higher" | ~ "Buy low, sell high!" | ~ "This Company Is A Big Winner!" | ~ "Next Week Will Be Huge!" | (~ "Get ",$CAPS [3-6] , ~ ~ "Immediately") | ~ "Make Some Fast Money" | ~ "Price Set To Jump" | ~ "on an upward trend")</td><td>;stock hype phrases</td></tr>
<tr><td>$define stokbuy</td><td>(("S", ~ "h",{"0Oo"}, ~ "rt Term") | ("S", ~ "trong ","B", ~ "uy") | ("I", ~ ~ "ndicat",{"Oo0"}, ~ "r") | ("T", ~ "icker") | ("S", ~ "ymb",{"Oo0"}, ~ "l:") | ("T", ~ "icker:"))</td><td>;stock buying phrases</td></tr>
<tr><td>$define smstox</td><td>~ "Poker Stocks" | $pennystock | (WP1, (~ ~ "small cap stocks" | ~ ~ "microcap stock" | ~ ~ "small caps" | ~ "microcaps"),WP1)</td><td>;smallstock references</td></tr>
<tr><td>$define radar</td><td>WP1, (~ "Under the Radar" | ~ "Put it on your screen now" | ~ "Radar It Right Now!" | (~ "ADD ",@-8, ~ " TO YOUR RADAR",@-10, ~ "NOW")),WP1</td><td>;references to stock/offer radar</td></tr>
<tr><td>$define validsale</td><td>(~ "offer" | ~ "price"),@-2, (~ "are" | ~ "is"), ~ ~ "valid ", (~ ~ "until" | ~ "from")</td><td>;sale offer limits</td></tr>
</table>

Scam		
Macro Name	**Macro Contents**	**Macro Description**
$define place	(~ "Cote D'lvoire" | ~ "Africa" | ~ "Zimbabwe" | ~ "Sierra Leone" | ~ "Ivory Coast" | ~ "Nigeria" | ~ "London" | ~ "Bahamas")	;scam location references
$define popwebs	$allrand [0-10] , (~ "walla" \| ~ "yahoo" \| ~ "msn" \| ~ "monster" \| ~ "sina" \| ~ "fbi" \| ~ "cia" \| ~ "cyber" \| ~ "aol" \| ~ "erols" \| ~ "test" \| ~ "att" \| ~ "every1" \| ~ "earthlink" \| ~ "hinet" \| ~ "lycos" \| ~ "comcast" \| ~ "excite" \| ~ "go" \| ~ "virgilio" \| ~ "gateway" \| ~ "lottery" \| ~ "specials" \| ~ "Promotions" \| ~ "bargain" \| ~ "marketing" \| ~ "rcn" \| ~ "knowledge" \| ~ "adelphia"),$allrand [0-10] ,$DOM3	;websites often used in spam/scam emails
$define popwebs2	$allrand [0-10] , (~ "good" \| ~ "zilla" \| ~ "mail" \| ~ "bell" \| ~ "net" \| ~ "web" \| ~ "soft" \| ~ "online" \| ~ "west" \| ~ "click2" \| ~ "gte" \| ~ "han" \| (~ "empa", ("l" \| "ss")) \| ~ "tom" \| ~ "china" \| ~ "buy" \| ~ "bigfoot" \| ~ "juno" \| ~ "daum" \| ~ "mindspring" \| ~ "prodigy" \| ~ "usa" \| ~ "wanadoo" \| ~ "mac" \| ~ " verizon" \| ~ "phreaker" \| ~ "reward" \| ~ "dsl" \| ~ "virgin" \| ~ "gmx" \| ~ "global" \| ~ "nate" \| ~ "surrealismo" \| ~ "paran" \| ~ "libero" \| ~ "flash" \| ~ "gawab" \| ~ "oicp" \| ~ "alibaba" \| ~ "site" \| ~ "fire" \| ~ "alltel" \| ~ "google" \| "free" \| "people"),$allrand [0-10] ,$DOM3	;more websites often used in spam/scam emails
$define spoofees	($popwebs) \| ($popwebs2)	;sites spoofed in spam emails
$define scamees	(~ "bankone" | ~ "southtrust" | ~ "compassweb" | ~ "Lasalle" | ~ "lasallebank" | ~ "regionsnet" | ~ "jpmorgan" | ~ "Bow" | ~ "charterone" | ~ "wammu" | ~ "wamu" | ~ "wamucorp" | ~ "wamu4u" | ~ "paypal" | ~ "ebay" | ~ "ANZ" | ~ "citizens" | ~ "citizensbanking" | ~ "citibank" | ~ "regionsbank" | ~ "regions" | ~ "bankofthewest" | ~ "thebankofthewest" | ~ "mbna" | ~ "boq" | ~ "bank-of-queensland" | ~ "central- bankonline" | ~ "keybank" | ~ "monster")	;sites used as part of scam emails
$define scamsalut	("\nDear",@-20, (~ "client" | ~ "user" | ~ "customer" | ~ "consumer" | ~ "member"),WP1)	;scam email salutations

$define scamtalk	(~ "fraud ", (~ "protection" | ~ "alert")) | ~ "anti-fraud team" | (~ "account",@-5, (~ "suspended" | ~ "suspension" | ~ "fraudulent")) | ~ "sensitive information"	;phrases used in scam emails
$define scamurl	~ "<A ,@-30, ~ "href=",$website,@-75, ~ "target=_self>",$http,@-10,$scamees,$DOM3,@-50,"<"#	;websites used in scamemails
$define scamact	((~ ~ "logon" | ~ ~ "login" | ~ "update" | ~ "confirm" | ~ "verify" | ~ "accept" | ~ ~ "re-enter" | ~ "review"),@-30, (W1, (~ "data" | ~ "Statement" | ~ "Agreement" | ~ "information" | ~ "account"),WP1))	;attempts to obtain consumer info by scammers
$define scamact2	((~ "partner" | ~ "partnership") | ("financial transaction") | (~ "investments"))	;attempts to scam through investing
$define norisk	(((~ "cost" | ~ "obligation" | ~ "hitch" | ~ "hassle" | ~ "risk"),@-1, ~ "free") OR (WP1, ~ "no",@-2, (~ "cost" | ~ "obligation" | ~ "risk" | ~ "hitch" | ~ "hassle"),WP1) OR (WP1, ~ "no ", (~ "personal" | ~ "financial"), ~ " risk",WP1) OR (~ "free from ",@-20, ~ "RISK") OR (~ "100% secure"))	;risk-free language
$define secret	(~ "secrecy" | ~ "confidential" | ~ "secret" | ~ "personal" | ~ "security")	;scam language
$define verify	(((~ "update" | ~ "verify" | ~ "secure" | ~ "confirm"),@-5, ~ "your",@-20, (~ "authenticity" | ~ "account" | ~ "information" | ~ "identity" | ~ "record")) OR (~ "account" | ~ "access"),@-10, (~ "frozen" | ~ "limited" | ~ "restricted" | ~ "closure" | ~ "suspended")) OR (~ "suspension",@-10, ~ "account"))	;attempts to obtain consumer info in scam emails
$define scamrep	~ "as",@-6, (~ ~ "next of kin" | ~ "beneficiary" | ~ "relative" | ~ "representative")	;scam email language for a representative
$define acct	~ "abandoned sum" | ~ "dormant account" | ~ "ghost account" | ~ "unclaimed account"	;unclaimed monies language
$define deth	(~ "died" | ~ "killed"),@-50, (~ "crash" | ~ "illness")	;death of relative scam language

<table>
<tr><td>$define bigbucks</td><td>((("USD" | "M" | ~ "million" | ~ "thousand" | ~ "hundred"),@-30,$currency) | (WP1, (~ "one" | ~ "two" | ~ "three" | ~ "four" | ~ "five" | ~ "six" | ~ "seven" | ~ "eight" | ~ "nine"),@-5, (~ "MILLION" | ~ "thousand" | ~ "hundred"),@-3,$currency,WP1))</td><td>;monies offered by scammer</td></tr>
</table>

<table>
<tr><th colspan="3">Medical</th></tr>
<tr><th>Macro Name</th><th>Macro Contents</th><th>Macro Description</th></tr>
<tr><td>$define weightloss</td><td>~ "Hydroxycut" | ~ "Metabolife" | ~ "Xenedrine" | ~ W"Thermo Phen Phen" | ~ "HGH " | ~ W"Human Growth Hormone"</td><td>;weightloss medications</td></tr>
<tr><td>$define erection</td><td>~ ~ "erection" | ~ "erektion" | ~ "erekshun" | (WP1,$letterE [1-3] ,WP0,$letterP [0-3] ,WP0,$letterR [1-3] ,WP0,$letterR [0-3] ,WP0,$letterE [1-3] ,WP0,$letterE [0-3] ,WP0,$letterC [1-3] ,WP0,$letterC [0-3] ,WP0,$letterT [1-3] ,WP0,$letterT [0-3] ,WP0,$letterI [1-3] ,WP0,$letterI [0-3] ,WP0,$letterO [1-3] ,WP0,$letterO [0-3] ,WP0,$letterN [1-3] ,WP0,$letterN [0-3] ,WP0)</td><td>;erection spelling</td></tr>
<tr><td>$define penis</td><td>$letterP,$punct [0-10] ,$letterE [1-6] ,$punct [0-10] ,$letterN [1-6] ,$punct [0-10] ,$letterI [1-6] ,$punct [0-10] ,$letterS [1-6] ,$punct [0-10]</td><td>;penis spelling</td></tr>
</table>

No MD		
Macro Name	**Macro Contents**	**Macro Description**
$define needed	(~ "needed" | ~ "required" | ~ "necessary")	;necessity variants
$define scriptsp	$letterP,@-3,{"Rr"},@-3,$letterE,@-3,$letterS,@-3,$letterC,@-3,{"Rr"},@-3,$letterI,@-3,$letterP,@-3,$letterT,@-3,$letterI,@-3,$letterO,@-3,$letterN	; (pre)script (ion) spelling
$define script	($scriptsp | (WP1, ~ "script",WP1) | (WP1, ~ "scrip",WP1) | ~ ~ "doctor's order")	;script/MD order
$define noappt	(~ ~ "no doctor" | ~ ~ "no appointment"),.*, $needed	;no appt/MD needed variants
$define noappt1	(WP1, ~ ~ "no doctor",@-5, ($needed | (~ "visit",{"s"} [0-1]) | (~ "appointment",{"s"} [0-1])),WP1)	;no MD needed/visit/appointment
$define noappt2	(WP1, ~ "no appointment",@-5,$needed,WP1)	;no appt needed
$define noscript	(WP1, (~ ~ "no prescriptions" | (~ "no",@-5,$script,@-5,$needed) | (~ "without a",.*$script)))	;script-free language
$define noMD	($noappt1 | $noappt2 | $noscript)	;no MD/appt/script language

<table>
<tr><th colspan="3">Drugs</th></tr>
<tr><th>Macro Name</th><th>Macro Contents</th><th>Macro Description</th></tr>
<tr><td>$define ambien</td><td>WP1, (~ ~ "ambien" | ~ "ambein"),WP1</td><td>;Ambien variants</td></tr>
<tr><td>$define vioxx</td><td>WP1, (~ "\/I ()}{}" | ~ ~ "vioxx"),WP1</td><td>;Vioxx variants</td></tr>
<tr><td>$define ritalin</td><td>{"Rr"} [1-6] ,$punct [0-10] ,$letterI [1-6] ,$punct [0-10] ,$letterT [1-6] ,$punct [0-10] ,$letterA [1-6] ,$punct [0-10] ,$letterI [1-6] ,$punct [0-10] ,$letterI [1-6] ,$punct [0-10] ,$letterN [1-6]</td><td>;Ritalin spammer-spelling</td></tr>
<tr><td>$define viagrasp</td><td>$letterV,@-3,$letterI,@-3,$letterA,@-3,$letterG,@-3,{"Rr"},@-3,$letterA</td><td>; Viagra spammer-spelling</td></tr>
<tr><td>$define viagra</td><td>~ ~ "viagra" | ~ "/iagr@" | ~ "Vagira" | ~ "Virgaa" | ~ "Viarga" | ~ "Vgiaga" | ~ "Vimaga" | ~ "iaq¬ßra" | ~ "V1AGR0"</td><td>;Viagra mispellings</td></tr>
<tr><td>$define cialissp</td><td>$letterC,@-3,$letterI,@-3,$letterA,@-3,$letterI,@-3,$letterI,@-3,$letterS</td><td>;Cialis spammer-spelling</td></tr>
<tr><td>$define cialis</td><td>($cialissp | ~ "cailis")</td><td>;Cialis mispelling</td></tr>
<tr><td>$define vicodin</td><td>$letterV,@-3,$letterI,@-3,$letterC,@-3,$letterO,@-3,$letterD,@-3,$letterI,@-3,$letterN</td><td>;Vicodin spammer spelling</td></tr>
<tr><td>$define phentermine</td><td>$letterP,@-3,$letterH,@-3,$letterE,@-3,$letterN,@-3,$letterT,@-3,$letterE,@-3{"Rr"},@-3,$letterM,@-3,$letterI,@-3,$letterN,@-3,$letterE</td><td>;Phentermine spammer-spelling</td></tr>
<tr><td>$define levitra</td><td>~ "levitra" | ~ "levitraa"</td><td>;Levitra variants</td></tr>
<tr><td>$define xanax</td><td>~ "xanax" | ~ "xanex"</td><td>;Xanax variants</td></tr>
<tr><td>$define valiumsp</td><td>$letterV,@-3,$letterA,@-3,$letterL,@-3,$letterI,@-3,$letterU,@-3,$letterM</td><td>; Valium spammer-spelling</td></tr>
<tr><td>$define valium</td><td>~ "Va | mum" | $valiumsp | $valium3</td><td>;Valium variants</td></tr>
<tr><td>$define propecia</td><td>(WP1,$letterP [1-3] ,$punct [0-1] ,$letterP [0-3] ,$punct [0-1] ,$letterR [1-3] ,$punct [0-1] ,$letterR [0-3] ,$punct [0-1] ,$letterO [1-3] ,$punct [0-1] ,$letterO [0-3] ,$punct [0-1] ,{"Pp"} [1-3] ,$punct [0-1] ,{"Pp"} [0-3] ,$punct [0-1] ,$letterE [1-3] ,$punct [0-1] ,$letterE [0-3] ,$punct [0-1] ,$letterC [1-3] ,$punct [0-1] ,$letterC [0-3] ,$punct [0-1] ,$letterI [1-3] ,$punct [0-1] ,$letterI [0-3] ,$punct [0-1] ,$letterA [0-3] ,$punct [0-1] ,$letterA [0-3] ,WP1)</td><td>;Propecia spammer-spelling</td></tr>
</table>

<table>
<tr><td>$define tblamb</td><td>(">", (("A" & ~ "mb") | ("A" & ~ "mbi") | ("A" & ~ "mbie") | ("A" & ~ "mbien") | (~ "Am"& ~ "bi") | (~ "Am" & ~ "bie") | (~ "Am" & ~ "bien") | (~ "Amb" & ~ "ie") | (~ "Amb" & ~ "ien")),"<")</td><td>;Ambien spelled out in table form</td></tr>
<tr><td>$define tblcial</td><td>(">", (("C" & ~ "ia") | ("C" & ~ "ial") | ("C" & ~ "iali") | ("C" & ~ "ialis") | (~ "Ci" & ~ "al") | (~ "Ci" & ~ "ali") | (~ "Ci" & ~ "alis") | (~ "Cia" & ~ "li") | (~ "Cia" & ~ "lis")),"<")</td><td>;Cialis spelled out in table form</td></tr>
<tr><td>$define tblviag</td><td>(">", (("V" & ~ "ia" | ("V" & ~ "iag") | ("V" & ~ "iagr") | ("V" & ~ "iagra") | (~ "Vi" & ~ "ag") | (~ "Vi" & ~ "agr") | (~ "Vi" & ~ "agra") | (~ "Via" & ~ "gr") | (~ "Via" & ~ "gra")),"<")</td><td>;Viagra spelled out in table form</td></tr>
<tr><td>$define tblsoma</td><td>(">", (("S" & ~ "om") | ("S" & ~ "oma") | (~ "So" & ~ "ma")),"<")</td><td>;Soma spelled out in table form</td></tr>
<tr><td>$define tblval</td><td>(">", (("V" & ~ "al") | ("V" & ~ "al") | ("V" & ~ "ali") | ("V" & ~ "aliu") | ("V" & ~ "alium") | (~ "Va" & ~ "li") | (~ "Va" & ~ "liu") | (~ "Va" & ~ "lium") | (~ "Val" & ~ "iu") | (~ "Val" & ~ "ium")),"<")</td><td>;Valium spelled out in table form</td></tr>
<tr><td>$define tblxan</td><td>(">", (("X" & ~ "an") | ("X", ~ "ana") | ("X", ~ "anax") | (~ "Xa", ~ "na") | (~ "Xa", ~ "nax") | (~ "Xan", ~ "ax")),"<")</td><td>;Xanax spelled out in table form</td></tr>
<tr><td>$define tblprop</td><td>(">", (("P", ~ "ro") | ("P", ~ "rop") | ("P", ~ "rope") | ("P", ~ "ropec") | ("P", ~ "ropeci") | ("P", ~ "ropecia") | (~ "Pr", ~ "op") | (~ "Pr", ~ "op") | (~ "Pr", ~ "ope") | (~ "Pr", ~ "opec") | (~ "P r", ~ "opeci") | (~ "Pr", ~ "opecia") | (~ "Pro", ~ "pe") | (~ "Pro", ~ "pec") | (~ "Pro", ~ "peci") | (~ "Pro", ~ "pecia")),"<")</td><td>;Propecia spelled out in table form</td></tr>
<tr><td>$define tbllev</td><td>(">", (("L" & ~ "ev") | ("L" & ~ "evi") | ("L" & ~ "evit") | ("L" & ~ "evitr") | ("L" & ~ "evitra") | (~ "Le" & ~ "vi") | (~ "Le" & ~ "vit") | (~ "Le" & ~ "vitr") | (~ "Le" & ~ "vitra") | (~ "Lev" & ~ "it") | (~ "Lev" & ~ "itr") | (~ "Lev" & ~ "itra")),"<")</td><td>;Levitra spelled out in table form</td></tr>
<tr><td>$define tblmer</td><td>(">", (("M" & ~ "er") | ("M" & ~ "eri") | ("M" & ~ "erid") | ("M" & ~ "eridi") | ("M" & ~ "eridia") | (~ "Me" & ~ "ri") | (~ "Me" & ~ "rid") | (~ "Me" & ~ "ridi") | (~ "Me" & ~ "ridia") | (~ "Mer" & ~ "id") | (~ "Mer" & ~ "idi") | (~ "Mer" & ~ "idia")),"<")</td><td>;Meridia spelled out in table form</td></tr>
</table>

<table>
<tr><td>$define tblmedz</td><td>$tblamb | $tblcial | $tblviag | $tblsoma | $tblval | $tblxan | $tblprop | $tbllev</td><td>;ambien/cialis/viagra/soma/valium /xanax/propocia/levitra in table form</td></tr>
</table>

<table>
<tr><th colspan="3">Meds</th></tr>
<tr><th>Macro Name</th><th>Macro Contents</th><th>Macro Description</th></tr>
<tr><td>$define sexmed</td><td>$viagra | ~ ~ "HGH " | ~ W"Human Growth Hormone" | ~ "Blue pill" | ~ "Xenical" | $phentermine | ~ "Somatotropin" | ~ "vprx"</td><td>; refers to sex-related medications</td></tr>
<tr><td>$define medicine</td><td>($sexmed) | ($weightloss) | ~ "v.aIium" | ~ "valium" | ~ "xanax" | ~ "xana | x"</td><td>;other meds</td></tr>
<tr><td>$define medsp</td><td>$letterM,@-3,$letterE,@-3,$letterD,@-3,$letterI,@-3,$letterC,@-3,$letterA,@-3,$letterT,@-3,$letterI,@-3,$letterO,@-3,$letterN</td><td>;medication spammer-spelling</td></tr>
<tr><td>$define medz</td><td>$letterM,@-3,$letterE,@-3,$letterD,@-3,$letterS</td><td>;meds spammer-spelling</td></tr>
<tr><td>$define meds</td><td>(($medsp) | ($medz | $pillz | $drugz | $steroids | ~ "tablets") | (~ "remedies" | ~ "medicine" | ~ ~ "alleviating item"))</td><td>;spam medication references</td></tr>
<tr><td>$define pillz</td><td>(WP1,$letterP [1-3] ,WP0,$letterI [1-3] ,WP0,$letterL [1-3] ,WP0,$letterL [1-3] , WP0, ($letterS | $letterZ),WP1)</td><td>;pills spammer-spelling</td></tr>
<tr><td>$define drugz</td><td>$letterD,@-3,{"Rr"},@-3,$letterU,@-3,$letterG,@-3,$letterS</td><td>;drugs spammer- spelling</td></tr>
<tr><td>$define steroids</td><td>$letterS,@-3,$letterT,@-3,$letterE,@-3,{"Rr"},@-3,$letterO,@-3,$letterI,@-3,$letterD,@-3,$letterS</td><td>;steroids spammer-spelling</td></tr>
<tr><td>$define spammeds</td><td>WP1, (~ "Phentermine" | ~ "Celebrex" | ~ "vicodin" | ~ "cialis" | ~ "ambien" | ~ "levitra" | ~ "xanax" | ~ "ultram" | ~ "viagra" | ~ "valium" | ~ "soma" | ~ "propecia" | ~ "meridia" | ~ "zyban" | ~ "Diazepam" | ~ "Xenical" | " HGH " | ~ "Paracodin" | ~ "Phentermine" | ~ "Human Growth Hormone")</td><td>;meds used in spam emails</td></tr>
<tr><td>$define edmed</td><td>($cialis | $viagra | $levitra | $propecia | (WP1, (~ "love" | ~ "sex"),@-3,$pillz))</td><td>;sex-related pill references</td></tr>
<tr><td>$define edmeds</td><td>($edmed,@-500,$edmed,@-500,$edmed) | ($edmed,@-500,$edmed,@-500,$edmed,@-500,$edmed) | ($edmed,@-500,$edmed,@-500,$edmed,@-500,$edmed,@-500,$edmed)</td><td>;reference to multiple sex meds in spam email</td></tr>
</table>

<table>
<tr><th colspan="3">Offers</th></tr>
<tr><th>Macro Name</th><th>Macro Contents</th><th>Macro Description</th></tr>
<tr><td>$define satis</td><td>(~ "100% Customer Satisfaction" | ~ "satisfaction guaranteed" | ~ "you will be satisfied")</td><td>;satisfaction language</td></tr>
<tr><td>$define BBB</td><td>(~ "VERIFIED BY ","BBB",@-25, ~ "APPROVED BY ","VISA") | (~ "official",@-3,"BBB",@-3, ~ "verification",@-25,"VISA",@-3, ~ "approval")</td><td>;Better Business Bureau references</td></tr>
<tr><td>$define salztalk</td><td>(WP1, (~ "NO PURCHASE NECESSARY" | ((~ "today" | ~ "this week"),W1, ~ "only") | ~ "Get In Early" | ("F", ~ "ree ",@-3,"G", ~ "ift!") | (~ "Try it ","FREE") | (~ ~ "If you don't ",@-15, ~ ~ "love it, send it back") | (~ "Save up to",@-3,$d [1-4] ,@-3,"%") | (~ "limited ", (~ "quantities available" | ~ "time offer")) | ((~ "act" | ~ "save" | ~ "order" | ~ "buy"),@-3, (~ "now" | ~ "here")) | ((~ "this sale" | ~ "these prices"), ~ " won't last") | ~ "Restrictions may apply" | ~ "Offer cannot be combined with other " | ((~ "This week" | ~ "today"), ~ " only") | ~ ~ "valuable coupons" | ((~ "incredible" | ~ "great" | ~ "special" | ~ "spectacular"),@-3, (~ "discounts" | ~ "prices" | ~ "deals" | ~ "offers" | ~ "purchases" | ~ "savings" | ~ "buys")) | $validsale | ~ "when these deals are gone, they are gone!"),WP1)</td><td>;sales language</td></tr>
<tr><td>$define sales</td><td>;sales language</td><td>(~ "Save up to ",$d [1-4] ,"%") | ~ "Buy now" | ~ "limited time offer" | ~ "limited quantities available" | ~ "act now" | ~ "this sale won't last" | ~ "special price"</td></tr>
<tr><td>$define onsale</td><td>(~ ~ "This week only" | ~ ~ "View Online Store" | ~ ~ "CLlCK HERE to view brochure" | ~ ~ "M0RE DETA1LS HERE" | ~ ~ "Save Now" | ~ ~ "spectacular savings" | ~ ~ "special offers" | ~ ~ "valuable coupons" | ~ "great deals" | (~ ~ "Offer valid from",@-15, ~ "to",@-15,WP1) | (~ ~ "Click Here For Details"))</td><td>;more sales language</td></tr>
<tr><td>$define foru</td><td>("4" | ~ "for"),WP0, ($letterU | ~ "you")</td><td>;4/for you</td></tr>
</table>

$define free	{"Ff"} [1-6] ,$punct [0-10] ,{"Rr"} [1-6] ,$punct [0-10] ,$letterE [1-6] ,$punct [0-10] , ($letterE [0-1] | $letterE [1-6]),$punct	;free spammer-spelling
$define watches	(~ "wristwatch" | ~ ~ "watch wear" | ~ "wrist" | ~ "watches" | ~ "timepiece" | ~ "timekeeper")	;watches variants

Diploma		
Macro Name	**Macro Contents**	**Macro Description**
$define university	~ "universi-y" | ~ "university" | ~ ~ "college"	;university/college
$define degree	~ "d-gree" | ~ "degree" | (~ "d",{"Ee3"},"gr",{"Ee3"} [2-3])	;degree spammer-spelling
$define diploma	~ ~ "diploma" | ~ "dlpL ()ma" | (~ "d",$letterI, ~ "p",$letterL,$O, ~ "m",$letterA)	;diploma variants
$define sheepskin	("B.S.",@-10,"M.S.",@-10,"Ph.D.") OR (~ "Bachelor",@-10, ~ "Master",@-10, ~ "Doctorate")	;B.S./M.S./PhD variants

Purchase		
Macro Name	**Macro Contents**	**Macro Description**
$define shipment	$letterS,@-3,$letterH,@-3,$letterI,@-3,$letterP,@-3,$letterM,@-3,$letterE,@-3,$letterN,@-3,$letterT	; shipment spammer-spelling
$define shipping	($shipment | ((~ ~ "first-class" | ~ "Priority" | ~ "rush" | ~ "free" | ~ "overnight" | ~ "express" | ~ "international" | ~ "overseas" | ~ "worldwide"),WP1, (~ "courier" | ~ "delivery" | ~ "shipp ing" | ~ "Mail")) | ((~ "ship" | ~ "deliver"),@-4, (~ "overseas" | ~ "internationally" | ~ "overnight" | ~ "express" | ~ "worldwide")) | ~ "trans port service")	;shipping language
$define order	~ "order | ~ "purchase" | ~ "buy" | ~ "obtain" | ~ "get"	;order variants
$define online	(WP1, (~ ~ "on the net" | ~ ~ "on the web" | ~ "internet" | ~ "worldweb" | ~ "web" | ~ "net"), (WP1 | (~ "s" | ~ "s"),WP1))	;online variants
$define tracking	~ ~ "online tracking" | (~ "track",@1-25, ~ "online") | ~ ~ "cyber tracking" | ~ ~ "real-time tracking"	; tracking variants

Pornogr		
Macro Name	**Macro Contents**	**Macro Description**
$define adultsp	$letterA,@-3,$letterD,@-3,$letterU,@-3,$letterL,@-3,$letterT	;adult spammer-spelling
$define adult	$adultsp | ~ "Adiolt"	;adult variants
$define fbomb	(({"Ff"}, {"WwUu*-_"}, {"Cc*-"}, {"Kk"}) | (WP1, ~ ~ "fuk "))	;f-word variants
$define cbomb	~ "c*ck" | ~ "c-ck" | ~ "cock" | ~ "c_ck" | ~ "c0ck" | ~ "kunt"	;c-word variants
$define pornline	~ "teens" | ~ "girls" | ~ "adults" | ~ "adu1ts" | (WP1, ~ "xxx",WP1) | ~ "celebrities" | ~ "sex " | ~ "porn" | ~ "p0rn" | ~ "p@rn" | ~ "t√®√®n" | ~ "f√∫ck" | ~ "p√Æcs" | ~ "GlRLs" | ~ "chicks" | ~ "Virg1n" | ~ "vag1nas" | ~ "peek1ng" | ~ "Hardc0re" | ~ "bl0nde" | ~ "ppoorrnn" | ~ "lesbo" | ~ "lesb0"	;porn-related words and spammer-spelling
$define pornline2	~ "amateur" | ~ "lesbian" | ~ "hardcore" | ~ "anal " | ~ "mature" | ~ "interracial" | "seex"	;porn-related words
$define eighteen	(~ "eighteen" | ~ "eighteen" | "18")	;18 variants
$define eighteenold	(~ ~ "MUST BE ATLEAST 18 YEARS OLD") | (~ W"Adults Only", WP1,$eighteen, WP1, ~ W"or older") | (~ W"at least", WP1, $eighteen, WP1, ~ W"years of age") | ((~ w"at least" | ~ W"must be"), WP1, $eighteen, WP1, (~ W"years of age" | ~ W"years old" | ~ W"or older")) | (~ W"You must be over", WP1, $eighteen) | (~ W"18 years or older") | (~ W"YOU MUST BE AT LEAST", WP1, $eighteen, WP1, ~ W"TO ENTER") | ("18+")	;18 years old variants
$define adultwarning	$eighteen OR $eighteenold	;all 18 years old variants
$define cheatwife1	(~ "wife" OR ~ "wives" OR ~ "women" OR ~ "woman"),.*, (~ "cheat")	;cheat language
$define cheatwife2	(~ "cheat"),.*, (~ "wife" OR ~ "wives" OR ~ "women" OR ~ "woman")	;cheating wife language
$define cheatwife	$cheatwife1 OR $cheatwife2	;all cheating wife language
$define pornadj	~ "horny" | (~ "teen",$nspo [0-1] , ~ "age") | ~ "teen" | ~ "underage" | ~ "barely legal" | ~ "virgin" | ~ "hot" | ~ "wet" | ~ "dripping" | ~ "willing"	; common porn adjectives

Click/Link		
Macro Name	**Macro Contents**	**Macro Description**
$define clickwords	(~ "Click" OR ~ "CLIKK" OR ~ "CLCIK" OR ~ "C1ick" OR ~ "cli ~ ick" OR ~ "ciilck" OR "Clicking Here")	;phrases for click here
$define click	($clickwords OR ~ ~ "Unsubscribe" OR ~ ~ "Optin" OR ~ ~ "Optout" OR ~ ~ "0ptout" OR ~ ~ "opt0ut" OR ~ ~ "0pt0ut" OR ~ "0pt" OR ~ ~ "press")	;phrases for click here or opt out
$define contact	($clickwords OR ~ W"more info" OR (~ "here", WP0, (~ "A>" | ~ W"A HREF" | ~ "http://")) OR ~ W"fill out" OR ~ W"please send")	;phrases for click here or opt out or request info
$define optin	(~ "click",@-30, ~ ~ "opt-out") OR (~ ~ "Opt-out instructions") OR (~ W"Member of the opt-in America List") OR (~ W"to opt out", @-100, ~ W"click here") OR (~ W"remove yourself from our opt in list") OR (~ ~ "removehtml") OR ($website,.*,"/optout.php",WP1) OR ($website,.*,"/accept",WP1) OR ($website,.*,"/confirm",WP1)	;opt out or opt in language
$define optin2	(~ W"To remove yourself from this opt-in mailing") OR (~ ~ "su.bs.cript.ionop.tio.ns")	;subscription options
$define link	~ "click ", (~ "the link" | ~ "here" | ~ "below" | ~ "on this link" | ~ "" | ~ "" | ~ "" | ~ "" | ~ "" | ~ "" | ~ "") AND $website	;linkage language
$define visit	~ "visit ", (~ "this link" | ~ "here" | ~ "us" | (~ "our ",@-5, ~ "site")) AND $website	;visit website language
$define vslink	((~ "visit ", (~ "this ", (~ "link" | ~ ~ "website")) | ~ "here" | ~ "us" | (~ "our ",@-5, ~ "site")) | ~ "use the link below")	; visit link language
$define clklink	((~ "click ", (~ "the link" | ~ "here" | ~ "below" | ~ "on this link")) | ~ "Follow this link")	;click link language
$define clkhere	(($letterC [1-2] ,$punct, {$letterL,$letterI} [1-2] ,$punct [0-2] ,{$letterL,$letterI} [1-2] ,$punct [0-2] ,$letterC [1-2] ,$punct [0-2] ,$letterK [1-2] ,WP1,$letterH [1-2] ,$punct [0-2] ,$letterE [1-2] ,@-3,$letterR [1-2] ,$punct [0-2] ,$letterE [1-2]) | ~ "Cilck Here")	;click here spammer-spelling
$define clklink2	((~ "follow" | ~ "click" | ~ "visit"),@-5, ~ "the link below")	; visit link language

$define clk	(~ "GO HERE:" | $clklink2 | $clklink | $vslink | ~ "check this out" | ~ "one click away" | ((~ "see" | ~ "use" | ~ "click on"), ~ " the link below") | $clkhere | ~ "View Online Store" | ~ ~ "M0RE DETA1LS HERE" | (~ "Click Here ", (~ "For Details" | ~ "to view")) | ~ "a few simple clicks away")	;click /visit link language
$define log	(~ "login" | ~ "logon")	;logon language
$define scamlink	(W1, (~ "click" | ~ "activate" | ~ "follow" | ~ "visit"),@-20, (~ "link" | ~ "here"),WP1)	;follow link language

Legal		
Macro Name	**Macro Contents**	**Macro Description**
$define legal1	(~ "bill", @-30, ~ "section 301") OR (~ ~ "Bill s.1618 Title III passed by the 105th U. S. Congress")	;legal language about spam
$define legal2	(~ "offer",.*, ~ "void",.*, ~ "prohibited",.*, ~ "law") OR (~ W"email complies with all regulations and is not spam")	;legal language about spam

Techno		
Macro Name	**Macro Contents**	**Macro Description**
$define microsoft	$letterM,@-3,$letterI,@-3,$letterC,@-3,{"Rr"},@-3,$letterO,@-3,$letterS,@-3,$letterO,@-3,{"Ff"},@-3,$letterT	;microsoft spammer-spelling
$define software	WP1, (("E", ~ "nterprise") | (~ "S", ~ "tudio") | ("F", ~ "irewall") | ("A", ~ "crobat") | ("A", ~ "dobe") | ($microsoft) | ("O", ~ ~ "ffice XP") | ("M", ~ "acromedia") | ("C", ~ "orel") | ("W", ~ ~ "indows XP") | ("D", ~ "reamweaver") | ("P", ~ ~ "icture It") | ("P", ~ "hoto",@-1,"P", ~ "ainter") | "Excel" | ("P", ~ "hotoshop") | ("I", ~ "nDesign") | ("P", ~ "remier") | ("I", ~ "llustrator") | ("W", ~ "indows") | ("M", ~ "icrosoft",@-1,"Office") | ("Office",@-10,"P", ~ "rofessional")),WP1)	;software for sale by spam
$define software2	((~ "Dreamweaver" | ~ "Picture It" | ~ "PhotoPainter" | ~ "Excel" | ~ "Photoshop" | ~ ~ "InDesign" | ~ ~ "Premier" | ~ "Illustrator" | ~ "Windows" | ~ "Office"),@-50, (~ "Dreamweaver" | ~ "Picture It" | ~ "PhotoPainter" | ~ "Excel" | ~ "Photoshop" | ~ ~ "InDesign" | ~ ~ "Premier" | ~ "Illustrator" | ~ "Windows" | ~ "Office"))	;software for sale by spam
$define softwares	($software,@-500,$software,@-500,$software,@-500,$software,@-500,$software,@-500,$software,@-500,$software,@-500,$software) | ($software,@-500,$software,@-500,$software,@-500,$software,@-500,$software@-500,$software,@-500,$software,@-500,$software,@-500,$software,@-500,$software) | ($software,@-500,$software,@-500,$software,@-500,$software,@-500,$software,@-500,$software,@-500,$software,@-500,$software,@-500,$software,@-500,$software,@-500,$software,@-500,$software,@-500,$software)	;multiple software products for sale by spam
$define softmakers	(~ "Macromedia" | ~ "Corel" | ~ "Linux" | "Adobe" | ~ "Microsoft" | ~ "Macintosh" | (~ "Mac",@-6, ~ "PC") | (~ "PC",@-6, ~ "Mac"))	;softwaremakers

$define softmakers2	(~ "Macromedia" | ~ "Corel" | ~ "Linux" | "Adobe" | ~ "Microsoft" | ~ "Macintosh" | (~ "Mac",@-6, ~ "PC") | (~ "PC",@-6, ~ "Mac")),@-50, (~ "Macromedia" | ~ "Corel" | ~ "Linux" | "Adobe" | ~ "Microsoft" | ~ "Macintosh" | (~ "Mac",@-6, ~ "PC") | (~ "PC",@-6, ~ "Mac"))	;softwaremakers
$define dvds	$letterD,@-3,$letterV,@-3,$letterD,@-3,$letterS	;DVD spammer-spelling

Tables		
Macro Name	**Macro Contents**	**Macro Description**
$define etbl	("\x3c\x2f", ~ "td","\x3e"),@-25, ("\x3c\x2f", ~ "tr","\x3e"),@-25, ("\x3c\x2f", ~ "table","\x3e")	;spam table formatting
$define tblstf	("\x3c", ~ "strong",\x3e") | ("\x3c", ~ "bold",\x3e") | ("\x3c\x2f", ~ "strong",\x3e") | ("\x3c\x2f", ~ "bold",\x3e")	;table formatting with bold/strong
$define tbldiv	("\x3c", ~ "div",\x3e") AND ("\x3c \x2f", ~ "div",\x3e")	;spam table with div
$define tbl	("\x3c", ~ "table" & "\x3e") AND ("\x3c\x2f", ~ "table" & "\x3e")	;spam table formatting
$define tblbrk	(W0, ("</DIV> <DIV>" | "</DIV> </TD>" | "<TD> <DIV>"),W0)	;spam table formatting with breaks

Unsubscr (Unsubscribe Language)

Macro Name	Macro Contents	Macro Description
$define unsub	(NOT ~ "unsubscribe") AND ($upunc | $letterU [1-6]),$punct [0-10] ,$letterN [1-6] ,$punct [0-10] ,$letterS [1-6] ,$punct [0-10] ,$letterU,$punct [0-10] ,$letterB [1-6] ,$punct [0-10] ,$letterS [1-6] ,$punct [0-10] ,$letterC [1-6] ,$punct [0-10] ,$letterR [1-6] ,$punct [0-10] ,$letterI [1-6] ,$punct [0-10] ,$letterB,$punct [0-10] ,$letterE	Unsubscribe Language
$define unsubscribe	(~ ~ "Unsubscribe",@-100, ~ ~ "Click") OR (~ ~ "Recurring",@-15, ~ "mail") OR (~ "Unsolicited",W1, ~ ~ "email") OR (~ W"to be taken 0ff our list") OR (~ W"no longer wish to receive our offers please Click Here For Deletion") OR (~ W"Stop Sending me this click here") OR (~ ~ "OPT-OUT from this mailing") OR (~ W"prefer to not receive future email messages") OR (~ W"avoid receiving additional advertisements from us") OR (~ W"If you received this letter by mistake, please click here") OR (~ W"Reply with off and I won't write you again") OR (~ W"To be removed send your remove request to") OR (~ W"If you dont wish to recieve any",WP0, ~ ~ "email") OR (~ W"avoid receiving this again, e- mail",.*, ~ "takemeoffthislist.com") OR (~ W"If you would not like to receive future offers") OR (~ W"If you prefer not to receive future mailings")	Unsubscribe Language

$define remove1	(~ W"removed from receiving future") OR (~ W"Remove Yourself Permanently")OR (~ W"Removed from this list") OR (~ W"to stop all future offers") OR (~ W"If you would like to be removed from mailings concerning weight loss") OR (~ W"to be removed from future mailings") OR (~ W"Replying to this message will not unsubscribe you") OR (~ W"Would you like to never hear from us again? Go here") OR (~ W"If you no longer wish to receive quality offers like this from us") OR (~ ~ "To unsubscribe, go to") OR (~ W"To unsubscribe from receiving further email") OR (~ W"use this link if you would like to be taken off this list") OR (~ W"To be removed from our list, please send an email to") OR (~ W"If you wish to receive no further such advertisements, please go here") OR (~ W"Sorry if this email caused you inconvenience",@-30, ~ W"to stop me sending you more please go here") OR (~ W"To be removed from all our future corporate mailings please click below") OR (~ ~ "takemeoffplease",W0,"")	Unsubscribe Language

$define remove2	(~ ~ "email",@-40, ~ ~ "remove",@-40, ~ ~ "subject line") OR (~ "terminate",@-7, ~ W"future offers") OR (~ W"To terminate future offers, please visit") OR (~ W"Use this link to be removed from this list") OR (~ ~ "FOR Removal mail to") OR (~ W"To unsubscribe send a blank email") OR (~ W"Please use this link to be taken off our list") OR (~ W"not responsible fOR any unwanted email sent to you using this software") OR (~ W"To unsubscribe from our list") OR (~ W"unsubscribe now by going to") OR (~ W"prefer not to receive these messages in the future") OR (~ W"no longer like to receive special promotions") OR (~ W"use this link to be taken off this list") OR (~ W"To be removed, please reply back with REMOVE") OR (~ "Not4me") OR (~ W"If you would rather not receive",@-30, ~ W"messages such as this one") OR (~ W"CLICK HERE For Instant Removal") OR (~ W"unsubscribe yourself off this list here") OR (~ W"send a removal request by clicking below") OR (~ W"to be removed:", WP0, (~ W"" | ~ "WWW." | ~ "HTTP"))	Unsubscribe Language

$define remove3	(~ W"removed from this mailing list") OR (~ W"removed from further mailing") OR (~ ~ "remove",@-30, ~ ~ "opt out") OR (~ W"hope you enjoyed receiving this message, but if you no longer wish to receive them, use this") OR (~ W"Want to be removed from our email list") OR (~ W"to be taken off go here") OR (~ ~ "OPT-OUT from this mailing") OR (~ W"Click Here if you no longer wish to receive email") OR (~ W"Follow this link to unsubscribe") OR (~ W"If you wish not to receive any email from us") OR (~ W"Click here to be removed") OR (~ W"If you do not wish to receive any further messages") OR (~ W"If you do not want to receive special offers in the future") OR (~ "takeMeoff") OR (~ "To say goodbye, please click here") OR (~ W"If you would no longer like us to contact") OR (~ W"Click Below to be removed fom our mailing systems") OR (~ "Click Below to be removed from our exclusive mailing opt-in systems") OR (~ W"RemovalLink") OR ((~ W"to change your email status" | ~ W"to change your e-mail status"), W0, ~ W"<A HREF")	Unsubscribe Language

$define remove4	(~ W"remove in the subject and you will be removed off our list") OR (~ ~ "Don't want to receive this email anymORe",@-30, ~ ~ "click") OR (~ W"to be removed from future mailings please reply",@-100, ~ W"with remove in the subject") OR (~ ~ "click",@-30, (~ ~ "unsubscribe" | ~ W"unsubscribe")) OR (~ ~ "To be taken off our database. Please click below") OR (~ W"Take Me OFF Your List Please allow 48 hours fOR processing") OR (~ W"you would like to remove yourself from future mailings") OR (~ W"If you no longer wish to receive our offers and updates") OR (~ W"to be removed", @-200, ~ W"click here") OR (~ W"no longer wish to receive such messages") OR (~ W"To unsubscribe from our mailing list") OR (~ W"withdraw from future offers by following the unsubscribe link") OR (~ W"You may withdraw from future offers by following the unsubscribe link") OR (~ W"if you would not like to receive future mailings") OR (~ W"if you wish to unsubscribe") OR (~ W"If you'd rather not hear from us") OR (~ W"I will take you off the list")	Unsubscribe Language

$define remove5	(~ ~ "unsubscribe",@-30, ~ ~ "send an email to") OR (~ W"To get out from future notifications, send any email here") OR (~ W"prefer not to receive future marketing messages from us, Click here") OR (~ W"To discontinue the receipt of emails, visit the following link") OR (~ W"Remove From List click here") OR (~ W"To Unsubcribe from our mailing list") OR (~ W"reply to this email and you will be removed from our data base") OR (~ W"take yourself out of the database permanently by choosing this link") OR (~ W"to unsubscribe reply to this email with unsubscribe in the subject line") OR (~ W"This is a one time mailing, you will not recieve mORe email from us") OR (~ W"To be off from future offers") OR (~ W"Delist Me Now") OR (~ W"click To Get off the mail list") OR (~ W"to stop receiving", @-25, (~ W"offers" | ~ W"mailings")) OR (~ W"To UnSubscribe yourself from this mailing list") OR (~ W"be removed from future email announcements")	Unsubscribe Language

$define remove6	(~ W"To change your subscription details OR stop receiving email") OR (~ W"FOR Removal mail to") OR (~ ~ "To be rem0ved, click here") OR (~ ~ "To be removed, click here") OR (~ W"Remove yourself from this recurring list") OR (~ W"if you prefer to be deleted from our database") OR (~ ~ "stop receiving emails, click here") OR (~ W"Click here to be removed from mailing list") OR (~ ~ "promotion",@-100, ~ W"please click here to be removed") OR (~ W"Remove me from receiving future email") OR (~ W"remove yourself from this and related email lists") OR (~ W"If you would prefer not to receive future marketing messages, click here") OR (~ W"Push Now If you aren't interested in this simply respond to my email") OR (~ W"dont want me to email you any mORe? go here") OR (~ W"To avoid receiving this again") OR (~ W"If you no longer wish to be notified") OR (~ W"If you do not wish to receive future") OR (~ W"Feel free to", @-30, ~ W"discontinue", @-30, ~ W"this service") OR (~ W"To be deleted from our member database")	Unsubscribe Language

$define remove7	(~ W"click here if not interested") OR (~ W"To Remove from our mailing") OR (~ W"immediately withdrawn from all future mailings") OR (~ ~ "SORry",@-20, ~ ~ "Send",@-30, ~ ~ "mail",@-30, ~ ~ "click",@-30, ~ ~ "reject" | @-30, ~ ~ "remove" | @-30, ~ ~ "unsubscribe" | @-30, ~ ~ "opt") OR (~ W"Please use this link if you would like to be taken off of future mailings") OR (~ W"come here to be taken off") OR (~ W"visit here to unsubscribe") OR (~ W"To be removed from our lists, click") OR (~ W"our apologies if you have been sent this email in errOR") OR (~ W"If you do not wish to receive offers") OR (~ W"Don't want any mORe adverts? Simply click here") OR (~ W"T0 BE REM0VED FR0M") OR (~ W"To never receive an ad like this, visit this link") OR (~ "TAKE ME 0FF") OR (~ W"If you no longer wish to be member and receive our emails") OR (~ W"If you do not wish to receive any further communications") OR (~ W"To get off of this campaign") OR (~ W"Click to be removed") OR (~ W"FOR unsubscribe reply this email with subject")	Unsubscribe Language

$define remove8	(~ W"to unsubsrcibe") OR (~ W"unsubscribe from mailing list") OR (~ W"Connect here if you wish to unsubscribe") OR (~ W"follow the unsubscribe instructions below") OR (~ W"you can unsubscribe by sending a blank email") OR (~ W"If you no longer wish to receive our offers and specials click here") OR (~ W"To stop receiving targeted mailings") OR (~ ~ "This is Unsolicited E-mail") OR (~ W"To no longer receive click this") OR (~ W"To avoid receiving similar offers in the future") OR (~ W"you can unsubscribe by going here") OR (~ W"one time mailing so no need to unsubscribe") OR (~ W"please see below fOR unsubscribe infORmation") OR (~ W"If you do not wish to receive future mailings from us, please go to the link below") OR (~ W"To opt out of our contact database click here") OR (~ W"If you don't want any mORe emails from us") OR (~ W"would like to be removed from any further mailings") OR (~ W"to stop receiving these offers") OR (~ W"To discontinue these offers", @-100, ~ "here") ; unsubscribe language	Unsubscribe Language

$define remove9	(~ W"If you don't want this type infORmation OR e-mail") OR (~ W"if you no longer wish to hear about future offers from us") OR (~ W"Want to stop future messages from us") OR (~ W"To be taken off our list within the next 48 hours please click below") OR (~ W"To be removed write back with remove in subject line") OR (~ ~ "To Be Removed Click Below") OR (~ W"To purge your email address from our database") OR (~ W"To never get this mail again, take preventive action") OR (~ W"If you would rather not receive these messages") OR (~ W"To prevent further contact please use the following link") OR (~ W"If you do not wish to receive these mailings anymORe") OR (~ W"If you would like to be unsubcribed from this newsletter") OR (~ W"Please click below to remove yourself from the list.") OR (~ W"If you would prefer not to receive further emails") OR (~ W"If you would rather not receive emails") OR (~ W"To remove your email from our service") OR (~ W"If you wish to be unsubscribed from future mailings")	Unsubscribe Language

$define remove10	(~ W"If you do not wish to receive special offers") OR (~ W"If you no longer wish to receive infORmation from me") OR (~ W"To be exluded from future mailings") OR (~ W"if you", WP1, (~ "wish" | ~ W"would like"), WP1, ~ W"to remove yourself") OR (~ W"to unsubscribe", @-30, ~ W"point your browser") OR (~ W"To be REM0VED", @-100, ~ W"CLlCK HERE") OR (~ W"If you don't want to receive this mail any mORe") OR (~ W"N0 mail ple@se") OR (~ W"Please see our website FOR removal Instructions") OR (~ ~ "No Thanks I am not interested", @-30, ~ W"<A HREF") OR (~ W"To stop n otifications click here") OR (~ ~ "unsuscribe",@-30, ~ "click") OR (~ W"click here to stop receiving these messages") OR (~ W"Use the link below to be permanently withdrawn from all future mailings") OR (~ W"If you do not with to receive",@-30, ~ ~ "unsubscribe") OR (~ W"to discontinue receiving messages from us") OR (~ W"To receive no mORe mail") OR (~ W"If you would no longer like to receive special") OR (~ "<A HREF=", @-75, ~ W"no mORe of this")	Unsubscribe Language

$define remove11	(~ W"Rem0ve bel0w:") OR (~ W"N0t interested:") OR (~ W"feel free to cancel your subscription") OR (~ W"Click here to unjoin from this list") OR (~ W"Click here to stop these notifications") OR (~ W"I D0N'T W@NT T0 RECIEVE THIS @NYMORE:") OR (~ W"unsubscribe here", @-3, ~ "</A>") OR (~ W"care to discontinue further sending of all our emails") OR (~ W"TO GET NO FURTHER OFFERS CLICK HERE") OR (~ W"remove me", @-10, ~ "</A>") OR (~ W"REMOVE FROM MAILLIST", @-10, ~ "</A>") OR (~ W"To be taken off please go to") OR (~ W"If you do not wish to receive any future email") OR (~ W"Don't want to receive this email anymORe") OR (~ W"No mORe mail to your email") OR (~ W"If you do not wish to receive any further") OR (~ W"If you do not want to receive these") OR (~ ~ "HERE TO STOP RECEIVN") OR (~ W"To opt out of future e-mailings") OR (~ W"To stop recieving") OR (~ W"To be rern0ved") OR (~ W"To be removed from this mailing") OR (~ "GoHereToBeDiscarded") OR (~ "Turn notifications off")	Unsubscribe Language

$define remove12	(~ W"to stop emails", @-15, ~ ~ "goto link") OR (~ W"To stop receiving these emails click here") OR (~ W"fOR r-moval instructions") OR (~ W"This is a one time mailing only") OR (~ W"To Remove from future offers") OR (~ W"To be taken-off from all further mailings") OR (~ W"No Thanks, take me off your list") OR (~ W"If you would no longer like to receive these offers") OR (~ "Here", @- 25, ~ W"to remove yourself from future") OR (~ W"To never recieve another email") OR (~ W"Stop recieving these") OR (~ W"reM0VE LNK") OR (~ W"If you would like to be removed from future offers just reply") OR (~ W"If you wish to stop sending of all our mails") OR (~ W"choose to stop further sending of mail") OR (~ W"FOR no further ads, click here") OR (~ W"Remove requests honORed 100%") OR (~ W"You will not get anymORe of our emails if you go here") OR (~ W"to stop email",@-30, ~ W"clk link") OR (~ W"click here to discontinue these messages") OR (~ W"De1ete your e-mai1") OR (~ W"to cease future mailings click on this") OR (~ ~ "Opt-Out here")	Unsubscribe Language

$define remove13	(~ W"wish to discontinue sending of emails") OR (~ W"Add me to the",@-10, ~ ~ "No send list") OR (~ W"be permanently remove from this distribution system") OR (~ W"If you want to discontinue sending of all our emails") OR (~ W"Add my address to the",@-20, ~ ~ "optoutlist") OR (~ W"To Be Removed From This Update List") OR (~ W"Stop receive from me") OR (~ W"nothing elese fOR me") OR (~ W"to ensure that the unsubscribe process has been completed successfully") OR (~ W"To remove yourself from this list, please click here") OR (~ W"To unsubscribe, visit here") OR (~ W"T@ke me 0ff") OR (~ W"you can be deleted from our 0pt", @-3, ~ "list") OR (~ W"cli ck here to be removed") OR (~ "w@nt",.*, ~ ~ "to be R E M 0 V E D") OR (~ W"Please stop future announcements") OR (~ w"Visit us anytime") OR (~ W"Wish not to recieve") OR (~ W"TAKE ME OFF HERE") OR (~ W"op-t out") OR (~ W"If you have recieved this in error please use") OR (~ "PleaseRemoveMeFromMailingList") OR (~ W"reply to this message with REMOVE in the subject line")	Unsubscribe Language
$define remove14	(~ w"to obstain from", @-15, ~ W"mailing", WP0, ~ W"click here") OR (~ W"To change your e-mail address or subscriber preferences") OR (~ W"For future listing options:") OR (~ W"to stop future contact", @-35, ~ W"<A HREF") OR (~ W"No More Offers", W0, ~ "</a>") OR (~ W">no more", W0, ~ "</A>") OR (~ W"to unsubscribe", @-35, ~ W"<A HREF") OR (~ W"To modify future contact options") OR (~ W"To be expunged from our list please proceed") OR (~ W"unsubscribe by sending any email to") OR ($website,.*, ~ ~ "/nothanks.php")	Unsubscribe Language

$define remove15	(~ ~ "Change your email preferences") OR (~ "REMOVE",@-25, ~ ~ "FROM FUTURE E-MAILS") OR (~ W"To stop future advertisement") OR (~ "git ",@-10, ~ "off ") OR (~ ~ "To opt out",.*, ~ "http") OR (~ W"2 Get Taken Off") OR (~ W"Stop promos") OR (~ "http",.*, ~ "remove") OR (~ "Click",@-20, ~ "href", .*, ~ "here") OR (~ W"To remove yourself from", @-75, ~ "http://") OR (~ W "remove your address from further mailings") OR (~ W"do not want to receive anymore promotions from us") OR (~ W"removed from",@-10, ~ W"future promotion") OR (~ ~ "Press here") OR (~ "Press",.*, ~ W"remove from this list") OR (~ ~ "This email advertisement") OR (~ "GetOffTheList") OR (~ ~ "stop receiving e- mails from us") OR (~ ~ "UsnuubcribeHeree") OR (~ ~ "press to stop receiving") OR (~ ~ "onsubscr") OR (~ W"No more",@-3, ~ "mail") OR (~ W"This is an advertisement") OR (~ ~ "optout from future messages") OR (~ W"be omited") OR (~ ~ "r m x on the bottom") OR (~ ~ "This email is not S P A M") OR (~ ~ "This is NOT unsolicited email") OR ($website,.*, ~ "/unsub",WP1) OR ($unsub)	Unsubscribe Language
$define exclusion	(~ W"excluded from our databases of addresses") OR (~ W"excluded from", WP1, (~ "future" | ~ "further"), @-25, (~ "offers" | ~ "mailings"))	Unsubscribe Language

$define wish	(~ W"not wish to receive further messages") OR (~ W"not wish to receive future messages") OR (~ W"not wish to receive further email") OR (~ W"not wish to receive future email") OR (~ ~ "no longer wish",@-30, ~ ~ "receive our email") OR (~ W"If you wish to be removed") OR (~ W"If you no longer wish to be a member and receive our emails") OR (~ W"wish to remove your name and e-mail address from our database") OR (~ W"if you no longer wish to be infORmed", @-150, ~ W"unsubscribe") OR (~ W"wish to no longer be on this", @-25, ~ "list2	Unsubscribe Language
$define mac	($unsubscribe) OR ($remove1) OR ($remove2) OR ($optin) OR ($exclusion) OR ($legal1) OR ($legal2) OR ($wish)	Unsubscribe Language
$define mac2	($mac) OR ($remove3) OR ($remove4) OR ($remove5) OR ($remove6) OR ($remove7) OR ($optin2) OR ($remove8) OR ($remove9) OR ($remove10) OR ($remove11) OR ($remove12) OR ($remove13) OR ($remove14) OR ($remove15) OR ($randcharend)	Unsubscribe Language

Appendix B: Pornography and Harassment VDLs

Racial Slur VDLs
VDL
:HARASVDL slur01, ($pronoun | $badadj), ~ "pollack",$ending#
:HARASVDL slur02, ($pronoun | $badadj), ~ "nigger",$ending#
:HARASVDL slur03, (NOT ~ "spic",@-5, ~ "span") AND ($pronoun | $badadj), ~ "spic",WP1 #
:HARASVDL slur04, ($pronoun | $badadj), ~ "wop",$ending#
:HARASVDL slur05, ($pronoun | $badadj), ~ "dago",$ending#
:HARASVDL slur06, ($pronoun | $badadj), ~ "kike",$ending#
:HARASVDL slur07, ($pronoun | $badadj), ~ "honkie",$ending#
:HARASVDL slur08, ($pronoun | $badadj), ~ "whitie",$ending#
:HARASVDL slur10, ($pronoun | $badadj), ~ "limey",$ending#
:HARASVDL slur11, ($pronoun | $badadj), ~ "infidel",$ending#
:HARASVDL slur12, ($pronoun | $badadj), ~ ~ "skinhead",$ending#
:HARASVDL slur13, ($pronoun | $badadj), ~ ~ "neonazi",$ending#
:HARASVDL slur14, ($pronoun | $badadj), ~ "Aryan",$ending#
:HARASVDL slur15,W1, (~ "mutha Fucka"),$ending#
:HARASVDL slur16,W1, (~ "mother fucker"),$ending#
:HARASVDL slur17,W1, ~ W"motha fuck" #
:HARASVDL slur18, ($pronoun | $badadj), (~ ~ "dickhead"),$ending#
:HARASVDL slur19, ~ "little" AND $pronoun,W1, ~ "vulgar",@-10, ~ "maggot" #
:HARASVDL slur20,$badadj, ~ "jew",$ending#
:HARASVDL slur21,W1, ~ "Jew Bastard",$ending#
:HARASVDL slur22,W1, ~ ~ "towelhead",$ending#
:HARASVDL slur23, ($pronoun | $badadj), ~ ~ "son of a bitch",$ending#

Violent Acts VDLs
VDL
:HARASVDL violentact1,W1, ((~ "letís" | ~ "we'll" | ~ "going to"), (W1 | $badadj)), ~ "lynch"),WP1#
:HARASVDL violentact2, ($pronoun | $badadj), ~ "rapist",$ending#
:HARASVDL violentact3, (~ "want to" | ~ "going to" | ~ "have to"),W1, ~ " rape",WP1#
:HARASVDL violentact4, NOT ~ "Newsletter" AND W1, ~ "rape site",WP1#
:HARASVDL violentact5, NOT ~ "Newsletter" AND W1, ~ "rape house",WP1#

Sex Slur VDLs
VDL
:HARASVDL sexslur1, ($pronoun | $badadj), ~ "dyke",$ending#
:HARASVDL sexslur2, ($pronoun | $badadj), ~ ~ "cock-sucker",$ending#
:HARASVDL sexslur3, ($pronoun | $badadj), ~ "whore",$ending#
:HARASVDL sexslur4,W1, ~ " whor", (~ "ed" | ~ "ing" | ~ "er" | ~ "ers"),WP1#
:HARASVDL sexslur5, ($pronoun | $badadj), (~ "fag", ("" \| ~ "got")),$ending#
:HARASVDL sexslur6, ($pronoun | $badadj), ~ "butch",$ending#
:HARASVDL sexslur7, ($pronoun | $badadj), ~ ~ "dirty-girl",$ending#
:HARASVDL sexslur8, ($pronoun | $badadj), ~ "slut",$ending#
:HARASVDL sexslur9,W1, ~ ~ "candy ass"#

<table>
<tr><th>Sex Act VDLs</th></tr>
<tr><th>VDL</th></tr>
<tr><td>:HARASVDL sexact01, NOT ~ "Georgia" AND (W1, (~ "cum", ("" | ~ "ming"),WP1) OR ~ "cmum1ng" #</td></tr>
<tr><td>:HARASVDL sexact03,W1, ~ "jerk ", (~ "him" | ~ "me" | ~ "them")," off",WP1#</td></tr>
<tr><td>:HARASVDL sexact07,W1, ~ "fondl",@-10,W1, (~ "him" | ~ "her"),WP1#</td></tr>
<tr><td>:HARASVDL sexact09,W1, ~ "suck ", (~ "his" | ~ "my" | ~ "your" | $badadj), (~ " cock" | ~ " dick"),WP1#</td></tr>
<tr><td>:HARASVDL sexact10,W1, (~ ~ "dry hump"),$verbend#</td></tr>
<tr><td>:HARASVDL sexact11,W1, (~ "doggie",@-4, ~ "style"),WP1#</td></tr>
<tr><td>:HARASVDL sexact12,W1, ((~ ~ "sixty nine ", ("" | ~ "with" | ~ "to"),W1, (~ "me" | ~ "you" | ~ "her" | ~ "him"))</td></tr>
<tr><td>:HARASVDL sexact13,W1, ~ "do",$verbend,@-3, ~ ~ "sixty-nine") #</td></tr>
<tr><td>:HARASVDL sexact14,W1, (~ "pork",$verbend),W0, (~ "her" | ~ "him" | ~ "them" | ~ "you"),WP1#</td></tr>
<tr><td>:HARASVDL sexact14,W1, ~ ~ "Fuckable",WP1#</td></tr>
<tr><td>:HARASVDL sexact15,W1, (~ "bon", (~ "e" | ~ "ing")),W1, (~ "her" | ~ "him"),WP1#</td></tr>
<tr><td>:HARASVDL sexact16,W1, ~ "Give",W1,@-5, ~ " Head",WP1#</td></tr>
<tr><td>:HARASVDL sexact17,W1, ~ "blow ", (~ "him" | ~ "me" | ~ "you")),WP1#</td></tr>
<tr><td>:HARASVDL sexact17,WP1, ~ ~ "blowjob",WP1 #</td></tr>
<tr><td>:HARASVDL sexact18,W1, ~ ~ "finger fuck", ("" | ~ "ing")WP1#</td></tr>
<tr><td>:HARASVDL sexact19,$totalact,W1, (~ "her" | ~ "your" | ~ "their"), (W1 | $badadj) ~ "beaver",$ending#</td></tr>
<tr><td>:HARASVDL sexact20,W1, ~ ~ "muffdiv", (~ "e" | ~ "ing")#</td></tr>
<tr><td>:HARASVDL sexact21,$totalact, (~ "him" | ~ "her" | ~ "them" | ~ "you"),W1, (~ w"in the " | ~ w"up the"), (W1 | $badadj), (~ "ass" | ~ "butt" | ~ "bum"),WP1#</td></tr>
<tr><td>:HARASVDL sexact22,W1, (~ "shove " | ~ "stick"), ~ " it up your ass"),$ending#</td></tr>
<tr><td>:HARASVDL sexact23,$totalact,W1,$pronoun,W1, ~ "ass",WP1#</td></tr>
<tr><td>:HARASVDL sexact23, (~ "up" | ~ "in"), "the ",$badadj,W1, ~ "ass",WP1#</td></tr>
<tr><td>:HARASVDL sexact23a,W1, (~ "suck" | ~ "lick"),$verbend, ~ "ass",WP1#</td></tr>
<tr><td>:HARASVDL sexact25a,W1, ~ ~ "buttfuck",$wordend#</td></tr>
<tr><td>:HARASVDL sexact26a,WP1, ~ "anal",WP1, (@-100, ~ "Penetration" | @-100, ~ ~ "Sex" | @-100, ~ ~ "fuck" | @-100, ~ ~ "Free")#</td></tr>
<tr><td>:HARASVDL sexact26b,WP1, ~ "anal",WP1, (@-100, ~ "f*ck" | @-100, ~ "f-ck" | @-100, ~ "f_ck" | @-100, ~ "fu*k" | @-100, ~ "fuc*" | @-100, ~ ~ "fuk")#</td></tr>
<tr><td>:HARASVDL sexact27a,$totalact,@-10,$pronoun,W1, ($badadj | ""), ~ "bush",WP1#</td></tr>
<tr><td>:HARASVDL sexact28a, ~ w"jizz",$wordend#</td></tr>
<tr><td>:HARASVDL sexact29a,$totalact,W1, (~ "her" | ~ "my" | ~ "your" | ~ "their"),W1, ("" | $badadj), (~ "gash" | ~ "slash"),$ending#</td></tr>
<tr><td>:HARASVDL sexact30p, ~ "felt",W1,@-10,$pronoun,W1, ~ "slash",$ending#</td></tr>
<tr><td>:HARASVDL sexact31a,W1,$totalact,W1,@-25,$pronoun,W1, ("" | $badadj), (~ "dick" | ~ "prick" | ~ "cock"),$ending#</td></tr>
<tr><td>:HARASVDL sexact32a, ~ W"kiddie porn",WP1#</td></tr>
<tr><td>:HARASVDL sexact33a, ~ W"candy whore",WP1#</td></tr>
<tr><td>:HARASVDL sexact34a, ~ W"man-boy love",WP1#</td></tr>
<tr><td>:HARASVDL sexact35a, ~ "kinderwhore",$ending#</td></tr>
<tr><td>:HARASVDL sexact36a,W1, ~ "poon" ("" | ~ "tang"),$ending,WP1#</td></tr>
<tr><td>:HARASVDL sexact37a,W1,$totalact,W1,@-10,$pronoun,W1, ("" | $badadj), (~ "puss", (~ "y" | ~ "ies")#</td></tr>
<tr><td>:HARASVDL sexact38a,W1,$totalact,W1,@-10,$pronoun,W1, ("" | $badadj), ~ "balls",WP1#</td></tr>
<tr><td>:HARASVDL sexact38c,W1, ~ "fist",$verbend#</td></tr>
<tr><td>:HARASVDL sexact40a,W1, ~ "0RGA$m" | ~ "ORG/\SM" #</td></tr>
<tr><td>:HARASVDL sexact42a, WP1, ~ "spread", ($verbend),@-3, ($pronoun),@-3, ~ "legs",WP1#</td></tr>
</table>

:HARASVDL sexact43a, ~ W"b@nging girls" #
:HARASVDL sexact44a,W1, ~ ~ "carpet munch", ($wordend | $verbend) #
:HARASVDL sexact45a, ~ W"getting fawkked" #
:HARASVDL sexact46a,W1, ~ "gulp",$verbend,W0, ~ "Cock" #
:HARASVDL sexact48a,W1, ~ "Jizz",WP1#
:HARASVDL sexact49,W1, (~ "Guzzle" | ~ "Drink" | ~ "gulp"),@-5, (~ "Jizz" | ~ "cum") #
:HARASVDL sexact50a, W1, ~ "Ejakulati0n" #
:HARASVDL sexact51,W1 ~ ~ "swallow his load"#
:HARASVDL sexact51b,W1, ~ "Suck",$verbend,@-3, (~ "him" | ~ "them"), ~ " off"#

Sex Parts VDLs
VDL
:HARASVDL sexparts01a,W1, ~ "clit", (~ "oris" | ""),$ending#
:HARASVDL sexparts02a,W1, (~ "my" | ~ "your" | ~ "his" | ~ "their"),W1, ~ "wang",$ending#
:HARASVDL sexparts03b,W1, (~ " her" | ~ "their"),W1, ($badadj \| ""), ~ "tit",$ending#
:HARASVDL sexparts04a,W1, ~ "schlong",$ending#
:HARASVDL sexparts05a,W1, (~ "his" | ~ "their" | ",W1,@-25, ~ "pecker"),$ending#
:HARASVDL sexparts11a,W1, ($pronoun,W1, ("" | $badadj), ~ ~ "asshole"),$ending#
:HARASVDL sexparts15a,W1, (~ W"set of knockers"),WP1#
:HARASVDL sexparts15b,W1, ($pronoun,W1, ("" | $badadj), ~ "knockers"),WP1#
:HARASVDL sexparts17a,W1, (~ W"bearded clam"),WP1#
:HARASVDL sexparts18a,W1, (~ ~ "fuckhole"),$ending#
:HARASVDL sexparts20a,W1, (~ w"loose meat sandwich"),$ending#
:HARASVDL sexparts25a,W1, (~ w"tampon socket"),$ending#
:HARASVDL sexparts29a,W1, ($pronoun | $badadj),W1, (~ " ~ "snatch"" \| ~ "quiff" \| ~ "muff" \| ~ "love purse" \| ~ "furry cushion" \| ~ "calyx" \| ~ "hot box"),$ending#
:HARASVDL sexparts33a,W1,$totalact,W0, ($badadj | $pronoun), ~ "boob",$ending#
:HARASVDL sexparts34a,W1, (~ W"pink oboe"),WP1#
:HARASVDL sexparts35a,W1,$pronoun, ("" | $badadj), (~ "gonzos" | ~ "wazoos" | ~ "titties")#
:HARASVDL sexparts38a,W1, ~ "twat",$ending#
:HARASVDL sexparts39a,W1, ~ "cunt",$ending#
:HARASVDL sexparts41a, (NOT ("pussy" | "Pussy")) AND $letterP,$letterU,$letterS,$letterS,$letterY #
:HARASVDL sexparts41b, W1, ($pronoun,W1, ~ ~ "horse cock"),$ending#
:HARASVDL sexparts41c, (NOT ("pussie" | ~ "puzzle" | "Pussie")) AND $letterP,$letterU,$letterS,$letterS,$letterI,$letterE #

<table>
<tr><th>Profanity VDLs</th></tr>
<tr><th>VDL</th></tr>
<tr><td>:HARASVDL profanity2a, ~ "fuck",$wordend,WP1#</td></tr>
<tr><td>:HARASVDL profanity2a1, ~ "fuck",$punct, ~ "head",$ending#</td></tr>
<tr><td>:HARASVDL profanity3a, ($pronoun | $badadj),, (~ ~ "asshole"),$ending#</td></tr>
<tr><td>:HARASVDL profanity3a, ($pronoun | $badadj), ((~ "asshole",$ending) | (~ "bastard",$ending))#</td></tr>
<tr><td>:HARASVDL profanity3b,W1, (~ "Crazy" | ~ "Nasty"),.*, ~ "Bitch" # ;</td></tr>
<tr><td>:HARASVDL profanity3c, W1, ~ "shithead" OR ~ W"holy SH | T" #</td></tr>
<tr><td>:HARASVDL profanity4a, W1, ~ " hot",.*, ~ " ass " #</td></tr>
<tr><td>:HARASVDL profanity5a,W1, ~ " fuc",@-3, ~ "ing",WP1 #</td></tr>
<tr><td>:HARASVDL Sexact 00005,WP1, ~ "fuck",$wordend#</td></tr>
<tr><td>;:H 00115, (WP1, ~ ~ "F",$a [0-1] , ~ ~ "ck",$wordend) OR (WP1, ~ ~ "Fu",$a [0-1] , ~ ~ "k",$wordend)#</td></tr>
<tr><td>:H 00116,WP1, ~ "lesbo",$ending#</td></tr>
<tr><td>:H 00117,WP1, ~ "lezzie",$ending#</td></tr>
<tr><td>:H 00118,WP1, ~ "bugger", (~ "ing" | ~ "ed"),WP1#</td></tr>
<tr><td>:H 00119,WP1, ~ "buggerer",$ending#</td></tr>
</table>