Cybersoft.com

The World is at War in Cyber Space
We Can Protect Your Company
We Will Be The Last Man Standing

CyberSoft White Papers

CYBERSOFT™ White Papers

Print this Document

Extensions to CVDL, the CyberSoft Virus Description Language

Dr. Rick Perry

VFind Version 11.2.1 or higher is required

Last Modified: Saturday, 11-Aug-2001 23:52:53 EDT

Introduction

This report documents extensions made recently to the original CyberSoft Virus Description Language (CVDL), and also provides a complete summary of the current language. If you have not done so already, you should first read the original 1996 paper on CVDL for background.

Many of the extensions to CVDL described here were motivated by the use of CyberSoft's VFind Security ToolKit to scan email for viruses and spam. In particular, the pattern matching capabilities of VFind are uniquely suited to search for and block both viruses and spam in parallel. However, the original CVDL was lacking in some constructs which would: (1) make it easier to specify certain patterns; (2) increase the speed of scanning for certain patterns.

A future revision to CVDL may add full-blown regular expression pattern matching, similar to that used in Unix utilities like ***awk***, ***grep***, ***perl***, etc. For now, the CVDL extensions described here were designed to implement certain constructs which have immediate applicability for anyone who writes their own VDL patterns. The extensions include both low-level operators, such as matching phone numbers, and high-level logic operators such as OR, XOR, and NOT, which are used to logically combine low-level patterns.

High-Level Logic Operators

The original CVDL had only one high-level logic operator, &, meaning AND. The AND operator specifies that two or more patterns must all appear, at any positions in the scanned file, for a match. For example:

```
"pets" AND "cat"
   The word "pets" must occur somewhere in the file,
   and the word "cat" must also occur somewhere.
"pets" AND "cat" AND "dog"
   All three of "pets" and "cat" and "dog" must occur
   somewhere in the file for a match.
```

In the original CVDL, AND was specified by the & character; in extended CVDL, AND can also be specified by the words: AND, And, and.

The new logical operators, and alternative case-sensitive spellings, are:

OR Or, or XOR XOr, Xor, xor NOT Not, not

Do not confuse the high-level OR operator with the original low-level `|' operator. The `|' operator specifies the occurrence of patterns at a position relative to the preceding pattern in the scanned file. For example:

```
"x", ("a" | "b"), "y"

  matches "x", followed by "a" or "b", followed by "y"
```

```
    at some position in the scanned file.
```

The high-level OR operator specifies the occurrence of patterns at ***any*** positions in the scanned file. For example:

```
  "pets" AND ("cat" OR "dog")

    matches a file which contains the word "pets" anywhere,
    and also contains either the word "cat" or "dog" anywhere.
```

XOR is the exclusive-OR operator, which by definition can be expressed using AND and OR:

```
  x XOR y == (x AND NOT y) OR (y AND NOT x)
```

but is more efficiently expressed and implemented in extended CVDL by the XOR operator, so you do not have to specify the `x` and `y` patterns twice.

NOT by itself seems to be a strange logical operator for pattern matching, since it means that we have a match if some pattern is not present. However, consider a situation where all files or email must contain a disclaimer, and we want to detect the lack of that disclaimer. An example pattern is:

```
 NOT "The views expressed here are my own and ",
    "do not reflect the position of my employeer"
```

If such a disclaimer was required to be present in all outgoing email from a company, that NOT pattern could catch messages which do not contain the required disclaimer.

Mid-Level Pattern Operators

The mid-level pattern operators match an indefinite number of bytes in the scanned input file.

Matches one or more white space characters. For WS1 and WS0, white space is defined as " " (blank), "\t" (tab), or "\\\n" (backslash newline).

WS0

Matches zero or more white space characters.

Example:

```
  "mailx", WS1, "<", WS0, "/etc/passwd"

    matches "mailx" followed by one or morewhite spacee characters,
    followed by "<", followed by zero or morewhite spacee characters,
    followed by "/etc/passwd"
```

~~ is designed for matching phone numbers, and is an extension of the original CVDL 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.
```

\d+

\d+ matches one or more digits, and is named after the similar Perl operator. This is useful for detecting obfuscated URL's, for example:

```
"http://", \d+, "/"

matchesURLss like http://3626287830/

"http://0", \d+, ".", "0", \d+, ".", "0", \d+, ".", "0", \d+, "/"

matchesURLss like http://00000325.0000030.00000341.00000116/
```

~#

~# 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.
```

Low-Level Pattern Operators

The original low-level pattern operators such as ",", "@", "|", etc. are described in the 1996 paper on CVDL.

One new low-level pattern operator, **ABS**, has been added, for specifying an absolute offset from the beginning of the scanned data to match a pattern. Alternative case-sensitive spellings for ABS are: Abs and abs.

Note: when a VDL is applied during scanning, if the current input data buffer position is beyond the position of an ABS position specified in the VDL, the ABS data will not be matched.

Example VDLs using ABS:

```
% cat abs.vdl
```

```
:a1, ABS 0, "#!", WS0, "/bin/sh" #

:a2, "abc", @-20, "def", ABS 14, "01234" # ; this is bad

:a3, "abc", @-20, "def" AND ABS 14, "01234" # ; this is ok

:MS/VBA, ABS 0, "\xD0\xCF\x11\xE0\xA1\xB1\x1A\xE1" AND "\xFE\xCA" #
```

The MS/VBA VDL example uses `ABS 0` to check for the 8-byte Microsoft signature file header which appears at the very beginning of most Microsoft application files.

The a1 VDL example checks for a Bourne shell script file header.

The a2 VDL example checks for "abc" followed by "def" within the next 20 bytes, followed by "01234" at absolute position 14. The is a bad example of the use of ABS, since if "def" is matched past absolute position 14 then "01234" will not be matched.

The a3 VDL example shows the correct way to implement the intent of the a2 example by using AND in conjunction with an ABS VDL.

Example run:

```
% cat abs.txt

#! /bin/sh
abc01234def

% vfind --vdl=abs.vdl abs.txt
...
##==>> Loading VDL code from: abs.vdl
##==> VDL model for `a1' loaded.
##==> VDL model for `a2' loaded.
##==> VDL model for `a3' loaded.
##==> VDL model for `MS/VBA' loaded.
##==> Checking file: "abs.txt"


##==>>>> VIRUS POSSIBLE IN FILE: "abs.txt"
##==>>>> VIRUS ID: CVDL a1

##==>>>> VIRUS POSSIBLE IN FILE: "abs.txt"
##==>>>> VIRUS ID: CVDL a3 %
```

Note that VDL's a1 and a3 matched the input, but a2 did not.

CVDL Macros

A VDL macro facility has been added to CVDL. VDL macros are simple name/value replacement macros and take no arguments, i.e. they are not function-like macros.

A VDL macro is specified using `$define` as the first word on a line, preceded by optional white space, 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 (starts with a letter then has 0 or more letters or digits), white space, value (continues to the end of the line). Trailing white space on the value is trimmed. Letters are [a-zA-Z_] and digits are [0-9]. The restrictions on characters in a VDL macro name are required for ease-of-use in writing and parsing macro invocations.

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 can not be used in a VDL rule before being defined, but they can be used in other VDL macro $define's before being defined. Macros have per-file scope; macros defined in one VDL file do not carry over to other VDL files.

Examples:

```
  % cat mac1.vdl

 $define  pf1    $pets AND $foo
d $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.

```
  % cat mac1a.txt

  The dog ate some pie.

  % cat mac1b.txt

 I ate some pie.

 % vfind --vdl=mac1.vdl mac1a.txt mac1b.txt

 ##==>> Loading VDL code from: mac1.vdl
```

```
##==> VDL macro `pf1' loaded.
##==> VDL macro `pf2' loaded.
##==> VDL macro `pets' loaded.
##==> VDL macro `food' loaded.
##==> VDL model for `v1' loaded.
##==> VDL model for `v2' loaded.
##==> Checking file: "mac1a.txt"

##==>>>> VIRUS POSSIBLE IN FILE: "mac1a.txt"
##==>>>> VIRUS ID: CVDL v1

##==>>>> VIRUS POSSIBLE IN FILE: "mac1a.txt"
##==>>>> VIRUS ID: CVDL v2

##==> Checking file: "mac1b.txt"
##==>>>> VIRUS POSSIBLE IN FILE: "mac1b.txt"
##==>>>> VIRUS ID: CVDL v1
```

File Type Restriction Directives

File type restriction directives may now be specified in CVDL files. When specified outside of a VDL, these directives have VDL file scope and apply only for VDL rules which appear following the directive in the same VDL file. When specified as part of a VDL, the directives apply only for that particular VDL.

The file type restrictions apply only if vfind is reading SmartScan input from UAD, which reports the file types. The directives and their meanings are:

```
<"...",...>  specifies a list of file types to scan,
              i.e. scan only the file types specified.

<!"...",...> specifies a list of file types to not scan,
              i.e. scan everything except for the file types specified.

<>           resets to scan everything.
```

If the SmartScan file type reported by UAD is "unknown", or if VFind is run standalone (without SmartScan input), then all VDL file type restrictions are ignored and everything is scanned.

Example VDL file:

```
:v1,"..."# ; all file types

<"text"> ; only "text" file types for the following vdl's

:v2,"..."#
:v3,"..."#

<!"HTML"> ; no "HTML" file types for the following vdl's

:v4,"..."#

:v5, <"JPEG","GIF"> "..."# ; only "JPEG" and "GIF" file types for v5
```

```
:v6,"..."#
```

```
<> ; all file types for the following vdl's
```

```
:v7,"..."#
```

VDL v1 will be used for all file types; v2 and v3 will only be used for "text" file types; v4 and v6 will only be used for non-"HTML" file types; v5 will only be used for "JPEG" and "GIF" file types; v7 will be used 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 SmartScan-reported type.

VDL Version Reporting

Versions for VDL files and rules can now be reported using an extension to the file type restriction syntax. If you use 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.

Here is an example which specifies a version for the VDL file and a version for VDL rule `b':

```
% 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"
...
```

We plan on extending the syntax of VDL file type restriction directives even further, to allow specification of scanning engines, speed, etc. on a per-file or per-VDL basis.

Language Syntax Summary

The CVDL language syntax summary is available in a separate file which is updated as new features are added to the language.

New VFind Command-Line Options

-vdl0=*vdlfile*, --vdl0=*vdlfile*

Tells VFind to read additional speed=0 virus descriptions from *vdlfile*.
With speed>0, most VDL rules are compiled into a parallel search engine, which provides fast scanning but no control over the order in which the VDL patterns are applied. With speed=0, VDL rules are placed in a first-in-last-out queue, so the last rule specified is the first one executed, and speed=0 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=*vdlfile*, --vdlc=*vdlfile*

Tells VFind to read additional case-insensitive virus descriptions from *vdlfile*.
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.

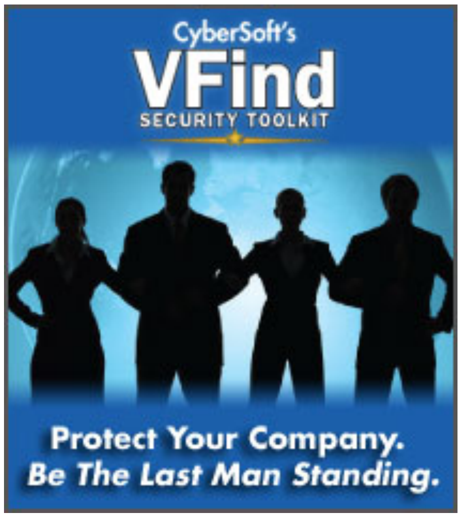
-#=*n*, --#=*n*

Stop scanning a file after finding ***n*** 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'

-sst, --smartscan-types
SmartScan Types: Displays file types and any VDLs skipped due to file type restrictions.
-p, --per-file
Per-file: count of number of possible virus infections displayed for each file.
-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.
--rcf=*rcfile*
Run Control File. Tells VFind to read additional command-line arguments from *rcfile*.

AhnLab
V3 Internet Security 8.0
Fastest and Lightest
All-in-One Security Solution
Exclusively imported and supported in the United States and Canada by CyberSoft Operating Corporation



508 Compliance TestPros TESTED

This site certified 508 Compliant