CyberSoft White Papers

CYBERSOFT™ White Papers

Print this Document

The CyberSoft Virus Description Language

Peter V. Radatti
Cybersoft, Incorporated

There were multiple reasons why CyberSoft felt it necessary to develop a virus description language. The increasing sophistication of the problems that CyberSoft was resolving using it's VFind product were becoming increasingly difficult using standard scanning technology. Many of the viruses that directly attacked VFind's primary platform, UNIX, were written entirely in source code and executed in interpretative languages such as script. Scan codes could not be easily designed to find a virus in which white space, the use of tabs and variable names change. Normal scan codes depended on the fact that binary executables contained stable strings of code that could be searched for at specific addresses. This was only partially true in the UNIX environment. Since VFind was designed to search for UNIX, MS-DOS, Apple Macintosh and Commodore Amiga viruses on the UNIX platform, addresses could no longer be specific since the infected file might exist within a pseudo-disk or a compound file such as a tar file. In addition,

the sequences of stable code values had to be increased in size to hold statistical validity and not generate false hits.

Scanning for viruses written in source code required several innovations in virus scanners. Many of the features required were normal parts of compiler parsers. Compiler parsers are the first step in the process of taking a computer program written in a source language and producing a binary executable. CyberSoft felt that a compiler parser would provide answers to its technical goals, however, it would be necessary to define an entire language for the parser to work correctly. At the time this decision was being made, December 1991, CyberSoft was unable to locate any standards for a virus description language. It was therefore necessary for CyberSoft to design its own language to solve this problem. The language was defined in January 1992 and named the CyberSoft Virus Description Language or CVDL.

During the design of CVDL, several goals were employed. The first was to design a universal way of describing pattern matching. The second requirement was that the language would incorporate enough features that unforeseen future requirements could be resolved without changing the language or code. The grammar and versatility of the language could become a standard which would allow general programming within the pattern matching framework. These goals dictated many of the intrinsic features within CVDL, including the necessity to process any character or hex stream. Originally, we desired the capability of processing any length pattern, however, timing constraints and practical limits prevailed and a limit of 32,000 bytes per pattern was defined. Boolean operators were defined and upper/lower case sensitivity or case insensitivity was made a user selectable option. One of the hardest requirements to efficiently design was the ability to provide forward reference proximity scanning. This feature was a necessity for the UNIX environment and its source code viruses. Proximity scanning allows the user to define a virus or pattern that will not be affected by the ambiguities of the typist. It was determined that since CVDL would be a language of general utility it could be used to scan for patterns other than viruses. CVDL can search for any pattern while VFind is performing a virus check. In this way, VFind with CVDL is capable of hitting two birds with one stone.

Using CVDL during a normal virus scan, VFind can perform keyword searches helping to secure company information. CVDL can be defined to scan for keywords such as "TOP SECRET" or "COMPANY CONFIDENTIAL" using case insensitivity and proximity testing. This allows the user to insure that compartmentalized information does not migrate from where it belongs and does not leave the facility on a tape that was virus scanned. It can also be used to help an analyst in sorting information or in cleanup after an information spill or reclassification. It is necessary to use a pattern matching language such as CVDL to resolve this type of problem instead of simple pattern matching utilities such as Unix "grep" or "awk" because of the need for proximity testing. Some document preparation packages do not represent text on the screen in the same manner saved to a file. There are common packages that use keywords such as " " to represent what would appear to be white space on a document screen. In addition, control characters, font change information, pagination and other variables make it almost impossible to locate patterns with high accuracy in documents. The pattern "TOP SECRET" may be represented as

"p12TuOuPu p12SuEuCuRuEuT" or any other variation. Using CVDL, many different possible patterns of actual code can be pattern matched with the original pattern within user defined constraints. In this way, CVDL is able to produce a basic model of a pattern that many other patterns of related origin can match with a high percentage of accuracy and integrity. A simple basic model of the pattern "TOP SECRET" defined in CVDL would b:

:top secret,~'t',@-10,~'o',@-10,~'p',@-20,~'s',@-10,~'e',@-10,~'c',@-10,~'r',@-10,~'e',@-10,~'t',#

In this model, each character is defined as case insensitive with a forward proximity of 10 bytes or characters. There is a forward proximity defined as 20 bytes or characters between the words "top" and "secret". This allows for tabs, white space and font change information that is larger than between characters within the same word. When the pattern model is matched, the message at the beginning of the definition between the colon and the first comma is displayed. The model is terminated with a pound sign. One or more models put together is a CVDL program. Each model may be up to 32,000 bytes long. Total program limit is defined by the systems available stack space. These models and programs, if required, could become very complex with the inclusion of Boolean operators.

The CVDL grammar is defined as follows:

1. Expressions

a. Decimal Expression
- any value in the set 0-9
- values can be 0 to 255 or 0 to 32767 depending upon use
examples: 255 3216

6 b. Hexadecimal Expression 0x
- used in prefix position
- any value in the set 0-9,A-F or a-f
- value(s) that follow must be 2 or 4 digits depending upon use
examples: 0x0a 0x01f

e c. Character expression ' quote
- used in prefix and postfix position to delimit the character
example: 'z

' d. Character String " double quote
- used in prefix and postfix position to delimit the string
example: "ABCD

" e. Literal Value \ backslash
- used in prefix position

- used for new line (\n), carriage return (\r), and tab (\t)
- can be used in String, Character, or standalone
example: "\n" or '\n' or \

n f. Offset Operator Expression
- Decimal or Hexadecimal following @ Operator
- Decimal value cannot exceed 32767
- Hexadecimal value cannot exceed 0x7fff
examples: @20169 @0x4ec

9 g. Range Operator Expressions
1. Range of Values Operator Expression
- Decimal or Hexadecimal expressions
- left and right expressions must be of the same type
- Decimal value cannot exceed 255
- Hexadecimal value cannot exceed 0xff
- Default left-hand value is 0
- Default right-hand value is 255
examples: 1-22,0x0a-0xff,10-,-200
- Character expression
- Default left-hand value is 0x20
- Default right-hand value is 0x7e
examples: 'a-z','a-','-z

2. Offset Range Operator Expression
- Decimal or Hexadecimal expressions
- left and right expressions must be of the same type
- Decimal value cannot exceed 32767
- Hexadecimal value cannot exceed 0x7fff
- Default left-hand value is 0
- Default right-hand value is 0x7fff
examples: @10-20169,@0x0a-0x4ec9,@10-,@-20

0 h. Repetition Operation Expression
- Decimal or Hexadecimal expressions
- Decimal value cannot exceed 255
- Hexadecimal value cannot exceed 0xf
fexamples: 1[22],2[0x4a]

2. Operators

: specifies Virus Name string operator
^ specifies exclusion operator (NOT)

~ specifies case insensitivity operator
[n] specifies repetition operator
- specifies range operator
@ specifies relative offset operator
() specifies grouping operator
| specifies Explicit Logical OR operator
& specifies Explicit Logical AND operator
, specifies field separator and Implicit Logical AND
operator
specifies end of Virus Description operator

3. Rules

a. The Virus Name Operator : semicolon
- used in prefix position
- values following are assumed to be ASCII
example :MAC Attack

b. Exclusion Operator ^ caret
- used in prefix position
- can be applied to a single value or a range of values
but not a group of values
examples: ^0xff ^1-10

c. Repetition Operator [] left and right bracket
- used in prefix and postfix position to delimit repetition count
- repetition count must be decimal or hexadecimal
- value cannot exceed 255 in decimal or 0xff in hexadecimal
examples: '1'[0xFF] or "ABC"[0x0C] or 0xee[31]

d. Range Operator - dash
- used in infix position
- values allowed are: decimal, hexadecimal and character
- pre and post values must be of the same type
(decimal and hexadecimal cannot be mixed)
examples: 1-20 0x22-0x45 'a-z'

e. Relative Offset Operator @ at sign
- used in prefix position
- adjust current position for further searching
forward reference only
- Range Token - (dash) is valid following @
- value cannot exceed 32767 in decimal or 0x7fff in hexadecimal

- must be followed by a comma and a valid data type
examples

@0x1E (New position = current position plus 1E)
@-0x1E (current position thru 1E)
@10-20 (New position = current position plus 10 thru New + 10)
@10- (New position = current position plus 10 thru EOF)

f. Grouping Operator () left and right parentheses
- used in prefix and postfix position to delimit a Set(s) of values
- can be nested to identify hierarchical dependencies
examples: ("ABCD",1,2,3,4) | (0x81,0x0a,12-15,@10,15[9])
(('x',2)|"XYZ")|88[100]

g. Explicit OR Operator | vertical bar
- used in infix position
- not valid when used with offset operator
examples: (1|2|9),22
(19,(1,2|9))|(22,28)

h. Explicit AND Operator & ampersand
- used in infix position
- specifies secondary search dependencies
- not valid when used with offset operator
example: 0x80,(1,4,9)&"ABCD"

i. Field Separator Operator , comma (Implicit AND Operator)
- used to delimit fields and operators
example: 0x80,(1|2,3,4)
0xff,0xe2,0xd1,@10,0x45

j. End of Description Operator # pound sign
- specifies end of Virus Description

k. Overall length of Entire VDL scan window cannot exceed 65535
i.e. 1,@-7fff,2,@-7fff,3 would not be allowed

Notes:

Positioning within the file/buffer to be searched is implied unless overridden/updated with the Relative Offset operator @.
Relative Offset is from end of the last value matched + 1.

4. Precedence of operators:

Operator	Associatively
: #	left to right, right to left
&	left to right
|	left to right
,	left to right
()	left to right
@	left to right
- []	left to right
~ ^	left to right

5. Example Descriptions:

:Hamburger Attack,0x83,0x49,@27,"Have a Nice Day",@44,128-129, 2|19|55),80[5]&0x69,'a',(0x4e|0x45|0xff),#

This example searches for two distinct data streams.

The first is:

position 0 = 0x83 AND position 1 = 0x49 AND
position 28 thru 42 = "Have a Nice Day" AND position 86 = 128 or 129 AND
position 87 = 2 or 19 or 55 AND position 88 thru 92 = 80 80 80 80 80

The second data stream is not dependent on the first data streams positioning.
It consists of the following :

position 0 = 0x69 AND position 1 = 0x61 AND position 2 = 0x4e or 0x45 or 0xff
The search will be complete when both data streams have been encountered.

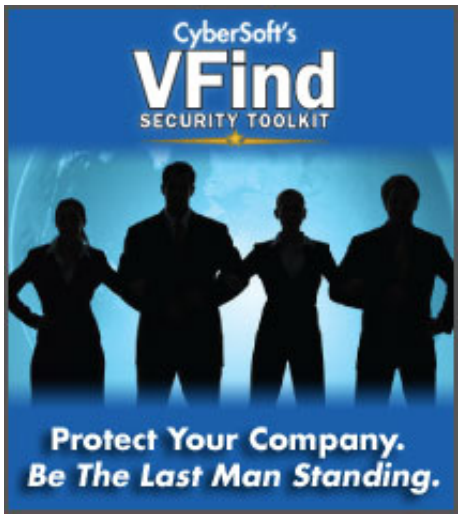
:nVIR,(0x4e,(0x71|'*'|0x75))|("A String",@19,^0x44,22[10],19-80),#

This example will search for one distinct data stream consisting of the following:

position 0 = 0x4e AND position 1 = 0x71 or 0x2a or 0x75 OR
position 0 thru 7 = "A String" AND position 27 NOT = 0x44 AND
position 28 thru 38 = 22 22 22 22 22 22 22 22 22 22 AND position 39 = any value between 19 and 80 inclusively

If you wish to comment on this paper you may address your correspondence to:

Peter V. Radatti, Cybersoft Inc.
1508 Butler Pike
Conshohocken, PA. 19428



508 Compliance TestPros TESTED

This site certified 508 Compliant