

Virus-description language

Peter Kleissner

This document is part of the “Hibernation File Attack” presentation at Black Hat Europe 2009. It describes the virus-description language used for defining values to be patched and how to apply the patch on the hibernation file.

Thanks to David M. Chess for his research work.

Virus Verification and Removal Tools and Techniques

David M. Chess 

*High Integrity Computing Lab, IBM T. J. Watson Research Center
Post Office Box 218, Yorktown Heights, NY, USA*

November 18, 1991

(Cover of original article by David M. Chess)

History

The virus-description language (vdl) was introduced by many people (and firms) in the early nineties. Actually David M. Chess was one of these people made the research work public and published an article, today still available on IBM research site [1].

He published it as part of VERV, a prototype virus verifier and remover, developed on the High Integrity Computing Lab at IBM Watson Research Center (like seen above). VERV was developed for PC-DOS and could verify over 40 different viruses and remove most common file infections at that time (1991).

VERV was only used internally by IBM; it was not published as a single product. However, I assume it has been integrated into IBMs “IBM AntiVirus” product.

IBM AntiVirus/DOS
for PC DOS 7.0

5555-937 Copyright (c) IBM Corp. 1989, 1994. All rights reserved.

Licensed Materials -- Property of IBM

(Product info of IBM AntiVirus)

Today

Today VERV is not in use anymore – the last update was in the year 1996. However the technology is still in use by leading AV vendors, they still have signature databases with their custom virus description language.

Signature Format

The virus description language is used to describe viruses and virus families. A single signature should find as many viruses as possible of a specific virus family. In the best case there would be one signature for one family – but in fact there are many signatures because of different sub versions of viruses. The format of a virus signature is optimized to find maximum viruses but at the same time being fast to apply.

The general format of a *virus signature* looks like:

```
One or more VIRUS records
A NAME record
One or more LOAD records
Zero or more DEGARBLE and related records
Zero or more ZERO records
One or more check records
Zero or more REPAIR blocks
```

A virus signature is a set of records.

Signature File

VERV implements a virus signature as plaintext – and so it is used in hibernation file attack. The signature file will be stored to C:\Signatures.vdb and is human read- and modifiable.

IBM AntiVirus implements signatures as binary encoded packets, they are not human readable and look like:

```
00000000 37 2E 30 20 28 31 39 39 34 2F 30 36 2F 30 31 29 7.0 (1994/06/01)
00000010 1A 1B 00 10 DC 0A 00 81 DC 31 39 39 35 2F 30 31 ....Û...Û1995/01
00000020 2F 30 31 09 00 08 00 42 45 47 49 4E 46 49 4C 45 /01....BEGINFILE
00000030 81 00 10 DC 08 00 11 DC 20 53 54 4F 4E 45 44 00 ...Û...Û STONED.
00000040 07 00 13 DC 53 74 6F 6E 65 64 00 06 00 18 DC 75 ...ÛStoned....Ûu
00000050 00 00 00 00 02 08 00 34 DC 89 01 A4 01 C6 54 D0 .....4Û%.α.ÆTÐ
00000060 1D 08 00 34 DC A5 01 AC 01 36 E4 89 24 08 00 30 ...4Û¥.-.6ä%$.0
00000070 DC 00 00 07 00 DE C3 66 F6 08 00 30 DC 15 00 88 Û....ÐÃfö..0Û..^
00000080 01 4E 89 80 8C 01 00 40 DC 76 06 00 4A DC 0B 01 .N%€E..@Ûv..JÛ..
00000090 08 01 07 01 00 00 50 DC 01 00 40 DC 77 06 00 4A .....PÛ..@Ûw..J
000000A0 DC 00 01 FD 00 FC 00 02 00 4D DC 00 00 00 00 4E Û..ý.ù...MÛ....N
000000B0 DC 00 00 52 DC 81 00 10 DC 12 00 11 DC 20 53 54 Û..RÛ...Û...Û ST
```

On the beginning of the signature file (“VERV.VDB”) you see a small header. Then, next, you see the signature of the very first famous MBR virus – “Stoned”. It was also mentioned by Symantec nowadays in their Sinowal paper.

Signature Records

A signature consists of a set of records, they can be:

VIRUS	Gives a simple list of one-word alias names for the virus
NAME	The primary name of the virus
FAMILY	Virus family name the virus belongs to
LOAD	Describes where in an infected file the virus can be found
DEGARBLE	DeGARBLE records tell the engine to perform a certain common type of degarbling
ZERO	Describes variable areas in the virus they have to be set to zero before any checks

	are done
Check	There are three basic types of check records (CODE, CONST, TEXT), describing different tests to be done
REPAIR	Tells the engine how to repair an infected file

Some of these records must exist for a signature and some can occur multiple times in a virus signature. You will see in context which records can exist multiple times. Every record is stored on a single line, with the typical end of line characters (CR + LF).

Records and the whole records set is backward but also forward compatible. The engine should skip unknown records, and new records will only extent functionality.

VIRUS Record

```
VIRUS Alias1 Alias2 Alias3 ...
```

The VIRUS record gives a list of one word aliases for the virus. They are not full names, only abbreviations. VIRUS records just exist to give the user short hand names for command line options in VERV.

NAME Record

```
NAME Virus-Name
```

The NAME record gives the full primary name of the virus.

FAMILY Record

```
FAMILY Virus-Family-Name
```

The virus family the virus belongs to. The family name can be used to tell the engine to scan the target for all virus family specific signatures. This feature is new to BHE2009 engine.

LOAD Records

```
LOAD Object-Type Offset Size
LOAD Object-Type SPECIAL Number
```

The LOAD records describe memory areas inside an object to copy into the engines internal buffer. The tokens of a load record are an object type, followed by either an offset and a length, or the keyword "SPECIAL" and a number. The offset and length, from the objects entry point, define the memory area to copy into the internal buffer.

If the offset to load from is not fixed, the SPECIAL keyword causes the engine to invoke an internal routine to perform special loading. That special loading can for example be to check the whole file against the signature (to use the whole file as internal buffer).

Following object types are defined:

P-COM	Prepending COM infector, supposable what today is known as "DOS stub"
S-EXE	EXE-infector, normal PE executable
E9-COM	for viruses that infect COM files by changing the first three bytes to a long jump to the virus (E9 is the hex code for a long

	jump)
E8-COM	for viruses that infect COM files by changing the first three bytes to a long CALL to the virus (E8 is a long call),
MBR	for viruses that infect hard disk master boot records and diskette boot records, and fit in a single sector
DISKETTE	for other sorts of diskette infectors (those that do not fit in a single sector)
HARDDISK	for other sorts of hard disk infectors (those that infect system boot records, and/or occupy more than one sector)

The object type is useful to determine the target file format.

The official documentation claims the count of load records to limit to different types “of objects that the virus can infect”. However, the BHE2009 engine extends this functionality to load multiple memory parts into the engines internal buffer.

DEGARBLE Records

```
DEGARBLE Number
DEXOR1 Data Offset Size Unknown/Maybe-Counter
```

The DEGARBLE record causes the engine to invoke an internal routine to perform decryption of general known encryptions (the number tells the kind of encryption, a function number).

The DEXOR1 record tells the engine to xor the bytes found at the offset with the specified byte. The third token is not documented, but assumable the counter or some other type that indicate how to change the xor value after every byte (this is used in polymorphic encryption).

There are some other, but undocumented degarbling records.

Internal Buffer

The most important thing behind VERV is its internal buffer used. After loading and degarbling, the engine has a complete “virus image” in its internal buffer. VERV includes a command line option to instruct VERV to save the contents of this buffer to a file.

ZERO Records

```
ZERO Offset Size
```

The ZERO record describes variables set to be zero before checking. This is a useful feature to reduce the number of records needed.

Check Records

Check records are used to describe memory areas to be checked against a CRC:

```
CODE Address Size CRC
CONST Address Size CRC
TEXT Address Size CRC
```

CODE records describe memory areas they contain code. If one CODE record fails the engine will indicate it is not the usual strain of the virus.

CONST records describe constant areas they should not change. They are exactly treated like code areas.

TEXT records describe memory areas that are not expected, but may change. This is useful to report new variants. Text records are not only used for text areas, messages that will be displayed to the user, but also to unused regions in the object.

REPAIR Records

```
REPAIR <Object-Type>
  EXE_LENGTH_BUG
  64K_COM_BUG
  FCOPY_TO -Count
  FCOPY_FROM Offset -Count
  BWRITE Internal-Buffer-Offset Offset Size
  BREAD File-Offset Size
  EXE_LENGTH_ADJUST Page-Count Last-Page-Length Subtract-Value
  R_SPECIAL Number
```

The REPAIR takes as token an object type (the same as in LOAD records), and following repair sub-records. The sub-records must be indented with two white spaces to differ them from normal records.

EXE_LENGTH_BUG	Tells the engine that the virus has the common bug that it assumes the image length to be the file length
64K_COM_BUG	Tells the engine that the virus has the common bug that it assumes the COM file length to be limited to 64 KB
FCOPY_TO	copies bytes from the start of the infected file up to a given number of bytes from the virus entry point (this is used to remove appending viruses)
FCOPY_FROM	copies bytes from the infected file, starting a given number of bytes from the virus entry point, and ending a given number of bytes before the end of the file (this is used to remove prepending viruses),
BWRITE	Copies bytes from its internal buffer to the file
BREAD	Reads bytes from the file into the internal buffer
EXE_LENGTH_ADJUST	Treats to words (first and second token as offsets) in the buffer as page count and as last page length fields of a PE header and subtracts them with the (accordingly adjusted) third token
R_SPECIAL	Cause engine to invoke an internal routine to perform functions not directly implemented in the language

Argh! Even Sinowal has the EXE_LENGTH_BUG! And the VERV paper was written 1991!

```
00000078  mov esi,[ebp+esi+0x50]    ; SizeOfImage; size of ntoskrnl
0000007C  add esi,ebp              ; absolute end address in memory
0000007E  dec esi
0000007F  or esi,0xffff
00000085  sub esi,0x1fff          ; alignment of end address to E00 (512)
```

(Original Sinowal hooking source code, MBR sector 60)

The REPAIR record has been enhanced with sub-records for BHE2009 engine:

```
REPAIR <Object-Type>
  FIXUP Address Data Data Data
  APPEND File-Name
```

The FIXUP sub-record tells the engine to fix single bytes at a given address in the file.

The APPEND record tells the engine to append a specific file. The file must lie in the same directory as the target object. The engine will determine on the object type how to append the file.

These two sub records have been included to add infection functionality for Hibernation File Attack.

Example Signatures

```
VIRUS slow slow-1721
NAME the Slow-1721 virus
LOAD P-COM 0 6B4
LOAD S-EXE 0 6B4
DEXOR1 001E 06AD 0012 0000 ; Degarble the code
DEXOR1 00EB 0159 0061 0001 ; and the data area
ZERO 0012 1 ; Zero the code-garble key
ZERO 0061 1 ; and the data-garble key
CODE 0000 00EA 38d5dc08 ; Code up to first data area
CONST 0144 014E 0ff22ad9 ; COMMAND.COM
CODE 015A 063C 74e00962 ; Code between data areas
CODE 0657 06AD ad3b0b41 ; After the second data area
```

This signature (for the Slow-1721 virus) loads 6B4h bytes from COM files and from EXE files from the entry point into its internal buffer. This means that the virus can either be a COM or an EXE file, but has – beside the headers – the same code and signature.

This is why we load from entry point (not file beginning) with LOAD records (and because headers can differ but the code won't).

Then 12h bytes at 6ADh will be xored with 1Eh, and 61h bytes at 0159h with 0EBh+1*Current Byte. 2 bytes at 12h and 61h will be zeroed out.

There are 3 code areas with a 31-bit CRC to be checked, and "COMMAND.COM" that should not change (most probably some target file of the virus).

```
VIRUS 1701
NAME the 1701 virus
LOAD E9-COM -1 06A5
DEGARBLE 1
CODE 0001 0026 19989c7e ; Degarble, MOV, jmp-in
CODE 0076 06A4 c03a91c5 ; Main code
```

This example now uses specific E9-COM object files. -1 is undocumented and seems to tell the engine to copy from the file begin, not the executable entry. The signature tells the engine also to invoke an internal routine for decryption.

```
REPAIR S-EXE
  EXE_LENGTH_BUG
  FCOPY_TO -0C5
  EXE_LENGTH_ADJUST 0053 0051 0710
  BWRITE 0043 0010 2 ; Fix SP
  BWRITE 0045 000E 2 ; Fix SS
  BWRITE 0047 0014 2 ; Fix IP
  BWRITE 0049 0016 2 ; Fix CS
  BWRITE 0051 0002 4 ; Fix image length
* Fixing COM files
REPAIR P-COM
  64K_COM_BUG
  FCOPY_FROM 0710 -5
```

This REPAIR record describes how to repair an infected file. There will be shown a warning that the virus makes the assumption that image size is equal to the file size when repairing. The appending virus will be removed and the length corrected to remove the virus. Also values like stack and instruction pointer will be corrected (in the PE header).

The * is undocumented, but may be a valid symbol for VERV to indicate a comment line. If the virus infected COM files, only the jump opcode has to be copied to remove the infection.

BHE2009 engine

I have mentioned the BHE2009 engine here on several positions, with that I mean the “Black Hat Europe 2009 Engine” (short BHE2009e, the last “e” stands for engine). Like I said in the introduction, the virus-description language is used for the Hibernation File Attack, and thus a virus engine was required (and in fact developed). The source code is public available via Hibernation File Attack presentation materials.

IBM AntiVirus

For my research work I downloaded an original copy of IBM AntiVirus from some ftp site [2]. The basic AntiVirus setup consists of following files:

ADMIN.PRF	IBM Antivirus profile
IBMAVSH.COM	IBM Antivirus shield program
IBMAVSP.EXE	IBM Antivirus IBMAV stand-alone program
LOCAL.MSG	IBM Antivirus text file, contains virus clean-up information
SHSIG.LST	IBM Antivirus virus signature file
VERV.VDB	IBM Antivirus binary database for disinfecting purposes
VIRSIG.LST	IBM Antivirus binary file containing virus signatures



I found out the last version to be 3.0 build 307 [3] from the year 1997. The product was also bundled to PC-DOS7. Another thing worth to mention is that IBM AntiVirus was primary implemented as DOS TSR (Terminate and Stay Resident) program (but also shipped as standalone tool IBMAVSP.EXE).

Although IBM AntiVirus is out of date, its files still contain interesting information. For example the VIRINFO.LST file contains very interesting information (what they did, how they work) about viruses of the last millennium.

A word to David M. Chess

I tried to contact him (using his e-mail address chess@watson.ibm.com) but got no response. It's a pity he didn't replied (I asked for further materials about VERV and the virus-description language) – first handed and solid information is always the best one.

Symantec Norton AntiVirus = IBM AntiVirus

During my research I found a very interesting inf (setup information) file [4] containing:

```
;*****  
;  
; Norton AntiVirus  
; Corporate Edition  
; Common INF file  
;  
;  
; Copyright(c) 1999-2000 Symantec Corporation
```

```
;*****  
...  
[CommonEngineFiles]  
...  
,, lic.dat  
,, vp*.vdb  
,, vd*.vdb  
,, ver.vdb  
,, virfilt.lst  
,, virsig.lst  
,, catalog.dat  
...
```

The two highlighted lines define Symantec Norton AntiVirus common engine files, which are also part of the IBM AntiVirus basic package (as I described on last page the basic setup of files).

The next thing I did was searching for IBM and Norton AntiVirus, what I found was Symantec news [5] from 21 December 1998:

“The initiative follows the announcement in May that Symantec licensed IBM's immune system technology and patents and would combine them with its own technology to produce a range of products and solutions to support IBM platforms. As part of that agreement, IBM assigned its existing anti-virus customers and OEM contracts to Symantec, and is encouraging its customers who currently have installed IBM anti-virus software to convert to Symantec's Norton AntiVirus software to get the monthly virus signature update files that IBM is no longer providing.”

Symantec's Norton AntiVirus is just a further developed version of IBM AntiVirus. Nice to know!

Conclusion

What I can say there is really magic in the air; such a concept is really great, but at the same time slow if you have to compare thousands of signatures with performing all its records. Also worth to mention, the virus-description language is not laid out for current file formats and current viruses (like text viruses as macros, scripts etc.).

So the virus-description language is better to be used by viruses rather than anti viruses. (They share the same technology, thought.)

References

- [1] Virus Verification and Removal Tools and Techniques, David M. Chess
<http://www.research.ibm.com/antivirus/SciPapers/Chess/CHESS3/chess3.html>

- [2] File list of PC-DOS 7 (containing IBM AntiVirus)
<ftp://ftp.3logic.net/pub/dos/pcdos7/PCDOS7.ENG/FILES.TXT>

- [3] IBM AntiVirus Introduction and User's Guide Version 2.5.1
<ftp://service.boulder.ibm.com/software/es/avug.ps>

The Integrated IBM Suites for Windows NT: A Powerful Package
<http://www.redbooks.ibm.com/redbooks/pdfs/sg245206.pdf>

- [4] Symantec AntiVirus installation configuration file containing references to IBM AntiVirus file
<http://www.dpsweb.com/Software/Anti-Virus/Symantec/AntiVirus/Corporate%20Edition/CD%202/NAVCORP/ROLLOUT/AVSERVER/LEGACY.INF>

- [5] Symantec Press: "Symantec, IBM Launch Anti-Virus Education Program Aimed at Current IBM Customers "
<http://www.symantec.com/press/1998/n981221.html>