Hooked on Mnemonics Worked for Me

An Intro to Creating Anti-Virus Signatures:

This is an introductory post on creating anti-virus signatures. This post will cover the three main types of signature detections. The most common signatures are hashes, byte-signature and heuristics. The intended audience is for malware analyst and reverse engineers. Experience in binary code analysis is expected. This article is going to focus primarily on creating signatures for Microsoft Portable Executables. The ideas expressed in this post could also be used for creating signatures on other file types that contain exploits. The reason for this post is because there is very little information in regards to creating anti-virus signatures on the intent. A few articles exist on how to create signatures with ClamAV or using diffing tools. I'm hoping the information in this post will be helpful for others in aiding in creating signatures for classifying malware or detecting malicious documents. If you have any questions or would like more clarity please leave a comment.

To error on the side of caution.
There are a couple best practices for creating signatures. The first one is do not target packer or cryptography library code. This type of code is often reused and has a high potential for false positives. The code is usually complicated and is hard to understand the assembly representation. In situations where the packer is unique the code can be used to identify the family of malware. This type of detection has it's flaws. Detecting malware based off of packers gives very little insight into classifying multiple variants. The code could be reused across multiple variants or authors. Packed code creates a dilemma for signature detection. If the files have been packed or compressed, the file will need to uncompressed or dumped before scanned. Anti-virus engines use emulators and unpackers to get the files to an uncompressed or dumped state before scanning the file. If the files are compressed or packed tools such as Python, TitanEngine or the Immunity Debugger could be used for creating dumps or uncompressed files. A side effect of these tools is they will need to be run in a isolated environment such as VMware or Wine. ClamAV can also unpack some packers such as UPX, FSG, Petite and a couple others.

Data that has been obfuscated or compressed should never be used as a candidate for a signature. As in the case of file hashes such as MD5; changing one byte of data can change the obfuscated or compressed code. Since the bytes can be easily changed by different data or key, there is a chance that the data will not be present in other variants. An example would be if the code obfuscated a URL and then the author changes the URL. All of the code would be static but the bytes of the obfuscated URL would be different. Hence breaking the signature. This scenario is not only applicable for PE files but also compressed steams in PDFs, compressed SWFs and other compressed or obfuscated data. If there is data in a section of a Portable Executable (PE) that is encrypted and is static throughout multiple files, this would be a good candidate for a sectional MD5.

The second best practice is do not create signatures on code that can not be understood. Experience in reverse engineering and analyzing a wide range of code will help with choosing sections of code that will not give a false positive. If the code's function can not be understood along with what is calling code, it should not be used as a signature. This will help eliminate the possibility of false positives on library code and other commonly used code. IDA's Flirt signature is extremely useful for helping with identifying common compiler code. If developers of open source and closed source libraries created Flirt signatures and sent them to Hex-Rays they would never have an anti-virus software detect their code again. If the code can be understood then more analysis will be needed or another section of code should be targeted.

The third is look for the author's hand or internet. A couple of questions to ask in regards to the authors hand. Is the section of code being targeted applicable only towards this piece of malware or is it a common routine such as reading a registry setting? Would the author have written this piece of code for this piece of malware or have hundred authors written the same routine for different programs? This can be difficult to access. Google can be helpful in this situation. Sometimes googling a combination of the APIs can return examples of the code or the malware source code.

The fourth best practice is to make sure to understand the scope of the scanned files and implications of detection. If the signatures are scanning a repository of malware then the signatures can be vague. If the signature will be scanning desktops across an enterprise network and deleting the detected files, the signature must be accurate. It's always best to error on the side of caution when their is a chance that removing a file can have negative effect. Creating multiple signature for multiple variants is usually safer than creating a generic signature that has potential for false positives.

The final best practice, strings should be used as a last resort. Strings rarely give any details about the actual code. Strings and data are cheap, code is more expensive and time consuming. Code is more likely to stay static. In some situations such as executables written in Visual Basic strings are advantageous but overall code is best to target. Code and strings can always be combined to make a stronger signatures.

Tools
There are a couple of open source tools that can be used for scanning files and will be used for examples in this article. Once the initial learning curve is over, these tools are easy to use and can be adapted quickly to most environments. These tools can be run in a Linux or Windows environments. Below is the name of tool and some of it's scanning capabilities. All commands are included in the examples so the reader can try out the examples.

* ClamAV - Hex Byte Scanning, regex, md5 file scanning, md5 sectional scanning, sigtool (tool for creating

signatures and hashes)
* Yara - A powerful rule based scanner that supports many conditions and data types, does not support hashing
* ssdeep - A tool for creating and comparing context triggered piecewise hash.

Hash Signatures
The most basic and easiest type of signature is a hash value. A hash value is created by a hash function that is a procedure or mathematical function which converts a large amount of data into a single value. The most commonly used hash function is MD5 and SHA-1. These hash functions are extremely accurate. For example if there is block of data that is hashed and then the same block of data has a byte changed then rehashed the hash values will be different.

```
md5 of the string "WeBuiltThisCity!"  = "07363ec9aa7a39c28da675ee2291946b"
md5 of the string "WeBuiltThisCity" = "80b135388f2c979d53c229546c129a61"
```

Md5 based signatures can be created using ClamAV. Yara does not support file hashing. ClamAV requires two attributes in order to create a MD5 hash signature. The first is the file size in bytes and the second is the MD5 hash. ClamAV comes with a tool called sigtool that can be used to generate MD5 signatures. Sigtool can be found in the "bin" directory in the installation folder of ClamAV.

```
C:\XOR\clamwin\bin>sigtool.exe --md5 1.txt
07363ec9aa7a39c28da675ee2291946b:16:C:\XOR\1.txt
```

ClamAV signatures are separated with by a colon ':'. The first part is the MD5 (07363ec9aa7a39c28da675ee2291946b), the second is the size (16) and the last part is the file location or output. The output is usually the malware name or something specific to the file. Colons can not be used in the output because these are treated as special characters in ClamAV signatures database. MD5 signatures need to be saved in a file with a .hdb extension. The above output from sigtool could be piped (>) to a file called shredder.hdb. This makes it very easy to create MD5 signatures with all files in a directory using a simple for loop in bash or the window command line. See reference 1 for more information. This also makes it possible to create ClamAV signatures from Virustotal results without having the file because the MD5 and file size are provided in the Virustotal Analysis.

Hash values are useful if the malware sample is static but if one byte in the executable is changed; the hash signature is broken. In some instances the code of the executable never changes but the data that the executable uses does. An example of code that might not change would be the executable produced by a GUI Remote Access Trojan kit. The codes sections would be the same but if one user used 192.168.0.1 and another used 192.160.2 the data sections would be different. In this example sectional hashing could be used to target the codes section data block and create a hash based signature.

```
Example of a sectional hash using ClamAV:
PESectionSize:MD5:MalwareName
.code:80b135388f2c979d53c229546c129a61:Rocksteady
```

The signature will need to be saved in a file with a .mdb file extension in order to be in the proper ClamAV format. As stated before colons are treated as special characters for separating data in the ClamAV signature database. The first part is the PE section name (.code), the second is the MD5 hash of the section (80b135388f2c979d53c229546c129a61) and the last is the output or name of the malware (Rocksteady). See reference 1 for more information.

Hashing can also be used to identify files that have been modified. Fuzzy hashing is a combination of recursive computing and context triggered hashing. It can be used to identify files that have had data deleted, modified or new data inserted. Jesse Kornblum is the developer of a tool called ssdeep. Below is an example of comparing two executable files with a single byte difference.

```
C:\XOR>md5sum *
623eea5c4c6209e1ebe44b2e6ca16428 *simple - edit.exe
e300ef28554d39ee3668dca05d5d5415 *simple.exe

C:\XOR>ssdeep *
ssdeep,1.1--blocksize:hash:hash,filename
384:hJyMBCI1Ex85hKuwoK/ZiBqv5ICmOKk7iw:hJlC38wBY0,"C:\XOR\simple - edit.exe"
384:BJyMBCI1Ex85hKuwoK/ZiBqv5ICmOKk7iw:BJlC38wBY0,"C:\XOR\simple.exe"

C:\XOR\simple.exe matches C:\XOR\simple - edit.exe (99)
```

Fuzzy hashing requires all the hashes to be stored in an ascii text file and then each hashed file will need to be matched against the ascii hash text file. This type of scanning is more computational intensive than other forms of scanning due to the hashing and comparing of all the hash values. An issue with fuzzy hashing (or all hashes for that matter) is that what might seem like slight modifications by the author/programmer can noticeably change the calculated match value. Our simple.exe was modified by changing some of the output strings and recompiled as simple2.exe; no code was changed.

```
C:\XOR\ssdeep *
ssdeep,1.1--blocksize:hash:hash,filename
384:BJyMBCI1Ex85hKuwoK/ZiBqv5ICmOKk7iw:BJlC38wBY0,"C:\XOR\simple.exe"
384:5JTMJCv1w05hDuwoKoRiBqP5IqmuKk7iw:5JOCXrQQ0,"C:XOR\simple2.exe"

C:\XOR\diff-strings\New Folder>ssdeep
ssdeep: No input files

C:\XOR\diff-strings\New Folder>ssdeep -d *

C:\XOR\diff-strings\New Folder>ssdeep *
ssdeep,1.1--blocksize:hash:hash,filename
384:BJyMBCI1Ex85hKuwoK/ZiBqv5ICmOKk7iw:BJlC38wBY0,"C:\XOR\diff-strings\New Folde
r\simple.exe"
384:5JTMJCv1w05hDuwoKoRiBqP5IqmuKk7iw:5JOCXrQQ0,"C:\XOR\diff-strings\New Folder\
simple2.exe"

C:\XOR\>ssdeep simple.exe > out.txt
C:\XOR\>ssdeep simple2.exe > out2.txt

C:\XOR\>ssdeep -d out*
C:\XOR\out2.txt matches C:\XOR\out.txt (50)
```

Since all hashes need to be saved to a text file and then compared ssdeep can be computational expensive. See reference 2 for more information.

Byte-Signatures

Byte-signature or byte detections are a signature based off a sequence of file bytes that are present in a file or data stream. Byte signatures are a very common form of detection and have been used since the first anti-virus scanner. Their usefulness is due to the accuracy they provide for detecting a sequence of bytes. The sequence of bytes is chosen because it exist in multiple variants of malware from the same family. Byte-signatures can be any type of data such as code or data contained inside of a data stream of an executable, a XORed .pdf or a Word document.

```
Example of a byte signature in the ClamAV format.
Simple dot exe:1:90FF1683EE0483EB0175F6
```

"Simple dot exe" is the output displayed by the ClamAV scanner; this is usually the malware family name. The second section is for the ClavAV engine to know the file type of the scanned file. The value '1' is for the engine to scan portable executable files. To scan any file type the value '0' needs to be used. The "90FF1683EE0483EB0175F6" is the hexadecimal representation of the opcodes. It represents the below assembly.

```
start: 0x401A2E  length: 0xC
90     nop
FF 16     call    dword ptr [esi]
83 EE 04    sub     esi, 4
83 EB 01    sub     ebx, 1
75 F6     jnz     short loc_401A30

Example of a byte signature in the Yara format.
rule example
{
strings:
signature = { 66 90 FF 16 83 EE 04 83 EB 01 75 F6 }
condition:
signature
}
```

Yara's signature format is much different than ClamAV. The syntax style is similar to a C struct. The first string defines that we are creating a rule of "example". The "strings" and "condition" are keywords used by the Yara engine. The strings defines the signatures. Yara's signatures can be any many formats from strings, hex-bytes, regex and a number of other formats. The keyword "condition" defines under what circumstances Yara should alert on the signature. There are many conditions that can be constructed for Yara to detect a file. See reference 6 for more information.

Binary Diffing

Manual and automated analysis techniques can be used for finding code blocks that are present throughout variants. The process of comparing multiple executables for similarities is called binary diffing. Manual binary diffing consists of reviewing the disassembly of multiple files and noting sections of code that are present in multiple files. Once blocks of code are identified a side-by-side comparison can be done checking for similarities. If the code has a high similarity it might be a good candidate for a byte-signature. A semi-automated approach of binary diffing would consist of grouping files into sets of files that are similar, use a disassembler to to get the assembly in a text output, diff the outputs, create more specific sets for diffs that match and then manually analyze the sections of codes that were found in the diffs that matched. For example, we could use IDA to create the assembly output and then use Kdiff to diff the assembly outputs.

```
C:\XOR\>dir
     23,503 simple.exe
     23,503 simple2.exe
```

```
C:\XOR\>"C:\Program Files\IDA Free\idag.exe" -B "simple.exe"

C:\XOR\>"C:\Program Files\IDA Free\idag.exe" -B "simple2.exe"

C:\XOR\>dir
     51,262 simple.asm
     23,503 simple.exe
    204,956 simple.idb
     51,274 simple2.asm
     23,503 simple2.exe
    204,956 simple2.idb
```

The "-B" flag is for creating an IDA database (.idb) and a text output of the assembly (.asm). The assembly output can then be diffed. Kdiff is a good tool to diff files because it does not do a straight diff of the two files. It will try to align sections that our separated by large amounts of data or code. These sections could be different because of a function that wasn't included in one executable or the presence of junk code or data. KDiff can also diff up to three files at a time. Automated tools such as Bindiff, PatchDiff2 and DarunGrim can be used to find differences between IDA database files. These tools are more specifically designed towards binary diffing paths. VxClass is another tool that can be used for creating sets of variants, binary diffing and identifying sections of code that could be used for potential signatures **.

Diffing is also very useful for displaying sections of code where wildcards will need to be used for byte-signatures. The wildcards will allow the scanner to ignore bytes of code that are not static throughout multiple variants. The non-static bytes could be caused by the insertion of junk code, addition or removal of code or the addition or removal of data. A slight change in file alignment can break byte-signatures that targets instructions. Common instructions that use offsets based off of file alignment are long jmps, call sub-routine or call an api or others that cmp, mov, sub, add, etc that reference offsets. It's best when creating signatures to always add the wildcards for the offsets even if they are static throughout multiple variants. An example below can be seen how file alignment can break byte-signatures. The two code blocks are from simple.exe and simple2.exe. The starting codes address, functionality and length are exactly the same. The only difference between the two blocks of code are the address offsets of the dword that is compared against ebx. The difference in file alignment was caused by the output strings being a different length.

```
simple.exe
start: 0x401902  length: 0x14
81 FB 8C 31 40 00   cmp     ebx, offset dword_40318C
0F 83 3A FF FF FF   jnb     loc_401848
BE 00 00 40 00    mov     esi, 400000h
8D 7D E0    lea     edi, [ebp+var_20]

81 FB 8C 31 40 00 0F 83 3A FF FF FF BE 00 00 40 00 8D 7D E0

simple2.exe
start: 0x401902  length: 0x14
81 FB 94 31 40 00   cmp     ebx, offset dword_403194
0F 83 3A FF FF FF   jnb     loc_401848
BE 00 00 40 00    mov     esi, 400000h
8D 7D E0    lea     edi, [ebp+var_20]

81 FB 94 31 40 00 0F 83 3A FF FF FF BE 00 00 40 00 8D 7D E0

ClamAV Format
Both Simple dot exe:1:81FB********0F833AFFFFFFBE000040008D7DE0

Yara Format
rule both_simple_dot_exe
{
strings:
signature = { 81 FB ?? ?? ?? ?? 0F 83 3A FF FF FF BE 00 00 40 00 8D 7D E0 }
condition:
signature
}
```

Heuristics

The last type of signature detections is heuristics. Heuristics is used when the malware is too complex for hash and byte-signatures. Heuristics is a general term for the different techniques used to detect malware by their behavior. It is one of the most complex forms of detections. An anti-virus engine might use emulation, API hooking, sand-boxing, file anomalies and other analysis techniques. Each anti-virus engine uses different algorithms and different proprietary techniques. A simple example of creating a heuristics signature would include an API logger and rules based off the APIs. Let's start with the "Hello World" of malware, Poison Ivy. We can use a server as an example for creating an API rule based signature. When Poison Ivy is executed it will create a mutex, write a registry key and copy itself over to the System32 directory. The default mutex for Poison Ivy is ")!VoqA.I4". Using Kerberos API Monitor we can simulate an API hook (that an anti-virus engine might use). A heuristics API rule for Poison Ivy can be seen below.

```
Rule A
An API call to RtlMoveMemory with a string of "SOFTWARE\Classes\http\shell\open\commandV"

Rule B
An API call to CreateMutexA with a string of ")!VoqA.I4"
```

```
Rule C
An API call to GetSystemDirectory

if ( Rule A then Rule B then Rule C )
then
Process  = PoisonIvy

Keribos Output
Rule A
simple.exe   | 00401447 | RtlMoveMemory(0012F458, 0040162F: "SOFTWARE\Classes\http\shell\open\commandV", 00000028) returns: 0012F4
...........
Rule B
simple.exe   | 0040155D | CreateMutexA(00000000, 00000000, 0012F43B: ")!VoqA.I4") returns: 0000003C
...........
Rule C
simple.exe   | 004018BF | GetSystemDirectoryA(0012F6F1, 000000FF) returns: 00000013
```

Parsing API logs from tools such as Norman and CWS Sandbox could be used to create similar rules for malware that is packed, encrypted or hash or byte-signatures fail to detect.

Conclusion
This post is just an introductory to creating anti-virus signatures. I am by no means an expert in the different technologies and algorithms used for scanning files. My experience is limited to hashing, byte-signatures, memory offset byte-signatures and creating detection mechanisms for malicious documents. I hope this article was useful and might give some ideas for other reverse engineers and malware analyst. If you know of any good articles on anti-virus scanning and the algorithms please leave a comment. I'd like to thank bone for all his help over the years.

References:
1. www.clamav.net/doc/latest/signatures.pdf
2. http://dfrws.org/2006/proceedings/12-Kornblum.pdf
3. http://www.virustotal.com/file-scan/report.html?id=fab272012d934f75915cd888f213e8857c390086363351eab3bf69f19ce67b65-1292244815
4. http://www.zynamics.com/bindiff.html, http://code.google.com/p/patchdiff2/, http://www.darungrim.org/
5. http://kdiff3.sourceforge.net/
6. http://code.google.com/p/yara-project/downloads/list
** The author has never personally used VxClass. This information is second hand from Zynamics blog and site.

12 comments:

Masokis January 18, 2011 at 2:09 AM

thanks sir... nice explaination
hope you can share about heuristic engine for detection too.

Reply

Matko January 20, 2011 at 3:10 PM

In my opinion the signatures in todays anti-virus solutions are too short and simple. We need more complex signatures with rules and deep binary inspection. Nice example for packers etc is TitanMist from ReversingLabs. Something similar must be used in malware recognition.

Reply

sri February 28, 2011 at 5:39 AM

CAN U TO TEL HOW TO CREATE AN ANTIVIRUS AND PROCEDURE FOR THAT

Reply

Z3Ro CooL March 8, 2011 at 4:34 AM

I tell you what? You could not have written it better. The best post on Virus Signatures I ever read. Thanks for putting in your time and effort into it. Thanks!

Cyber Crime

Reply

Mamoun Alazab January 24, 2012 at 4:37 AM

Loved reading your post. Good work.

Reply

Replies

Pattu February 15, 2012 at 10:13 PM

I want to know do antivirus softwares include API call sequences while creating a signature for a malware??????????

Reply

Shin-Kun July 20, 2012 at 10:30 AM

Nossa perfeito!!!
Gosto desse tipo de assunto.
Desperta em mim muita curiosidade de saber como funciona os motores de um Anti-Virus.
Obrigado.

Reply

Pablo Ramos January 8, 2013 at 11:12 AM

Amazing blog! Nice, clear and well written. Thanks for sharing!

Reply

Gert Horne January 18, 2013 at 7:53 AM

Thanks for this! very informative!

Reply

Anonymous April 5, 2014 at 11:01 PM

Hi,

Very good intro, there is some exciting news regarding creating virus signatures.. ClamAV team is now accepting "Community Signatures" into the official database. more details here http://www.clamav.net/lang/en/2014/02/18/introducing-clamav-community-signatures/
If you need samples to work on creating signatures you can get them on http://www.virusign.com.

Reply

Anonymous July 3, 2014 at 10:03 AM

Hansel your so dreamy

Reply

pkpdeveloper August 15, 2014 at 1:18 PM

thanks great info

Reply

Enter your comment...

Comment as: I am what I am ▾ Sign out

Publish Preview Notify me

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

Pages

- Home
- Portable Executable Virustotal Example
- Malware Analysis Search
- iheartxor
- injdmp

About Me

View my complete profile