Reverse Engineering Malware Basic Template

Tuesday, October 24, 2023 9:00 AM

Static Operations

SHA256sum	Hash the sample
VirusTotal **(Optional)**	Upload the sample to VirusTotal and look for malicious indicators.
Strings	Look for Windows API's, any commands, suspicious links, user agents, and other items. Still note APIs that have normal names but weird letters. Compare weird looking file headers with: FileSec.io
PEview	Look under the "IMAGE_FILE_HEADER" under the "IMAGE_NT_HEADER" tab and find the time/date stamp (Could be third entry). Log the timestamp and log if the timestamp looks incorrect. Look under the "IMAGE_SECTION_HEADER.text" under the "IMAGE_NT_HEADER" tab. Compare the virtual size and the size of the raw data (In hex) for the file. If the values are similar in size (Similar in a range of around 2000 mb) then the file is not packed. If the file is packed UPX0 is the .text file to look for. Usually packed files will have a very small import address file and the Raw data will usually be 0 while the virtual data will be a high number. If a file is packed note the packer. (Example packer: UPX) When a packed file is present LoadLibraryA and GetProcAddress may be present to load in the external library. NOTES: Borland Delphi compiler could display the timestamp somewhere in the year 1992
PEstudio	Under the imports tab note the indicators and details for what they may be attached to. Under the library's tab note any blacklisted libraries along with what they are and why they could be malicious. Note the imported Windows APIs. Compare these API calls with various sources to determine the good and bad APIs. Document what the APIs do and how they are malicious. Sources: Malapi.io **(Optional)** Could also look at the strings tab to look at additional information for strings. Any blacklisted strings should be documented along with the group and name of the string.
Capa	Use capa against the sample: `capa malware.exe` - Note the MITRE AT&CK techniques and MBC objective/Behaviors mapped to the give APIs. `capa malware.exe -vv` - Note the capability, namespace, functions, matches, and APIs matched to the triggers for the file. Docs

Dynamic Operations

Wireshark	Wireshark could be ran on the host machine or REMnux. Note any activity, HTTP requests, packet data, available. ## Host and REMnux
inetsim	Run inetsim (Could also pair that with wireshark on REMnux)
Procmon	Use the filter to input the name of the file or ports or other criteria to search for to find any kind of processes, connections, or file activity from the program. Note any malicious processes, and others. Look at properties of incoming items Look at process tree to see if any mallcious processes are the childen or parents of other processes.
TCPview	Observe active connetions when malware is ran. Note process name, protocol, state, local address, local port, remote address, and remote port of item.
nc	Use netcat to listen to ports on remnux from the host machine.

Advanced dynamic operations

Cutter	Look for je, jne, and jmp paths and patch where needed. How to find the main function in almost any binary: Find the program entry point and then look for "ret"s at the end of the graphs. Work the way up the functions graph until "mov ebp, esp" or "mov rbp, rsp" is present in a call. Use Cutter to also copy memory locations of sus api's to send to debugger How to extract shellcode from debugger: Count back the functions from a given API and right click the parameter wanted and click follow in dump and select the address. Under the dump area select the highlighted text down until the bottom green text ends. Extract that file as a bin file and use scdbg to parse the file
x64/32dbg	Remove pre-set breakpoints in the program if they are present. CTRL + F2 - Restarts the program F9 - Starts the program and then goes to next breakpoint F8 - Goes through each instruction F7 - Goes through each of the instructions in a call

	F2 - Set breakpoint To patch make sure to change ZF flag to 1 or 0 to patch live. Note this will only work if there is a test before the jump. Look for Anti-Analysis techniques Search for API strings if needed

Misc Tools

Base64	`base64 -d`
nc	Use netcat to listen to ports on remnux from the host machine.
unzip	Unzips zip files (REMnux)
AnyRun **(Optional)**	Use AnyRun to run the malware specimen.
ChatGPT **(???)**	Use ChatGPT to explain and decipher malware?
xxd	Displays a file in hex on Flare
HxD	Displays a file in hex on FLARE

Other file types

Excel	.xlsm indicates it's a macro file
	
	Use unzip in REMnux to unzip the file
	
	Then use oledump.py to extract
	
	`oledump.py example.txt`
	
	`oledump.py -s * example.txt`
	(To see the hex of what is in that stream)
	
	`oledump.py -s * -S example.txt`
	(Same thing as before but with a focus on strings)
	
	`oledump.py -s * --vbadecompresscorrupt example.txt`
	(Extracts and puts the stream back together)

Word	.docm indicates it's a macro file Unzip the file on FLAREVM instead of REMnux In rels in settings.xml.rels could be where the malicious content could exist. If not there somewhere else in the word file
Powershell	If any (0x9f, 0xd3, 0x6a) is found use cyberchef, or notepad++ to decode hex.

	If hex looks weird save it as a bin possibly
	
	If base 64 use Base64 on REMnux (send it via a text file with `python3 -m http.server 8000`)
	
	If base64 decoded looks weird save it as a bin or inflate it possibly with:
	
	`$shell = io.compression.deflatestream([io.memorystream] [system.convert]::Frombase64string('BYChCQAADMOuHaOmpnlfBG6kLw=='), [system.io.compression.compressionmode]::decompress) \| % {new-object system.io.streamreader($_,[system.text.encoding]::ascii) }| % { $_.readtoend()}`
	
	`write-host $shell`
	
	If the file has been exported as a bin file use `scdbg /f example.bin -s -1` (-s -1 means seps are unlimited) to extract the API calls for the binary

Virtual Basic	Look for weird commands or executions within the script. Look for the calling of additional API's/windows files within the script. Look for any COM classes Leverage MSBuild to run any XML files in any VNS scripts to see and understand what they are doing
HTA Files	Look for any shellcode but also examine how the shellcode works such as seeing and explaining certain commands that could be executed. Understand what is in the payload and how it operates.
DLLs/C#	Don't forget to floss! (Mscorlib a C# library) Considering C# files are in intermediate language use dnspy to view the intermediate language. Look for the classes (Under Yellow into the green program) on the side in dnspy. Within the program look for suspicious functions. After analyzing the functions run them with the command below: (Functions are numerical like 1,2,3, etc) `rundll32 example.dll,function`
Go	Don't forget to floss! Floss with -n 7 for a minimum length of 7 With floss maybe pipe it to grep with -i "go" to search for case insensitive go Use PE-Bear to analyze the Go file Under sections look for .symtab to see if the file is truly made with go
APK files	Run MobSF in REMnux: `docker run -it --rm -p 8000:8000 opensecurity/mobile-security-framework-mobsf:latest` (Or install it with `docker pull opensecurity/mobile-security-framework-mobsf`)

	Upload the file to REMnux then MobSF In the MobSF interface its possible to view the decompiled code and to view the components of the malware. Look under the applications permissions and note the associated permissions, status ,info, and descriptions with the APK file. Look and note the API for the APK. Note the files too.

The YARA

Yara

Example rule:

```
rule ExampleRule {

    meta:
        last_updated = "11/3/2023"
        author = "Name"
        description = "description of malware"

    strings:
        $example_string = "string" ascii

    condition:
        $example_string
}
```

Example real rule:

example.yara

```
rule Yara_Example {

    meta:
        last_updated = "2021-10-15"
        author = "PMAT"
        description = "A sample Yara rule for PMAT"

    strings:
        // Fill out identifying strings and other criteria
        $string1 = "YOURETHEMANNOWDOG" ascii
        $string2 = "nim"
        $PE_magic_byte = "MZ"
        $sus_hex_string = { FF E4 ?? 00 FF }

    condition:
        // Fill out the conditions that must be met to identify the binary
        $PE_magic_byte at 0 and
        ($string1 and $string2) or

        $sus_hex_string
}
```

To run the rule use: `yara32 example.yara (-r for recursive if you are scanning many`

```
files + file path) -w (suppresses warnings) -p 32 (-p to specify program threads)
```

The Report

The Executive Summary	Include the SHA265 hash and MD5 under the title. Tell the story but keep it brief. Give some technical details but not many. State the kind of malware and when it was first found. State how many payloads were found and what the payloads did. Include symptoms of infection such as URL's, files being deleted or added or modification of registry keys or finding other executables or files appearing in certain folders or directories files appearing in certain folders or directories list all of this in the appendix. Also state that the yara signature rules are located in the appendix.
High-Level Technical Summary	Include a paragraph (Or two small ones) about how the malware works but also include URL's associated with the malware and make sure to defang the URL's . Use diagrams and pictures to tell the story of the malware and how it functions.
Malware Composition	Here include the file names associated with the malware along with the hashes at the top. After this make short paragraphs about each of the files associated and maybe include pictures of different encodings slash bits of the file and annotate them.
Basic Static Analysis	Include many screenshots with annotations
Basic Dynamic Analysis	Include many screenshots with annotations
Advanced Static Analysis	Include many screenshots with annotations
Advanced Dynamic Analysis	Include many screenshots with annotations
Indicators of compromise	Include screenshots, files, text, network indicators, strings, and other indicators of compromise.
Host-based indicators of compromise	Include screenshots of the desktop changing or any files in the windows directory changing or any other changes that the normal user would find. (Obvious changes)
Rules & Signatures	Include specific attributes of the malware such as strings URLs or other kinds of signatures that this specific kind of malware would have. Also mention that a full set of rules are included in the appendix.
Appendices	Label the appendix by letters with screenshots or content under them. If any code is in the appendix use code formatting. Categories can be your rules, call back URLS, decompiled code snippets, and any other info. Label the figure under each entry. "*Fig 1: Code in cutter*"