SIGPwny

SP2025 Week 06 • 2025-03-11

Reverse Engineering Antivirus for Evasion

Ronan Boyarski

Overview

- Brief review of theoretical model of AV & EDR
- Case studies from Protections-Artifacts and Windows Defender
    - Avoiding artifact detection
    - Bypassing behavioral detection
    - Avoiding process tree detection

Antivirus Detection Methods

- Signature based detection
  - Whether a file looks malicious statically, through sequences of bits or the file hash
  - Basic and easy to evade with polymorphism & encryption
- Behavioral detection
  - The AV will run the file in a sandbox and see what it does, and judge intelligently if the activity is malicious
  - Reversing these sandboxes is basically impossible
  - We can however use certain methods to tell if we're in a sandbox and alter our execution depending on where we are
    - We can even tell if we're in a debugger, and use that as an opportunity to troll reverse engineers

Antivirus Detection Methods

- Heuristic detection
  - Uses big data & AI/ML to see how suspicious a file looks statically
  - Not good against low-level malware (written in C & ASM), but highly effective against C#
- Command Line
  - Flags against known LOLBAS
- Process Tree
  - If Microsoft Word is running PowerShell commands, something has gone horribly wrong

Where We Left Off

- Use a polymorphic loader to get initial execution
- Problem: what happens once our malware is sitting in memory?
  - C2 frameworks will have sleep obfuscation, modules will not
- Problem: Even if we get execution, what if we do something that could be flagged?
  - For example, when are we able to read LSASS process memory safely?
  - How do we know which processes get to use Kerberos and get away with it?

Reversing

Static Signatures

- Need to **extract** the database
    - Elastic signatures are here
    - Windows Defender requires dumping & decompressing the database
        - Also found here
- Usually this will be in YARA format
    - Designed around pattern matching and boolean expressions

```
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
        threat_level = 3
        in_the_wild = true

    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

    condition:
        $a or $b or $c
}
```

Static Signatures

- How would you go about making this return false?

```
rule Windows_Hacktool_SharpUp_e5c87c9a {
    meta:
        author = "Elastic Security"
        id = "e5c87c9a-6b4d-49af-85d1-6bb60123c057"
        fingerprint = "4c6e70b7ce3eb3fc05966af6c3847f4b7282059e05c089c20f39f226efb9bf87"
        creation_date = "2022-10-20"
        last_modified = "2022-11-24"
        threat_name = "Windows.Hacktool.SharpUp"
        reference_sample = "45e92b991b3633b446473115f97366d9f35acd446d00cd4a05981a056660ad27"
        severity = 100
        arch_context = "x86"
        scan_context = "file, memory"
        license = "Elastic License v2"
        os = "windows"
    strings:
        $guid = "FDD654F5-5C54-4D93-BF8E-FAF11B00E3E9" ascii wide nocase
        $str0 = "^\\W*([a-z]:\\\\.+?(\\.exe|\\.bat|\\.ps1|\\.vbs))\\W*" ascii wide
        $str1 = "^\\W*([a-z]:\\\\.+?(\\.exe|\\.dll|\\.sys))\\W*" ascii wide
        $str2 = "SELECT * FROM win32_service WHERE Name LIKE '{0}'" ascii wide
        $print_str1 = "[!] Modifialbe scheduled tasks were not evaluated due to permissions." ascii wide
        $print_str2 = "[+] Potenatially Hijackable DLL: {0}" ascii wide
        $print_str3 = "Registry AutoLogon Found" ascii wide
    condition:
        $guid or (all of ($str*) and 1 of ($print_str*))
}
```

Static Signatures

- How would you go about making this return false?
  - Change the GUID
  - Change the ordering of .exe|.bat, or similar
  - Rename each print string

```
rule Windows_Hacktool_SharpUp_e5c87c9a {
    meta:
        author = "Elastic Security"
        id = "e5c87c9a-6b4d-49af-85d1-6bb60123c057"
        fingerprint = "4c6e70b7ce3eb3fc05966af6c3847f4b7282059e05c089c20f39f226efb9bf87"
        creation_date = "2022-10-20"
        last_modified = "2022-11-24"
        threat_name = "Windows.Hacktool.SharpUp"
        reference_sample = "45e92b991b3633b446473115f97366d9f35acd446d00cd4a05981a056660ad27"
        severity = 100
        arch_context = "x86"
        scan_context = "file, memory"
        license = "Elastic License v2"
        os = "windows"
    strings:
        $guid = "FDD654F5-5C54-4D93-BF8E-FAF11B00E3E9" ascii wide nocase
        $str0 = "^\\W*([a-z]:\\\\.+?(\\.exe|\\.bat|\\.ps1|\\.vbs))\\W*" ascii wide
        $str1 = "^\\W*([a-z]:\\\\.+?(\\.exe|\\.dll|\\.sys))\\W*" ascii wide
        $str2 = "SELECT * FROM win32_service WHERE Name LIKE '{0}'" ascii wide
        $print_str1 = "[!] Modifialbe scheduled tasks were not evaluated due to permissions." ascii wide
        $print_str2 = "[+] Potenatially Hijackable DLL: {0}" ascii wide
        $print_str3 = "Registry AutoLogon Found" ascii wide
    condition:
        $guid or (all of ($str*) and 1 of ($print_str*))
}
```

Static Signatures

- How would you go about making this return false?
  - Change the GUID
  - Change the ordering of .exe|.bat, or similar
  - Rename each print string
- How would you test that it worked specifically from the change you made?

```
rule Windows_Hacktool_SharpUp_e5c87c9a {
    meta:
        author = "Elastic Security"
        id = "e5c87c9a-6b4d-49af-85d1-6bb60123c057"
        fingerprint = "4c6e70b7ce3eb3fc05966af6c3847f4b7282059e05c089c20f39f226efb9bf87"
        creation_date = "2022-10-20"
        last_modified = "2022-11-24"
        threat_name = "Windows.Hacktool.SharpUp"
        reference_sample = "45e92b991b3633b446473115f97366d9f35acd446d00cd4a05981a056660ad27"
        severity = 100
        arch_context = "x86"
        scan_context = "file, memory"
        license = "Elastic License v2"
        os = "windows"
    strings:
        $guid = "FDD654F5-5C54-4D93-BF8E-FAF11B00E3E9" ascii wide nocase
        $str0 = "^\\W*([a-z]:\\\\.+?(\\.exe|\\.bat|\\.ps1|\\.vbs))\\W*" ascii wide
        $str1 = "^\\W*([a-z]:\\\\.+?(\\.exe|\\.dll|\\.sys))\\W*" ascii wide
        $str2 = "SELECT * FROM win32_service WHERE Name LIKE '{0}'" ascii wide
        $print_str1 = "[!] Modifialbe scheduled tasks were not evaluated due to permissions." ascii wide
        $print_str2 = "[+] Potenatially Hijackable DLL: {0}" ascii wide
        $print_str3 = "Registry AutoLogon Found" ascii wide
    condition:
        $guid or (all of ($str*) and 1 of ($print_str*))
}
```

Detection Unit Testing - YARA

- We can test individual YARA detections with yara64.exe
- Example: Testing for an injected SharpUp into a Havoc agent

```
PS C:\Users\Robert Banks\Desktop> .\yara64.exe .\SharpUp.yar 7036
Windows_Hacktool_SharpUp_e5c87c9a 7036
```

- We can use this as a unit test to see if our malware works

Detection Unit Testing - YARA

- We can test individual YARA detections with yara64.exe
- Example: Testing for an injected SharpUp into a Havoc agent

```
PS C:\Users\Robert Banks\Desktop> .\yara64.exe .\SharpUp.yar 7036
Windows_Hacktool_SharpUp_e5c87c9a 7036
```

- We can use this as a unit test to see if our malware works

Detection Unit Testing - YARA

- We can test individual YARA detections with yara64.exe
- Example: Testing for an injected SharpUp into a Havoc agent

```
PS C:\Users\Robert Banks\Desktop> .\yara64.exe .\SharpUp.yar 7036
Windows_Hacktool_SharpUp_e5c87c9a 7036
```

- We can use this as a unit test to see if our malware works

Detection Unit Testing - YARA

- Change the GUID (in AssemblyInfo.cs)

```
// The following GUID is for the ID of the typelib if this project is exposed to COM
[assembly: Guid("feadf084-a02a-4553-b755-076f382869a8")]
```

- Shuffle one of the signatured regexes

```
@"^\W*([a-z]:\\.+?(\.dll|\.exe|\.sys))\W
```

- Change each print_str

```
Console.WriteLine("[!] Modifiable scheduled tasks were not evaluated due
```

```
Console.WriteLine("[+] Potentially Hijackable DLL: {0}\n" +
```

```
Console.WriteLine("Found Registry AutoLogon\r\n");
```

- Success!

```
PS C:\Users\Robert Banks\Desktop> .\yara64.exe .\SharpUp.yar 9540
PS C:\Users\Robert Banks\Desktop> _
```

Detection Unit Testing - YARA

- We can automatically perform binary search with ThreatCheck
- Even though you should be searching for this stuff with reverse engineering, if you're in a pinch this can work *sometimes*
- Goes well with Ghidra to go see if certain assembly is being flagged

```
PS C:\Tools\ThreatCheck> .\ThreatCheck.exe -f C:\Payloads\http_x64.exe
[+] Target file size: 315392 bytes
[+] Analyzing...
[!] Identified end of bad bytes at offset 0x9E1
00000000   C3 66 66 2E 0F 1F 84 00  00 00 00 00 0F 1F 00 48   Ãff.··?·······H
00000010   83 EC 28 48 8B 05 C5 BF  04 00 C7 00 00 00 00 00   ?ì(H?·Å¿··Ç·····
00000020   E8 8A 04 00 00 E8 75 FC  FF FF 90 90 48 83 C4 28   è?···èuüÿÿ??H?Ä(
00000030   C3 66 66 2E 0F 1F 84 00  00 00 00 00 0F 1F 00 48   Ãff.··?·······H
00000040   83 EC 28 E8 4F 19 00 00  48 85 C0 0F 94 C0 0F B6   ?ì(èO···H?À·?À·¶
00000050   C0 F7 D8 48 83 C4 28 C3  90 90 90 90 90 90 90 48   À÷ØH?Ä(Ã???????H
00000060   8D 0D 09 00 00 00 E9 D4  FF FF FF 0F 1F 40 00 C3   ?·····éÔÿÿÿ··@·Ã
00000070   90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 48   ???????????????H
00000080   FF E1 48 63 05 C6 2A 00  00 85 C0 7E 26 83 3D BF   ÿáHc·Æ*··?À~&?=¿
00000090   2A 00 00 00 7E 1D 48 8B  15 06 FD 04 00 48 89 14   *···~·H?··ý··H?·
000000A0   01 48 8B 15 03 FD 04 00  48 63 05 A4 2A 00 00 48   ·H?··ý··Hc·¤*··H
000000B0   89 14 01 C3 41 54 55 57  56 53 48 83 EC 40 41 B9   ?··ÃATUWVSH?ì@A¹
000000C0   04 00 00 00 4C 63 E2 48  89 CF 4C 89 C5 31 C9 41   ····LcâH?ÏL?Å1ÉA
000000D0   B8 00 30 00 00 4C 89 E2  4C 89 E6 FF 15 52 FD 04   ¸·0··L?âL?æÿ·Rý·
000000E0   00 48 89 C3 31 C0 39 C6  7E 15 48 89 C2 83 E2 07   ·H?Ã1À9Æ~·H?Â?â·
000000F0   8A 54 15 00 32 14 07 88  14 03 48 FF C0 EB E7 48   ?T··2··?··HÿÀëçH
```

Bypassing Behavioral Detections

- Behavioral detections are quite robust
  - Usually triggered by Kernel Callbacks and can introspect into ETW-TI
  - ETW-TI: Event Tracing for Windows Threat-Intelligence
    - A Kernel-Mode subscription for security-relevant events
    - Can capture essential data about processes and threads, among other things
- How can we get around a robust behavioral detection when we can't tamper with the incoming data?
  - Assume we do not have kernel code execution

Bypassing Behavioral Detections

- Behavioral detections are quite robust
  - Usually triggered by Kernel Callbacks and can introspect into ETW-TI
  - ETW-TI: Event Tracing for Windows Threat-Intelligence
    - A Kernel-Mode subscription for security-relevant events
    - Can capture essential data about processes and threads, among other things
- How can we get around a robust behavioral detection when we can't tamper with the incoming data?
  - Assume we do not have kernel code execution
  - Exclusions!

Bypassing Behavioral Detections

- Go take a look at this - a good detection for PPID spoofing

```
[rule]
description = """
Identifies parent process spoofing used to thwart detection. Adversaries may spoof the parent process identifier (PPID)
of a new process to evade process-monitoring defenses or to elevate privileges.
"""
id = "816ba7e7-519a-4f85-be2a-bacd6ccde57f"
license = "Elastic License v2"
name = "Parent Process PID Spoofing"
os_list = ["windows"]
reference = [
    "https://blog.didierstevens.com/2017/03/20/",
    "https://www.elastic.co/security-labs/elastic-security-labs-steps-through-the-r77-rootkit",
]
version = "1.0.46"

query = '''
sequence with maxspan=5m
 [process where event.action == "start" and
  process.parent.executable != null and
```

Bypassing Behavioral Detections

- Take a moment to read the whole rule, look at line 120

```
not process.executable : ("?:\\Windows\\SysWOW64\\WerFault.exe", "?:\\Windows\\system32\\WerFault.exe")
```

- What does line 120 do?

Bypassing Behavioral Detections

- Take a moment to read the whole rule, look at line 120

```
not process.executable : ("?:\\Windows\\SysWOW64\\WerFault.exe", "?:\\Windows\\system32\\WerFault.exe")
```

- What does line 120 do?
    - If the current EXE is called WerFault, we return false, meaning **the PPID spoofing check is ignored**
- Practically, this means that if we spawn and inject (aka fork and run) into WerFault, we can PPID spoof into any process on the system, **bypassing the detection**
- This is unlikely to get fixed in the near future

Bypassing Process Trees

- Take a look at this rule for detecting unbacked LSASS dump (think nanodump)
- Do any of these exclusions jump out at you as being exploitable?
  - Remember, we can control our parent process, current process (spawn and inject), and command line arguments

Bypassing Process Trees

- Take a look at this rule for detecting unbacked LSASS dump (think nanodump)
- Do any of these exclusions jump out at you as being exploitable?
  - Remember, we can control our parent process, current process (spawn and inject), and command line arguments

```
not (process.executable : "?:\\Windows\\system32\\netstat.exe" and user.id : "S-1-5-18" and process.args : ("-a", "/a")) and
not (process.executable : "?:\\Windows\\system32\\tasklist.exe" and process.args : "/M") and
```

- One tried and tested tactic is to inject into netstat to dump LSASS
  - If you look at the if statement, we need to be running as SYSTEM, and supply a "-a" somewhere into the argument

How to Inject into Netstat?

- If you're not super familiar with running a modern C2, you can actually do this with no code!
- C2's like Cobalt Strike and Havoc will have support for setting a **spawnto**
  - When we process inject, what process do we create?
  - Cobalt Strike has support for spoofing the command line args
    - This isn't too hard to write on your own
  - So, usually you would run something like
    - **set spawnto C:\Windows\System32\netstat.exe**
    - **argue -a**
  - Then, we would be able to dump LSASS

How did we get here?

Case Study: Defender

- Defender comes with a handful of files, including two **.vdm** files
- mpasbase.vdm is the Anti-spyware database
- mpavbase.vdm is the Antivirus database
- mpasdlta.vdm is the AntiSpyware recent changes ("delta") database
- mpavdlta.vdm is the AntiVirus recent changes ("delta") database
- This repo goes over the reversing process of figuring out what these are
- Then, further research went into extracting individual signatures, which lets us get at things like ASR rules https://github.com/hfiref0x/WDExtract
- Finally, someone decided to convert many of them into YARA rules
- Note that none of this is exhaustive - Defender still has some signatures that haven't been analyzed / extracted