CS-501

Introduction to Malware, Threat Hunting & Offensive Capabilities Development

Lecture 23: Hooking

Agenda for Today

Import Address Table Hooking: how to intercept and modify call to functions declared in the PE in both local and remote processes

How to detect Import Address table Hooks.

Spongebob memes that you better laugh at. Reminder, I control your grade, and am a capricious god.

MARIO
058700
×20
WORLD
1-4
TIME
200
Thank you Mario but the address of MessageBoxA is in another castle!

Review: hooking

Hooking involves intercepting function calls, inspecting/modifying the contents, and then (optionally) passing execution to the original function

Definitions

- Hooking a function: intercepting a call to a function
- Hook (or Hook handler): the function that receives the arguments to the hooked function, and optionally passes execution back to the hooked function

Some reasons to hook functions

- Game hacking/cheating
- Information stealing
- Anti Malware tasks
- Debugging

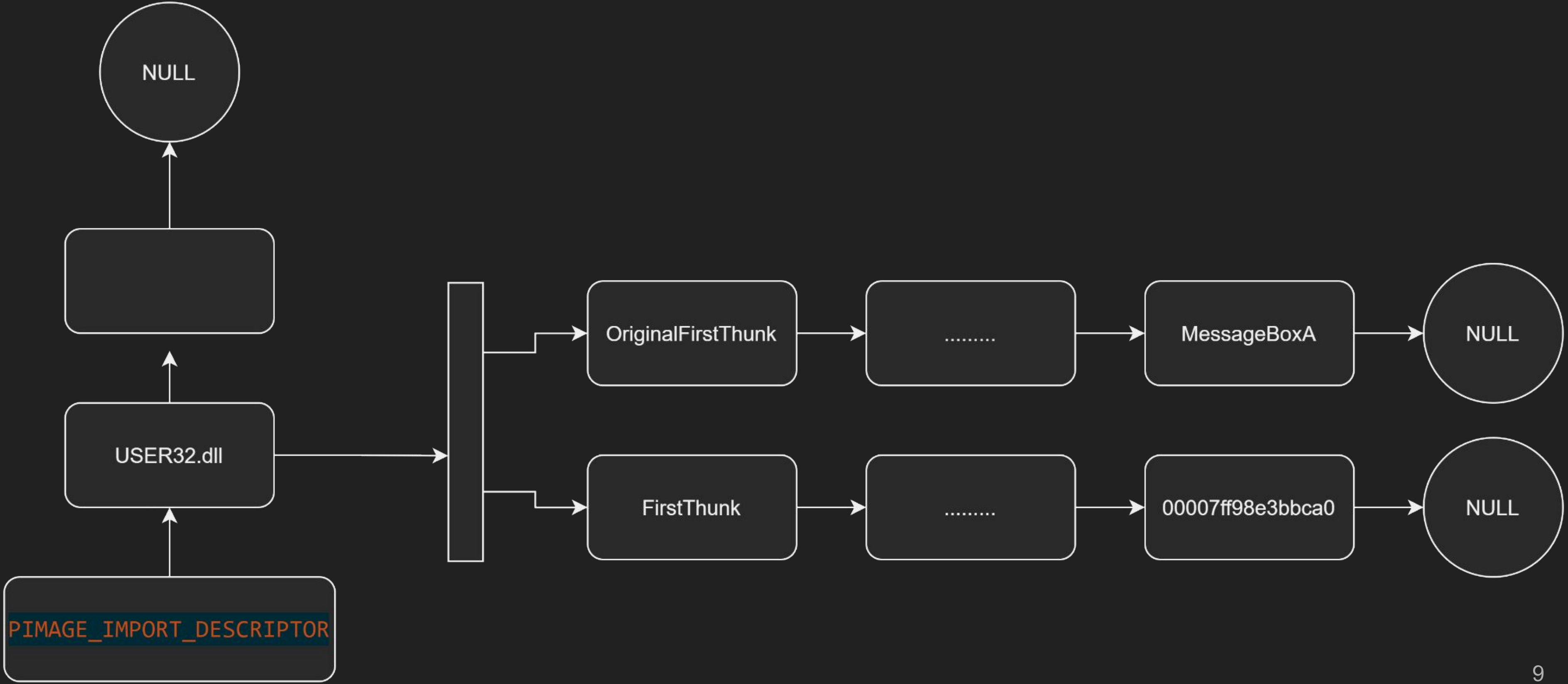
Review PE File Format

- PEs exist on disk as raw bytes, and are not memory mapped
- The DOS header is the start of the PE, and is used by us to find the offset of the NtHeaders. In this case, Offset = RVA, so we can find the start of the NT Headers and retrieve the rest of the information we need
- The NT Headers contain the RVA of the Optional headers
- The Optional Headers Contains the Data Directories

Image Imports Descriptors

- The Import Directory (1 `IMAGE_DIRECTORY_ENTRY_IMPORT`) contains a null terminated Array of Image Import Descriptors
- Each Image Import Descriptor has a field Name which is an RVA to a null terminated string corresponding to a DLL that exists on Disk
- There are also two NULL terminated arrays of Function names, and function pointers
- We can walk these two in parallel to find the function's name in the library, and its VA

Review: Import Address table

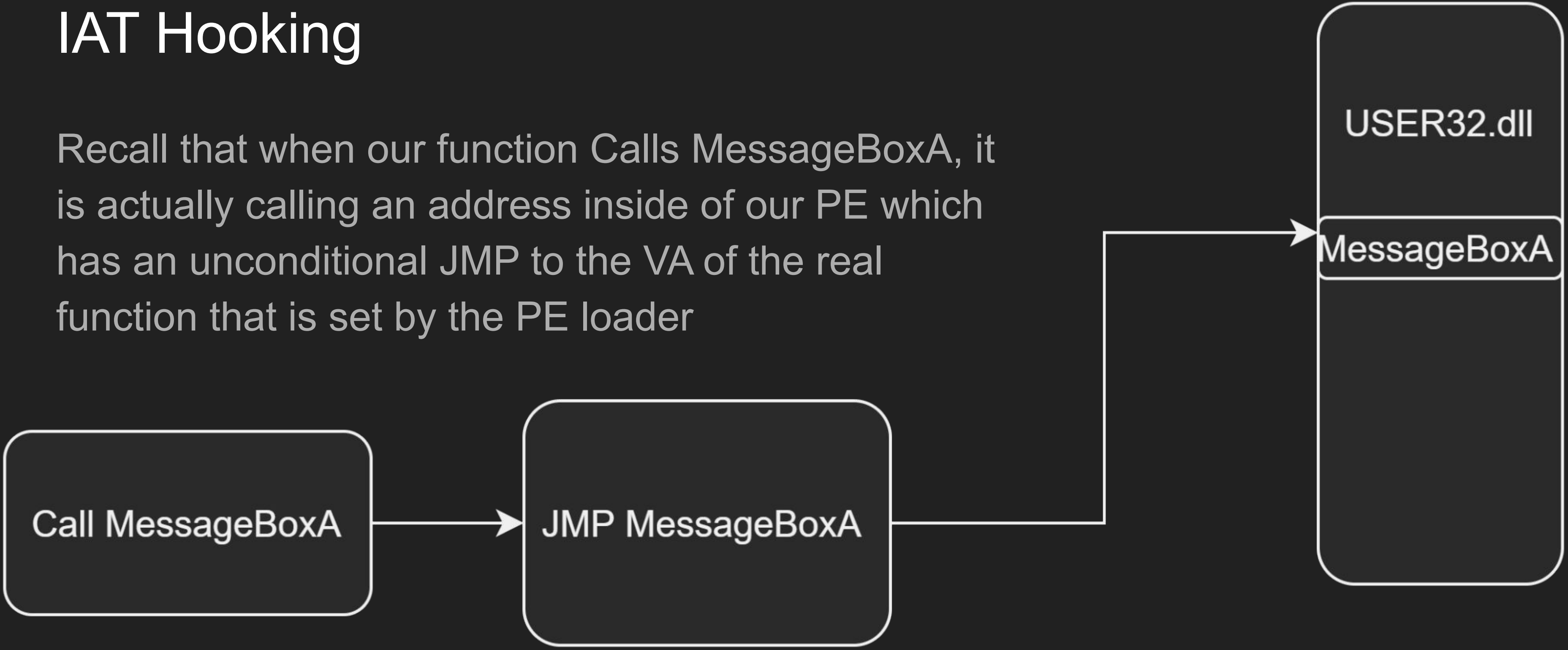


Hooking Import Address Table

- IAT hooking involves overwriting the function pointer in the IAT with a function that we control
- Using this, we can inspect and modify the arguments to whatever end
- Many AV solutions will hook a select number of functions to try and detect suspicious behavior
- IAT hooking is generally very, stable and very easy to implement

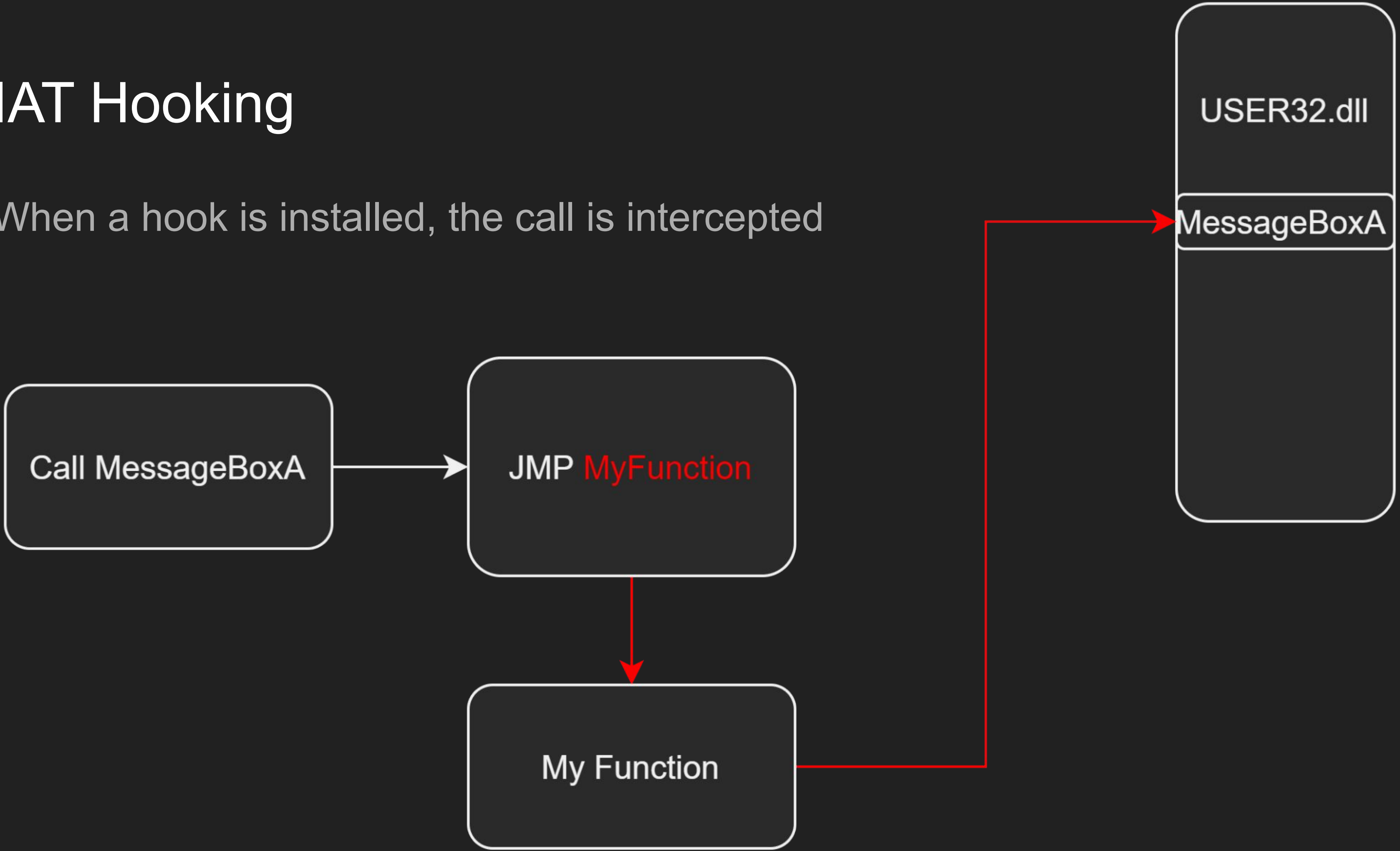
IAT Hooking

Recall that when our function Calls MessageBoxA, it is actually calling an address inside of our PE which has an unconditional JMP to the VA of the real function that is set by the PE loader



IAT Hooking

When a hook is installed, the call is intercepted



Implementation

- We can reuse our code from the PE loader to walk the list of imported libraries, and then the pair of arrays that contain function names, and function pointers
- If we want to hook user32.dll$MessageBoxA, we would first find the base address of the target loaded executable PE.
- Next, we parse its headers and walk the imports
- If we come across MessageBoxA while walking the imports for User32.dll, we can overwrite the function pointer!

Implementation Details

1) Get the base virtual address of the target PE
2) Get the Address of the function we will use to overwrite the entry in the IAT
    a) I.e., the address of the hook handler
3) Walk imports, and once you find the function of interest, modify its memory protection to allow for writing
4) Overwrite the current entry with the address of the user controlled function
5) Profit

Step 1)

- We can get the base address of the current PE by calling `GetModuleHandleA`
- If we are in a remote processes, once we have the base address of Kernel32.dll, we can walk the exports until we find `GetModuleHandleA`.

“If this parameter is NULL, GetModuleHandle returns a handle to the file used to create the calling process (.exe file).”

https://docs.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-getmodulehandlea

Step 2

- If we want to overwrite the entry in the IAT with the address of MyFunc, simple get the address by casting the function to a pointer

  I.e.,

  ```
  UINT_PTR lpFuncAddr = (UINT_PTR) MyFunc;
  ```

Step 3

- We need to walk the imports of the target PE
- Once we find the import matching the target function in the correct library, we can save the original VA of the imported function
    - We need to save this so we can pass execution to the real function later
- The section of the PE containing the IAT is NOT writeable. We need to remedy this with two calls to VirtualProtect
    - One to add write permissions
    - One to take it away once we are done

Step 4

Just change the value stored in the function pointer to the address of the function we control

Example:
Hooking (and unHooking) MessageBoxA

Common Footguns

- Case matters for dlls and function names
- Saving the VA address of the Hooked function’s entry in the IAT instead of the VA of the actual function. Hope you like recursion and exploding stacks
- Race conditions for when the hook is set, and the original function address saved

UnHooking

- Same thing. You hook the hooks :)
- This is a common defense evasion tactic-- if an AV hooks functions in your processes, just hook their hooks!

Legitimate Reasons to Hook Functions

A common strategy for detecting processes injection is to hook various functions commonly used to perform process injection

For example, you could hook VirtualAllocEx, and CreateRemoteThread to try and identify reflective DLL injection

Example: Ch0nkyhook

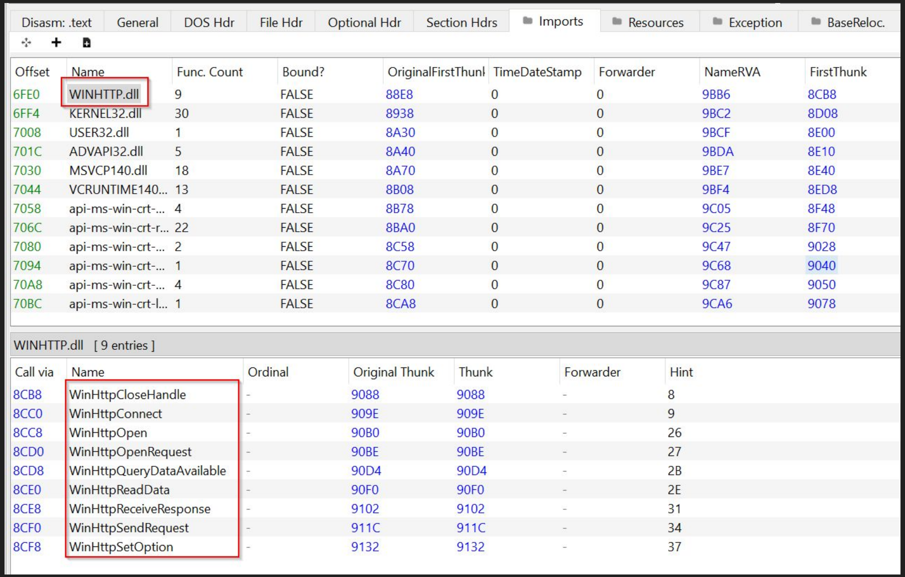
Suppose you want to develop countermeasure for Ch0nky Loader.

This could be a solution you deploy on your network, or in an analysis environment to speed up triage.

Example: Ch0nkyhook

After analyzing Ch0nky Loader, you determine the C2 Channel is HTTPS with WinHTTP as the client library.

You also notice that it makes many long calls to Sleep during execution.



Offset	Name	Func. Count	Bound?	OriginalFirstThunk	TimeDateStamp	Forwarder	NameRVA	FirstThunk
6FE0	WINHTTP.dll	9	FALSE	88E8	0	0	9BB6	8CB8
6FF4	KERNEL32.dll	30	FALSE	8938	0	0	9BC2	8D08
7008	USER32.dll	1	FALSE	8A30	0	0	9BCF	8E00
701C	ADVAPI32.dll	5	FALSE	8A40	0	0	9BDA	8E10
7030	MSVCP140.dll	18	FALSE	8A70	0	0	9BE7	8E40
7044	VCRUNTIME140...	13	FALSE	8B08	0	0	9BF4	8ED8
7058	api-ms-win-crt-...	4	FALSE	8B78	0	0	9C05	8F48
706C	api-ms-win-crt-r...	22	FALSE	8BA0	0	0	9C25	8F70
7080	api-ms-win-crt-...	2	FALSE	8C58	0	0	9C47	9028
7094	api-ms-win-crt-...	1	FALSE	8C70	0	0	9C68	9040
70A8	api-ms-win-crt-...	4	FALSE	8C80	0	0	9C87	9050
70BC	api-ms-win-crt-l...	1	FALSE	8CA8	0	0	9CA6	9078

WINHTTP.dll [9 entries]

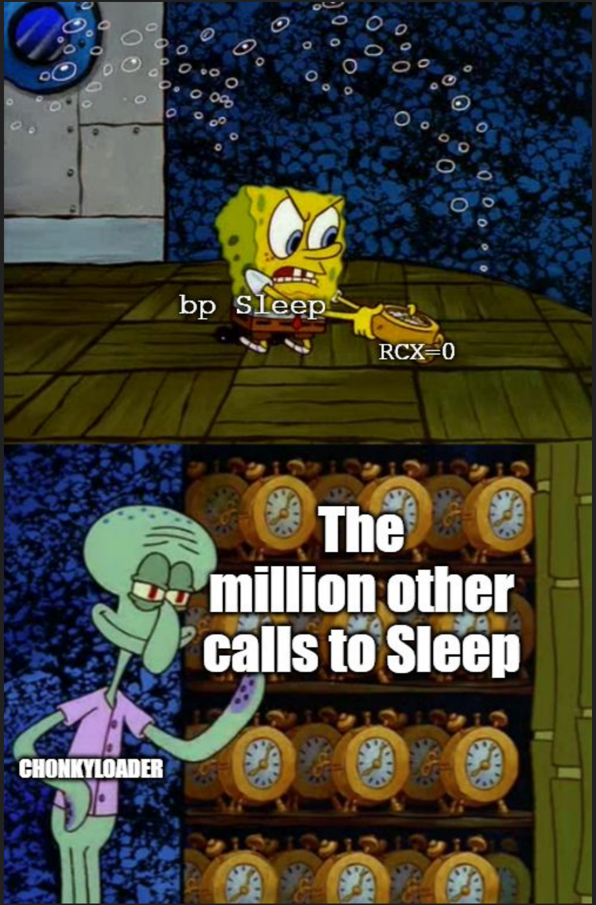
Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
8CB8	WinHttpCloseHandle	-	9088	9088	-	8
8CC0	WinHttpConnect	-	909E	909E	-	9
8CC8	WinHttpOpen	-	90B0	90B0	-	26
8CD0	WinHttpOpenRequest	-	90BE	90BE	-	27
8CD8	WinHttpQueryDataAvailable	-	90D4	90D4	-	2B
8CE0	WinHttpReadData	-	90F0	90F0	-	2E
8CE8	WinHttpReceiveResponse	-	9102	9102	-	31
8CF0	WinHttpSendRequest	-	911C	911C	-	34
8CF8	WinHttpSetOption	-	9132	9132	-	37

Example: Hooking Sleep

Ch0nky Loader will periodically sleep for 60 seconds.

Our current strategy has been to set a breakpoint at Sleep and change the argument to 0

This of course would be way more annoying if instead the malware made 60 calls to Sleep



Hooking Sleep

We can write a simple dll that hooks the IAT for the running processes, hooks the call to Sleep, and sets the value to a smaller amount!

(note x64dbg can do this for you as well)

Example: Hooking Ch0nkyLoader

We can hook the function that sends data back to the C2

As it uses WinHttpSendRequest, we can hook the hook the function, and scan for references to “ch0nky”

We should take the trampoline for WinHttpSendRequest
AND PUSH IT SOMEWHERE ELSE

Hooking: Remote processes

One strategy is to leverage (Reflective) DLL injection to inject a DLL into a remote process

The DLL will be responsible for hooking the target functions, and (optionally) reporting back the results to the calling process

This means the DLL needs to have a function to handle calls from the hooked function,

Limitations

- IAT hooking is incredibly stable, BUT it can only hook functions that exist in the IAT
- If a processes explicitly links its imports (i.e., LoadLibraryA→ GetProceAddress) then IAT hooking will fail
-

Detecting IAT Hooks

- Easy(ish): in simple cases, we can walk the IAT and compare the function address listed and the value of LoadLibraryA→ GetProcAddress
- We can look for changes in memory protections for read only sections of memory

Detecting Hooks in General

- You need to know what the unhooked function looks like
    - We can map the PE into memory, and compare the content of the clean function, to a possibly hooked function
- Alternatively, you could leverage heuristics for what the execution flow should look like
    - For example, kernel32.dll is loaded into a static location, and will usually call into ntdll.dll. If the function is directed to something in user code, that is sus

Next time: Inline Hooking

- How to trampoline hook
- How to detect AV hooks in common DLLs