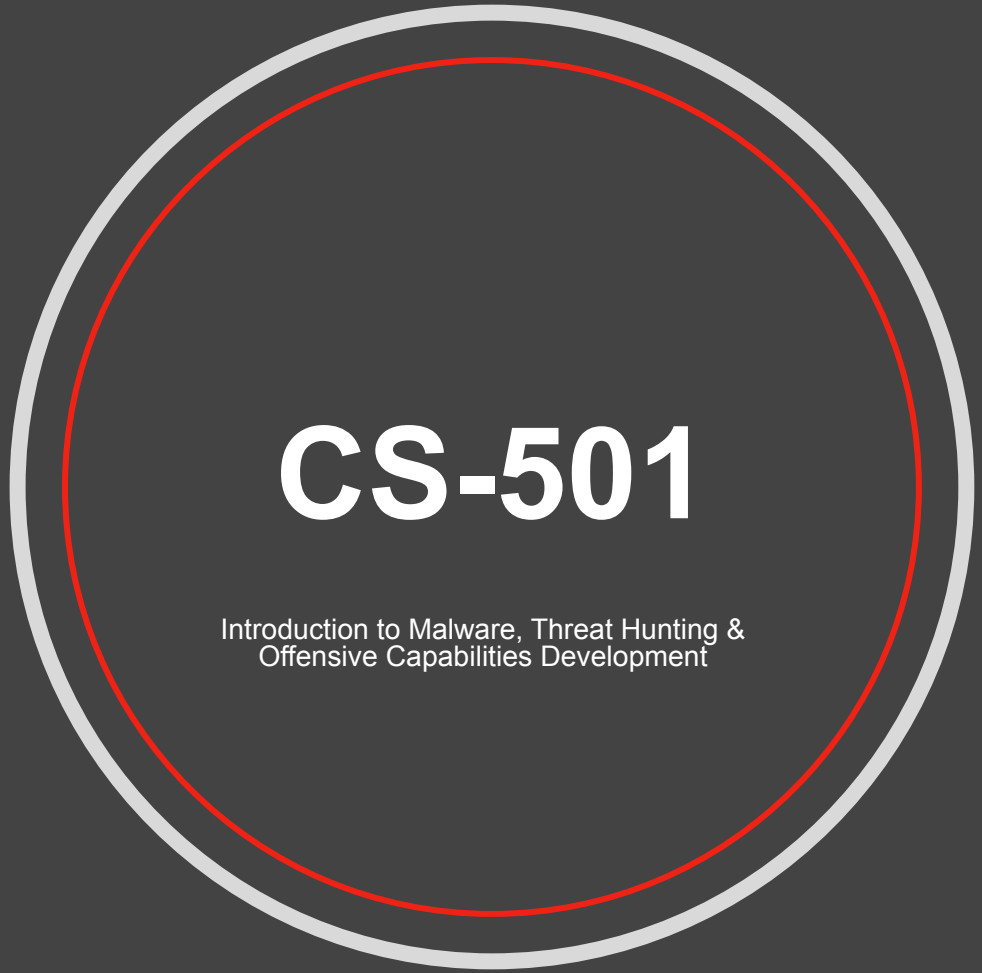

CS-501

Introduction to Malware, Threat Hunting &
Offensive Capabilities Development

Lecture 14: Adversary Simulation, Intro Hooking

Homework 3 is postponed until Monday

Based on the homework, I want to fill in some blanks before we dive into more complex technical topics

Let's talk about Homework 2

Common Mistakes:

- Uninitialized Variables
- Lack of Error Checking
- Returning stack variables
- Failure to call free/delete on allocated data

C/C++ don’t stop you from shooting yourself in the Foot

I recommend compiling all of your code with -Wall

Going forward, I will include that flag in the Makefile.

Uninitialized Memory

Uninitialized Memory

- An uninitialized variable is one that is declared, with no value set
  - For example, DWORD x;
- Depending on the compiler, the contents of x could be completely random
- This will lead to unexpected behavior

Memory Leaks

- Allocating memory on the heap (malloc/new) without freeing it leather
- This memory is no longer usable for allocation and from the perspective of the processes is used
- Some compilers/runtimes will detect when allocations have gone out of scope
- You should not rely on this. Anytime you call new/malloc, you should also call delete/free
-

Returning stack variables

- If we have a pointer to memory in a stack frame, when we exit the function that memory might no longer be valid
- By accessing it directly after the function it *might* look OK, but this is incredibly dangerous
- In particular, the process will reclaim this memory and it will likely be filled with garbage.

Asking Questions:

It is my job to answer your questions! I will answer any and all questions that you ask

A few observations from questions I have received:

- You can save yourself hours of debugging by carefully reading the documentation
- If something isn’t working the way you expect it to, attach the process to a debugger, open it up in ghidra. Read the code
- Ask your classmates in the group discord! It is possible someone has already encountered this
- This is a reverse engineering class. Use a debugger!!

Demo: Final Payload

- Look at the strings
- Look at it in Ghidra
- Look at the imports
- Think about where to set breakpoints

Hooking

Hooking (in a general sense) is a way to intercept function calls, inspect or modify the calling behavior, then forward the call.

There are many legitimate uses for Hooking. For example, a debugger hooks functions by setting breakpoints

Malware authors use hooking to intercept calls to functions to (among other things)

1) Reading data
2) Modify data

Example: Read sensitive information

- Reading data from Chrome before an HTTP request is sent
- Reading the contents of a document
- Reading the contents Keepass memory when the vault is unlocked

Modifying Data

- Hiding the existence of a process/pipe/file/socket
- Modify bitcoin wallet address (replace legitimate recipient with malicious one)

Hooking: Using Win32 Legitimately

SetWindowsHookEx allows us to intercept function calls for a select number of messaging events. For example, keystrokes.

SetWindowsHookEx

```
HHOOK SetWindowsHookEx(
  [in] int       idHook,
  [in] HOOKPROC  lpfn,
  [in] HINSTANCE hmod,
  [in] DWORD     dwThreadId
);
```

Cookbook:

1) Get a handle to your module (LoadLibrary)
2) Find the function that installs hooks (GetProcAddress)
3) Get a handle to the remote process you wish to install hooks in
4) Find the relevant thread
5) INstall your hooks
6) handle callbacks in some way (SetNotificationThread, Files, Named Pipes...etc)
7) Intercept Arguments and forward them along

Interesting idHook

WH_GETMESSAGE

WH_KEYBOARD/WH_KEYBOARDLL

Using WH_GETMESSAGE

- Intercept keystrokes to a GUI application

Demo