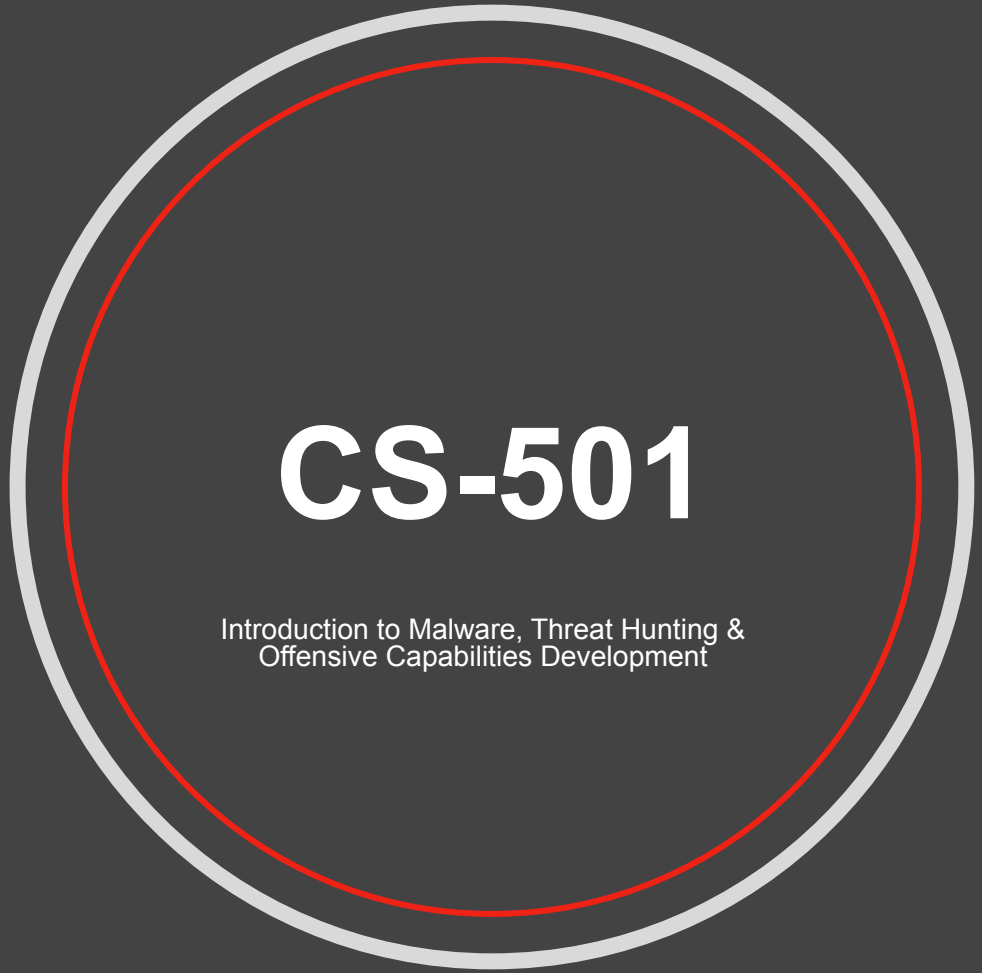

CS-501

Introduction to Malware, Threat Hunting &
Offensive Capabilities Development

Lecture 16: Capstone, Packers and Injection

Academic integrity and Mental Health

TLDR

I give incompletes.

I am not a cop.

I give extensions.

If you tell me you need an incomplete, I will give it to you no questions asked. You do not need to explain your specific situation if you don’t have the emotional energy to do so.

Submitting identical code where it is not expressly permitted is forbidden.

This is your last warning

Capstone: Developing your own C2 Framework

To pass this class, you must develop a C2 Framework. This lecture aims to spell out the functional requirements of the C2 Server, the client, and the Implant

You may work in groups of up to 5 people.

You may work alone. It is not recommended.

You must use a private git repository (either github or gitlab) that You add my account to. This is in part how I will observe group contributions.

You should probably get started right away. Your homework is relevant to the capstone :)

Terminology: Review

C2: Command and Control Server

Implant: the malware “implanted” on a victim machine

Client: the software that allows the malware operators to control an implant by communicating with the team server

General Comments

Course staff will directly support students who

1) Write their C2 Server and Client in python
2) The Backend Message Broker with RabbitMQ, Redis, or ZeroMQ
3) The client UI with prompt toolkit
4) Implant in C/C++

You are welcome to use any language you want for the C2 server and client.

For the implant, your options are:

C, C++, Golang, Erlang, Nim, Zig, Rust, C#, Assembly

You may not use **python for the implant.**

General Requirements: C2

Your Command and Control Server needs to be able to handle connections from multiple operators, and multiple implants.

You may accomplish this anyway you want! But course staff recommends:

1) Using Flask as the primary listener with Gunicorn as the WSGI (flask is not production ready on its own)
2) Using Postgres or MySQl as the backend Database
3) Using flask-SqlAlchemy to facilitate CRUD operations on agents, and operators
4) Using RabbitMQ/ZeroMQ/Redis to broker messages between Implant and C2, and client and C2.

C2->Database

Your database should should have several tables that correspond to different object models

1) Implants: More on next slide
2) Commands : Keep track of which operators issued what command
3) Jobs: Keep track of jobs sent to implants that are in progress/finished
4) Clients: Keep track of operators connected to the C2 via the client

C2->Database->Implants

Implant ID: Create an ID for the implant to distinguish is from others

Computer Name: What computer did it connect from?

Username: What user are you running as?

GUID: What is the computer's GUID?

Integrity: What privileges do you have?

Connecting IP address: what address did it connect from?

Session Key: After you negotiated a session key, store it per agent

Sleep: How often does the agent check in?

Jitter: How random of a check in is it?

First Seen: When did the agent first check in?

Last Seen: When was the the last time you saw the agent?

Expected Check in: When should you expect to see the agent again?

Messaging

Operators should be notified whenever one of the following occurs:

1) A new implant connects to the C2 for the first time
2) A client connects to the C2
3) An operator issues a command to an agent (current operator only)
4) An agent responds to a job that was issued by an operator

RabbitMQ Pub Sub is probably the easiest way to handle this

Your client code can be run on the same box as the C2. You may assume that you have SSH Access to your C2 server.

Another option would be to implement an endpoint to query information over HTTP

Implant Functionality

Stager: A small payload that loads the final payload from the C2. this should leverage an injection technique to be discussed later. Either Reflective DLL injection, PE Injection, or shellcode injection. You may use a library for this, but you must understand all code you submit. And site all sources

Implant Functionality

- Cryptography
- Situational awareness
- Modify Config
- Execution
- Loot
- Persistence
- File I/O
- Defense Evasion
- RPC and C2 Channel
- *Special Feature*

Implant->Cryptography

Your implant must use **secure** cryptography.

Asymmetric:

- Implant to C2: Your implant must use asymmetric cryptography to establish a secure session to the C2.
- This could be a key exchange algorithm to agree on a key or could leverage public crypto by embedding the public key in the implant, and encrypting the session key as part of the handshake with the server.

Symmetric:

- Your implant must encrypt all relevant configuration, and strings. This could be something as simple as a XOR cipher, but you can get as creative as you want here. The goal of string “encryption” is to slow down the reverse engineer.
- Your implant must use authenticated crypto to communicate with the server if your C2 Channel is HTTPs. Example: AES-GCM (recommended)

Hash Functions:

- You must tailor your payload to a device that has the C:\malware\ch0nky.txt file path using a secure hash function. More on this later
- Bonus: have the stager report back the MAC address or CPU ID or other identifying information and tailor the payload to only run on a machine that it is supposed to!

Situational Awareness

- Device a way to limit the number of implants running on the same host (Mutex, named pipe...etc)
- Read the environment variables
- Get the computer's mac address
- Get the computer’s IP address
- Get the current username and token
- Get the Computers name
- Get the Machine GUID
- List files in a directory
- Change Directory
- Bonus: Anti Sandbox Detection

Config Modification

Each implant will be compiled with configuration data including

1) The C2(s) to communicate back to
2) The amount of time to go dormant before communicating again
3) The randomness to add to that sleep time
4) The kill date of the implant
5) The server’s public key

Execution

- CreateProcess and redirect output to pipe (shell commands)
  - Feel free to use powershell.exe /c
- Shellcode Execution (Local execution, Remote process Execution)
- 1 of the following: Reflective DLL Injection, PE Injection, (bonus) COFF injection, Dark LoadLibrary based injection.
  - There are many, many, many ways to do this. The most common being Allocate memory → Copy shellcode → create (remote) thread. More points will be given to more exotic techniques
- Bonus: Integrate framework such as Donut

Loot

Programmatically loot information from the victim machine.

- Basic: Chrome passwords and cookies from default user profile
- Full Credit: All chromium based passwords, cookies, autofill, and web history from all user profiles
- Or you could implement a different type of looting functionality approved by course staff. Just ask!

Persistence

You must implement functionality for the device to persist on the target machine passed reboots. While we strive to make all of our payloads file-less, sometimes we need to touch disk in order to persist on the machine. The basic method of doing this is to copy the Executable to disk, and run it at startup. There are many ways to do this.

To get full credit, you must implement at least 1 strategy for persisting. Bonus points if you tailor your payload to the machine (i.e., the exe will only run if the machine has the same name, MAC address...etc)

File I/O

The implant should be able to

1) Read files from the victim machine and send it back to the C2.
2) Download files from the C2 and write them to disk either encrypted, or unencrypted.

Defense Evasion

You must implement at least 1 tactic for evading defenders. This can be

1) A crypter that “packs” and obfuscates your payload
2) A method for defeating AMSI
3) Anti-analysis techniques
4) Anti-Sandbox Techniques
5) An RPC to mimic a legitimate service
6) Payload tailoring

RPC and C2 Channel

Your implant must communicate to the C2 server using an RPC.

This can be JSON, base64 encoded data, whatever. So long as it is communicating over an approved C2 Channel.

- When using HTTP, your implant **must be proxy aware**
- If using HTTP, you must use TLS (HTTPs)
- Your implant shouldn’t crash if internet cuts out

Special Feature

You must implement an advanced C2 feature.

This could be an advanced

- Implant command
- RPC
- UI feature
- General

You may propose your own and I have final say over what is “advanced” :D

Special Feature implant

1) API Hashing for dynamic Resolution
2) Tailored Payloads
3) Initial Access Payload (word document, HTA, LNK...etc)
4) Defeating Impash with randomized imports
5) Token Manipulation
6) Proxy Pivots (port forwarding)
7) P2P C2 ← Automatic A
8) Custom Image Loader ← Automatic A
9) Direct System Calls ← Automatic A

Special Feature: RPC

1) Computation based sleeping
2) Advanced use of cryptography (Password Authenticated Cryptography)
3) Programmatically control your C2 via API
4) Protocol Buffers for RPC / gRPC
5) Steganography ← instant A
6) Support alternate C2 Channel: DNS, Websockets, SMTP...etc ← Instant A
7) Malleable C2: customizable RPC ← instant A

Special Feature: UI

- Use Rich and prompt toolkit to make a “pretty” User interface
- Build an HTTP panel to control the implant (with secure authentication)
- Open to suggestions!

General

- Trolling. I love a good troll. If your implant does something funny, that counts. But it can’t be as simple as playing music or moving the mouse. It needs to be funny.
- Containerize everything

Have an idea you think is cool? Let me know! And I will probably approve it as the special feature.

Remarks

- All code needs to run on the Windows VM setup for this class
- All commands needs to be documented
- Your code needs to be coherent and commented
- Your project needs to be organized.
- Rules of thumb:
  - No functions with more than 50 lines of code
  - No files with more than 500 lines of code
  - Provide examples of usage for the client, server and implant
- Your code needs to be compilable on a Linux machine. No Visual studio

Build Recommendations

- Stick with mingw g++ /clang++
- GNU Make is powerful enough for this project
- If you would like, you may also use cmake so long as it is documented
- For other languages/build systems, please ask me first
- You can also configure implants via loading a resource file. If you use this strategy and build implants from a template file on a linux machine, this is also OK

The best C2 will get a prize :-)

The team with the “best” C2, according to course staff, will receive a mystery reward.

This will be judged on

- Opsec considerations: is your C2 easy to detect? Does it do anything that is super noisey? Does Windows Defender catch it :)
- Documentation: how easy is it to follow what your C2 does?
- How cool is the special feature?
- Is the UI usable?

Projects You can, and should use for reference

https://github.com/rapid7/metasploit-payloads/tree/master/c/meterpreter (C implant)

https://github.com/bats3c/shad0w (really great C2, built by a freakin 18 year old! C + python)

https://github.com/byt3bl33d3r/SILENTTRINITY (really cool interpreter embedding, really solid python c2)

https://github.com/stephenfewer/ReflectiveDLLInjection

https://github.com/fancycode/MemoryModule (Like RDI but better)

https://github.com/TheWover/donut (turns PEs and EXEs (native and .Net) into shellcode

https://github.com/monoxgas/sRDI (Same idea(

https://docs.mythic-c2.net/ (very cool project)

https://github.com/gentilkiwi/mimikatz (Does

https://github.com/bats3c/DarkLoadLibrary (A custom implementation of LoadLibrary that works on both DLLs and reflective DLLs. Also written by the same 18 year old!)

Open Source Tools: Shad0w

- C implant, python C2
- HTTP Client
- Execution
- Defense Evasion
- Use of Cryptography
- RPC
- C2 Client