First check the group memes folder in our GitHub!
https://github.com/jynCoder/501Project/tree/main/group%20memes

## How do I install your implant?

- Our implant is implemented in scattered sections based on requirements necessary for the project.
- Different folders in our repository contain different C++ files related to implant functionality. Makefiles are provided in these folders in order to create executables that our implant can run.
- Folders:
  - "crypto", "execution", "loot", "parser", "persistence", "payloads", "implant_awareness", "c2", "http"
- Once created, please see the functionality of our "parser" in the "Other/Extra" section. This section provides more information regarding how the implant was intended to be constructed.
- Executables are not in the repository. They must be compiled locally.

## How do I install the C2?

- Pull the files off of our team GitHub and look for the C2 folder
- Inside the C2 folder there should be a makedb.py file, run this.
- Next, run the c2_server.py file and the server should start up in a local form
- Once the server is up, run the agent.py file and at the endpoint "/agent/list" grab the agent ID that is displayed
- With the agent ID copied, run the test.py file with the agent ID as the parameter. Check out the "/task/list" endpoint to see that a task was created.
- Next, try running one of the .exe files and you should see that info was relayed on the "/output/list" endpoint

## How do I modify the config to allow for connections to different C2s?

- You would need to go through each of the C++ files related to capabilities of our implant and manually change the IP.

## Requirements

- All required files are already in our repository. Any Python-related packages can be downloaded using pip. Any C++-related packages/imports are statically linked to executables (in our repository).

## Capabilities

- Our implant can:
  - Execute programs (see "Execution" and "Injection" sections)
  - Encrypt/decrypt strings with asymmetric and symmetric methods (see "Cryptography" section)
  - Loot Google Chrome passwords (see "Looting" section)
  - Persist on a victim's machine (see "Persistence" section)
  - Gather information about a victim's machine (see "Defense Evasion" section)
- Our C2 can:
  - Communicate with our implant via GET/POST requests from the implant
  - Send tasks to the implant
  - Receive a tasks output/data from the implant
  - Display such information on a dashboard UI

## Execution

- Please see the "execution" folder in our repository
  - https://github.com/jynCoder/501Project/tree/main/execution
- "createProcess.cpp" can be used to create a process using pipes
- "injection.cpp" can be used to inject a running process by PID (please see the "Injection" section)
- "local_shellcode.cpp" can be used to inject a process that is currently running using "GetCurrentProcess" with shellcode
- "http.h" and "http.cpp" are used to send POST requests to our C2. These requests contain output related to the execution of the specific programs run. Requests are sent to the endpoint "/output" on our C2 server.
- "Makefile" is used to take the C++ files in this folder and turn them into executables for use by the implant. File "http.cpp" is statically linked and is necessary for compilation of the three C++ files.

## Looting

- Please see the "loot" folder in our repository
  - https://github.com/jynCoder/501Project/tree/main/loot
- "loot.cpp" can be used to obtain Google Chrome passwords from a user that is currently logged in (using "GetUserNameA")

- “http.h” and “http.cpp” are used to send POST requests to our C2. These requests contain output related to the execution of the specific programs run. Requests are sent to the endpoint “/output” on our C2 server.
- “aes_gcm.h” and “aesgcm.cpp” are used for AES-GCM encryption and decryption. These functions are necessary for the looting process.
- “sqlite3.h” and “sqlite3.c” are used as a SQLite library in C++. Source: “https://www.sqlite.org/capi3ref.html”. They are needed in order to obtain data from the “Login Data” database.
- “json.hpp” is used as a JSON library in C++. Source: “https://github.com/nlohmann/json”. This library is used to parse JSON data in “Local State”.
- “Makefile” is used to take “loot.cpp” in this folder and turn it into an executable for use by the implant. Files “http.cpp”, “aesgcm.cpp”, and “sqlite3.c” are statically linked and are necessary for compilation of “loot.cpp”.

## File I/O

- Work in Progress. Please see our branch “implant_fileIO”.
    - https://github.com/jynCoder/501Project/tree/file_i/o

## Defense Evasion

- All tasks related to situational awareness do not use “powershell.exe” or “cmd.exe”.
- A XOR cipher has been created in C++ and Python (see “Use of Cryptography” section) for use in our implant and C2. We have partially implemented symmetric cryptography in our implant and C2.

## Situational Awareness

- Our implant has several functions that allow us to retrieve various information about the user’s machine and filesystem. Specifically, we can retrieve environment strings, adapters info, windows version, user token, username, computer name, machine guid, filenames in a particular directory (set to current drive by default), all running processes, and change the directory that the exe is currently observing.
- That said, with our given implant architecture, this aspect of our implant will simply retrieve filenames for the default directory and send it to the C2. Our future goals would include integration with the other functions so that the C2 client can thoroughly snoop through the target machine.

- https://github.com/jynCoder/501Project/tree/main/implant_awareness

## Injection

- Please see the "execution" folder in our repository
    - https://github.com/jynCoder/501Project/tree/main/execution
- "injection.cpp" and "local_shellcode.cpp" have been discussed in the "Execution" section

## Persistence

- Please see the "persistence" folder in our repository
    - https://github.com/jynCoder/501Project/tree/main/persistence
- "persistence.cpp" is used to copy an executable to disk. We use the "RunOnce" Windows registry key to start a program when a user logs in.
- Please change the placeholder "malware.exe" to the name of our compiled malware.
- "http.h" and "http.cpp" are used to send POST requests to our C2. These requests contain output related to the execution of the specific programs run. Requests are sent to the endpoint "/output" on our C2 server.
- "Makefile" is used to take "persistence.cpp" in this folder and turn it into an executable for use by the implant. File "http.cpp" is statically linked and is necessary for compilation of "persistence.cpp".

## Resilience

- Our implant should maintain persistence on a user's machine if Internet connection cuts out or if there are any other disruptions. This is assuming that the C2 channel has sent a request for persistence on the victim's host machine.

## Supported C2 Channels

Our C2 currently supports HTTPS as that is the type of calls the implant makes to it to get all the necessary tasking information.

## Use of cryptography

- Please see the "crypto" folder in our repository.
    - https://github.com/jynCoder/501Project/tree/main/crypto

- "asymmetric_crypto.cpp" can be used for RSA encryption, decryption, and key generation.
- "symmetric.cpp" is an implementation of a simple XOR cipher in C++. "symmetric_crypto.py" is an implementation of a simple XOR cipher in Python. These two programs can be used to encrypt and decrypt strings in our C2 and implant. The original idea was to use the C++ implementation for our implant and the Python implementation for our C2.
- "check_filename_hash.cpp" can be used to check that the file "C:\malware\ch0nky.txt" exists on the victim's machine. Also, that the hash of the filename "C:\malware\ch0nky.txt" matches with the hash of the filename on the victim's machine.
- "http.h" and "http.cpp" are used to send POST requests to our C2. These requests contain output related to the execution of the specific programs run. Requests are sent to the endpoint "/output" on our C2 server.
- "Makefile" is used to take the C++ files in this folder and turn them into executables for use by the implant. File "http.cpp" is statically linked and is necessary for compilation of the three C++ files.

## Other/Extra

- Contents of the "http" folder in our repository:
  - https://github.com/jynCoder/501Project/tree/main/http
  - Copies of the "http.h" and "http.cpp" files used for the other features of our implant.
  - To recap:
    - "http.h" and "http.cpp" are used to send POST requests to our C2. These requests contain output related to the execution of the specific programs run. Requests are sent to the endpoint "/output" on our C2 server.
  - Also, there is a README that can be used with the commented out "main" function in "http.cpp" to send GET/POST requests. The README explains the parameters used for functions "makeHttpRequestGET" and "makeHttpRequestPOST" in "http.cpp".
  - "Makefile" can be used to create a standalone executable of "http.cpp" by un-commenting the "main" function in "http.cpp".
- Contents of the "parser" folder in our repository:
  - https://github.com/jynCoder/501Project/tree/main/parser
  - The parser ("parser.cpp") is meant to be a continuously running program that repeatedly checks our C2 for any new tasks.

- ➢ Every five seconds, the implant will send a GET request to endpoint "/tasks/get-jobs". If any task is there and marked with status "CREATED", the proper executable is run.
- ➢ Executables can be created using makefiles in different folders related to implant functionalities. The idea is to place the executables in this folder, and then they will be called by "parser.exe".
- ➢ To run an executable, a modified version of "createProcess.cpp" is used: "parserCreateProcess.h" and "parserCreateProcess.cpp".
- ➢ After a process is run, output from the program is sent as a POST request to mark the task as "COMPLETED".
  - ■ The URL for POST requests might need to be modified.
- ➢ This file, "parser.cpp", is supposed to be the main point of operation of our implant.
- ➢ "Makefile" is used to take "parser.cpp" in this folder and turn it into an executable for use by the implant. Files "http.cpp" and "parserCreateProcess.cpp" are statically linked and are necessary for compilation of "parser.cpp".
- ➢ "http.h" and "http.cpp" are used to send POST requests to our C2. These requests contain output related to the execution of the specific programs run. Requests are sent to the endpoint "/output" on our C2 server.
- ➢ "json.hpp" is used as a JSON library in C++. Source: "https://github.com/nlohmann/json". This library is used to parse JSON data from GET requests to our C2.

❖ Our payload:
- ➢ https://github.com/jynCoder/501Project/tree/main/payloads
- ➢ "Death.exe" raises an error that causes a Blue Screen of Death on Windows. It does this without requiring administrator privileges and queries the registry for crash dump settings so that we'll have some idea what type of dump we're going to generate, and where it will be.
- ➢ "Death.ps1" is compiled into Death.exe using our ps2exe tool, in order to avoid powershell script execution policy if disabled by default.
- ➢ "Screen.exe" is a troll, which makes the screen appear to "melt" away to the new screen, The "melting" screen is subdivided into vertical slices, two pixels wide. Each slice moves down the screen at a uniform speed, but they do not all begin moving at the same time, each slice given a random yet short delay.

    - "Evil-preprocessor" was meant to be a troll-related reference to replace victim lines of code to include evil macros when we have access to their machine.
- Tor Circuit to anonymize C2:
    - [https://github.com/jynCoder/501Project/blob/main/payloads/autostart_tor_client.ps1](https://github.com/jynCoder/501Project/blob/main/payloads/autostart_tor_client.ps1)
    - Plan was to connect and access C2 via Tor circuit, which would provide anonymity both to clients as well as to servers using its hidden service. The protocol would allow a client to connect to a server knowing only an identifier and never needing to know the C2 IP.
    - To do so, we tested binding the tor.exe with other files but AV detects it as malicious file; other way around was to download tor portable exe, and auto-start it powershell script with autorun by changing the registry entry in client machine, which works on local system when tested.

Evaluate the operational security of your framework. What tradeoffs do you make?

We use HTTPS as our form of communication between C2 and implant, and from what we understand isn't the safest way of sending information, but this was the way we best understood. The C2 itself could use better login and authentication to help prevent unwanted users from accessing it, but time did not allow for this. The implant uses simple GET/POST requests for convenience-sake.

Provide a few situations in which your implant would be practical/useful:

The implant has a function to loot Google Chrome passwords and mess with files, so if it were on a high priority target it could get highly sensitive information. High priority target being a domain admin computer, person of high status, etc. Along with looting Google Chrome passwords, our implant has the functionality to create new processes, which could be used to run external (downloaded) programs.

Who contributed what?

Luke: Worked mostly on the implant side and fully implemented cryptography-related programs ("crypto" folder), execution-related programs ("execution" folder), our Google Chrome looter ("loot" folder), persistence for our implant ("persistence" folder), HTTP GET/POST capabilities to/from our C2 ("http" folder), and our parser ("parser" folder). Also worked with Dalen to connect our C2 server and our implant, debugging issues on the implant-side of the project. Worked on completing sections of our poster.

Dalen: Worked mostly on the C2 side where I set up endpoints for the implant to communicate with for tasking and sending back data. Also built out the dashboard UI that holds the tasking information as a part of the special feature. Worked on completing sections of our poster.

Shirene: Worked on the supporting functions for situational awareness, used in our implant.

Harshit: Worked on the payloads for the C2 to deploy once a victim is infected, and building Tor circuit to anonymize the server; (working with savant when enabled hidden_services to access server live with our customized .onion site). Worked on completing sections of our poster.

Annette: Worked on functionality of our implant related to file I/O. Also ordered our poster and picked it up.