Jacob Torres

CS 564

5 February 2025

Assignment 7

I've organized my files into a GitHub repository, which can be found here (https://github.com/JacobTpng/C2_base_imp). I will be double-linking everything in case the clickable link does not register in my submission. The following is my writeup for the development of this C2 implementation.

Tasking:

My communication protocol in the c2_server.py file approaches the tasking requirement using a disguised GET endpoint that appears as a harmless static resource, "/images/logo.png." The implant in *implant.py* polls this endpoint by sending its identifier, with the C2 server responding with an AES-encrypted task command. This process mimics the behavior of malware beaconing to allow the implant to receive instructions like executing shell commands or performing control functions.

Exfiltration:

Exfiltration is implemented through a disguised POST endpoint called "/updates/check" in c2_server.py. After executing a task, the implant in implant.py encrypts the output using AES and sends the encrypted data back to the server. The C2 server then decrypts and logs exfiltrated data with a timestamp. This mechanism uses techniques found in various open-source C2 projects like Baby Shark (https://github.com/UnkL4b/BabyShark), and Flask's documentation (https://flask.palletsprojects.com/en/stable/) to make sure that both data and status information are securely transmitted from the implant to the C2 server.

Control of the implant:

Implant control is done in the tasking protocol within implant.py, which supports specific control commands like "destroy" to self-delete, "status" to gather and report system details, and "contingency" to trigger fallback behavior when C2 server communication is lost. The design for these control commands is influenced by our week 7 course lectures and real-world implant strategies used in frameworks like [Cobalt Strike](https://www.sentinelone.com/cybersecurity-101/threat-intelligence/what-is-cobalt-strike/) (https://www.sentinelone.com/cybersecurity-101/threat-intelligence/what-is-cobalt-strike/). It demonstrates how an implant can be directed to perform administrative and self-management tasks in response to remote commands.

Obfuscation Techniques:

I used two layers of obfuscation in the implementation. The first is AES-CBC encryption used to secure communications between the implant and the C2 server, as defined in crypto_utils.py. It is based on patterns from the PyCA Cryptography documentation and its GitHub repository (https://github.com/pyca/cryptography), which provides clear examples of using PKCS7 padding and random IV generation. The second technique is endpoint disguise, where the endpoints on the C2 server in c2_server.py are renamed to mimic benign resources. I specifically used "/images/logo.png" for tasking, "/updates/check" for exfiltration, and HTTP headers like "User-Agent: Mozilla/5.0" to further blend traffic with ordinary web requests.

Testing:

I tested the implementation on a Linux virtual machine. I installed all dependencies with “pip install flask cryptography requests.” The C2 server was launched with “python c2_server.py” in one terminal, which started listening on port 5000. The implant was started in a second terminal with “python implant.py.” On the C2 server terminal, each poll by the implant generated log entries like:

```
127.0.0.1 - - [Date Time] "GET /images/logo.png?id=implant_001
HTTP/1.1" 200 -
[Date Time] Exfil from implant_001:
*my personal device’s directory*
```

from the whoami task, followed by:

```
... GET /images/logo.png?id=implant_001 HTTP/1.1" 200 -
[Date Time] Exfil from implant_001:
[STATUS]
Time: ...
User: ...
Uptime: ...
Disk: ...
```

for the status task. The destroy command output “Implant destroyed” from the implant console, followed by an emptied task queue. All following GET /images/logo.png?id=implant_001 requests returned HTTP 204, and all exfiltrated data appeared properly decrypted by the server, as they are all human readable, confirming the AES-CBC layer in crypto_utils.py worked as intended.

On the implant terminal, the sequence of tasks was printed:

```
[implant_001] Received task: whoami
[implant_001] Received task: status
[implant_001] Received task: destroy
Destroying implant...
[implant_001] Received task: contingency
Entering contingency mode.
```

And the file is no longer present after the stated deletion, proving its effectiveness.

Documentation:

In addition to this document, all code files are commented on to explain their functionality and the rationale behind design decisions.