Jake McLellan

Kayla Hodgson

12/07/2021

CSEC 467 Final Paper

Table of Contents

1. Introduction

Adversary emulation and Red Team tools exist as a means to test the preparedness and reliability of security teams. Typically, these tools exist in two primary components - a Command and Control (C2) server and the agent. The agent repeatedly “phones home” to the C2, indicating to the adversary that the connection is still alive. Additionally, the agent may query the C2 for additional actions to execute. This may include shell commands, files to upload or download and more. Thus, the C2 and agent are important tools for adversaries looking to maintain persistent access within a compromised system (Liu, Jing, et al., 2009, p. 6).

Android applications create ease of use with a C2 as one is able to develop it to meet specific needs. This app seeks to utilize WiFi to receive callbacks from devices within a LAN, and cellular data on the device running the C2 to exfiltrate data to an external, attacker-controlled device, while bypassing egress firewall rules in the process. This is ideal for attacks targeting corporate networks with significant data egress restrictions. Additionally, a C2 within a mobile application allows the adversary to continuously have control of the infected device without the need to be in front of a computer for any portion of the day. This is especially useful in conducting a physical penetration test, as it gives the attacker instant and inconspicuous access to the C2 for time-sensitive operations.

This paper outlines the reasoning, methodology and future work in developing the mobile C2. Additionally, results and drawbacks of this research will be thoroughly examined to improve further development of the project. It is hopeful that this research leads to a new branch of C2s which increases portability, usability and covertness within adversary emulation tooling.

2. Background

Command and Control (C2) servers also referred to as a C&C or malware server are adversary-controlled servers in which infected devices call back to in order to receive additional instructions (Gardiner, Cova, Nagaraja 2014, p. 3). Most C2 servers are platform-agnostic as they only provide a means for an adversary to communicate with the agents. The agents, however, may differ depending on the target platform and design choices. Additionally, this tool allows the adversary to carry out a post-exploitation attack against their target.

Network communication is a key highlight of C2 development. There are numerous questions that a C2 developer should ask themself while planning. First, priorities should be established which may include the need for stealth, reliability, and simplicity to balance the ability to avoid detection from network security teams while maintaining a connection, and to implement communication easily (Pieterse, Olivier, 2013, p. 19). Additionally, it is important to consider whether the traffic must be bidirectional, or if unidirectional communication will suffice. Scalability should also be a consideration in overall C2 network architecture, allowing for a large quantity of bots at any given moment. These considerations help guide C2 developers in the right direction when planning their protocol of choice for network communication.

Red Team exercises are defined by CrowdStrike as "a cybersecurity assessment technique that uses simulated attacks to gauge the strength of the organization's existing security capabilities and identify areas of improvement in a low-risk environment" (CrowdStrike, 2021). They date back to military training exercises and are an active way to test the reliability and responsiveness of a security team. There are a number of existing tools that can be used for conducting Red Team exercises such as Cobalt Strike, Empire and other proprietary solutions.

The Mobile C2 is intended to be used in Red Team exercises by teams looking to emulate the dangers of insider threats and physical security mishaps.

Recently, it has become more common to see attackers use IoT devices as a means to stage and exfiltrate stolen data from targeted networks, as they are easily susceptible to exploitation and may be overlooked by network administrators (D'Orazio, Choo, Yang, 2016, p. 525). Currently, properly configured corporate networks have numerous restrictions on what data can leave the network. They have ways to detect when an attacker is trying to steal large quantities of data at a given time, through egress firewall rules, but the defense against data exfiltration can pose an issue for Red Teams. However, if the data is instead exfiltrated over cellular data and not over the corporate network, it is possible to circumvent these restrictions altogether allowing mobile devices to have the potential to serve as an optimal device for data exfiltration.

3. Methodology

a. Mobile C2 Architecture

Due to portability of mobile devices, it gives the adversary the ability to use the C2 in their everyday life inconspicuously, as it appears that they are just using their phone like any normal user and data is not tracked from WiFi because they are using cellular data for exfiltration. Today, mobile phones are reasonably common to see within a corporate network, and even more so in residential. Additionally, it offers the adversary flexibility and speed to interact with infected devices instantly via one's device. As previously mentioned, this is an advantage that helps make physical penetration tests more inconspicuous. It is typical to see people around a given corporate environment spending time on a mobile device for either business or personal reasons. With the proposed Mobile C2, the adversary would be able to blend in with people whilst interacting with any infected devices as soon as they begin calling back.

Android applications were chosen for development since it is much simpler compared to iOS, as there is a much lower barrier to entry in terms of development. Using Android allows one access to the device with a high level of privileges, whereas iOS would require exploitation for the same result. Additionally, in terms of the client, an Android device would be easier to find vulnerabilities that could be exploited for privilege escalation, code execution (perhaps the addition of a worm component) and beyond.

For the Mobile C2, TCP was chosen as the primary method of network communication for various reasons. First, it’s easier to prototype with as there is less overhead while developing. Secondly, a functioning client and server are not required for basic testing - just a simple netcat listener is all that is needed. Finally, unlike UDP it provides reliable message delivery as TCP will attempt to retransmit data that does not successfully reach the intended destination.

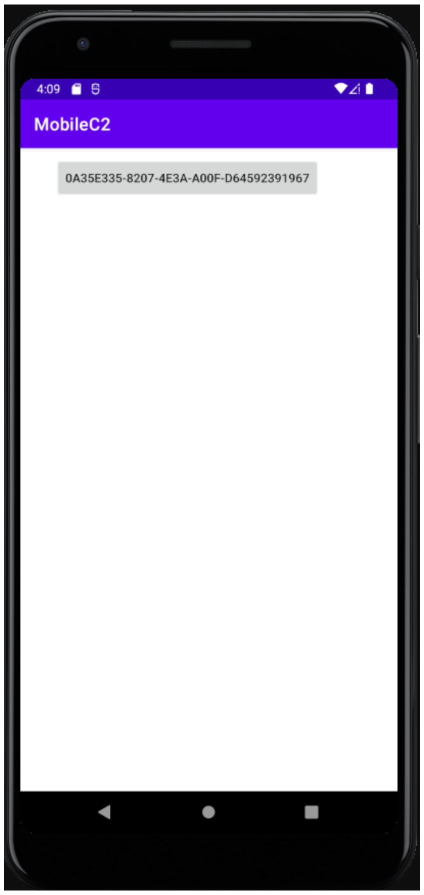
b. The Development Process

The overall development process took place across three distinct phases: planning, implementation and testing. Throughout the planning phase, proof of concept code was developed to test functionality of multiple design choices such as multithreading capabilities, database management, data exfiltration, Graphic User Interface (GUI) options and socket communication. The SQLite database schema includes a universally unique identifier (UUID) as the primary key, as well as entries for a command and output. Additionally, a client protocol was developed to include functionality for fetching a new UUID from the server, setting the UUID upon reconnection on the server-side and polling the server to receive commands to be executed known as a heartbeat.

i. C2 Design and Implementation

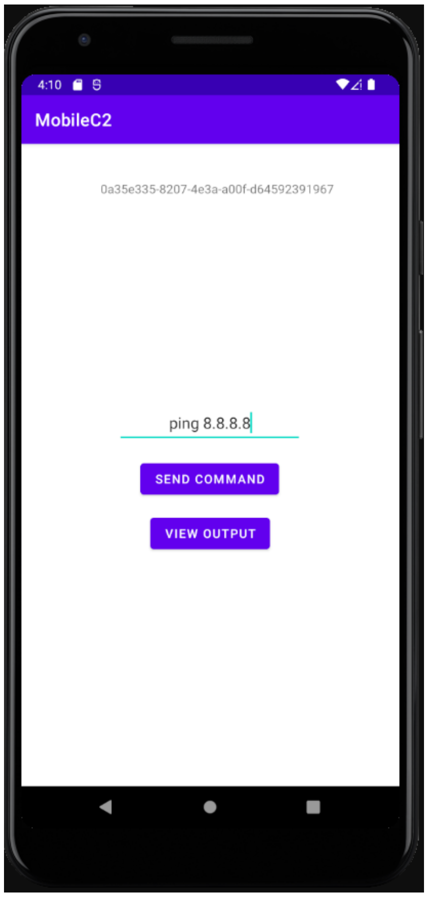
After finalizing the planning phase, implementation of the C2 development started, beginning with handling basic network communications with a ServerSocket bound to TCP port 25565. Once successful, multithreading was implemented in the C2 ClientThread class to handle multiple incoming connections at any given moment. Then, the SQLite database was implemented to handle persistence between sessions and store bot information. When an entry is added to the database, the bot's UUID and command are added, but the output remains empty until information is received back from the client in which the server will then update the database to later be displayed. Once this step was achieved, the core C2 functionality was complete.

To tie everything together, three GUIs were developed passing the bot UUID via Intent to each activity to display a list of available bots by UUID, input commands, view output from the bots and exfiltrate the UUID and output to an external Flask server.



0A35E335-8207-4E3A-A00F-D64592391967

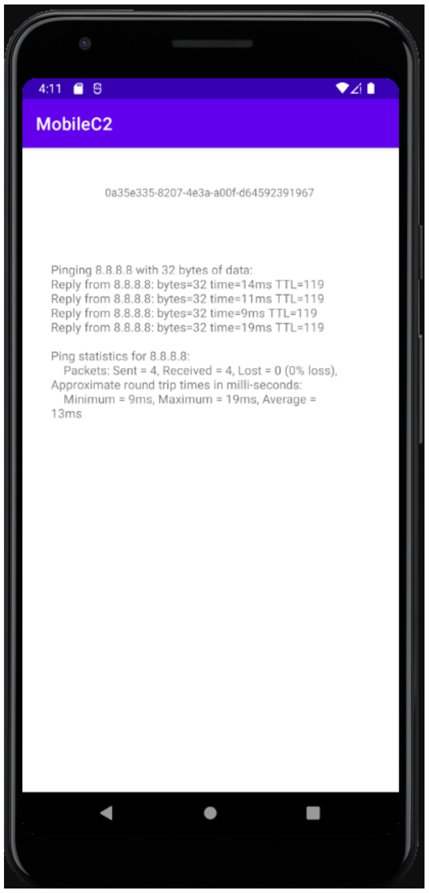
Figure 1. GUI 1



0a35e335-8207-4e3a-a00f-d64592391967

```
ping 8.8.8.8
```

Figure 2. GUI 2



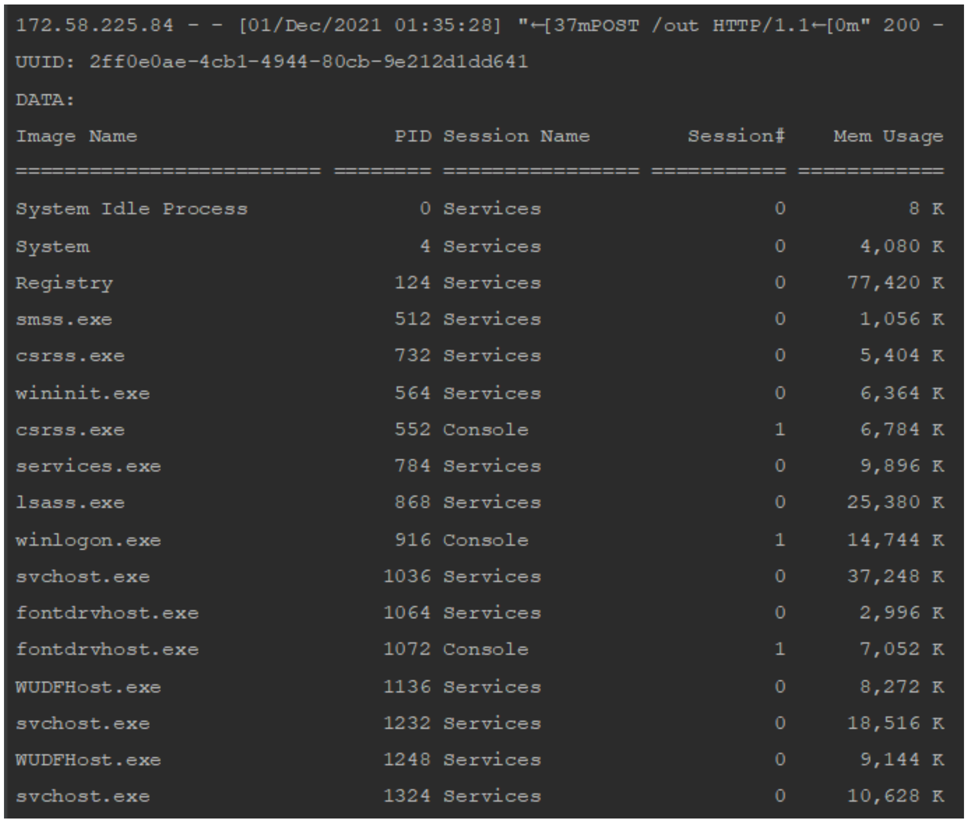
0a35e335-8207-4e3a-a00f-d64592391967

```
Pinging 8.8.8.8 with 32 bytes of data:
Reply from 8.8.8.8: bytes=32 time=14ms TTL=119
Reply from 8.8.8.8: bytes=32 time=11ms TTL=119
Reply from 8.8.8.8: bytes=32 time=9ms TTL=119
Reply from 8.8.8.8: bytes=32 time=19ms TTL=119

Ping statistics for 8.8.8.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 9ms, Maximum = 19ms, Average =
13ms
```

Figure 3. GUI 3

The final function implemented is data exfiltration. When the "Exfiltrate" button is pressed, the UUID and output fields are gathered from the GUI shown in Figure 3 and the output is Base64 encoded to avoid formatting errors. Then, the Android Connectivity Manager is used to find an available Network with the NetworkCapabilities.TRANSPORT_CELLULAR transport type enabled. If one is found, the cellular network is used to make an HTTP POST request to the specified URL with the UUID and encoded output as parameters to exfiltrate the gathered data to an external Python Flask server. This server parses POST requests to the "/out" endpoint, Base64 decoding the output and logging the UUID and decoded content to the console as shown in Figure 4 below.



```
172.58.225.84 - - [01/Dec/2021 01:35:28] "←[37mPOST /out HTTP/1.1←[0m" 200 -
UUID: 2ff0e0ae-4cb1-4944-80cb-9e212d1dd641
DATA:
Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
System Idle Process              0 Services                   0          8 K
System                           4 Services                   0      4,080 K
Registry                       124 Services                   0     77,420 K
smss.exe                       512 Services                   0      1,056 K
csrss.exe                      732 Services                   0      5,404 K
wininit.exe                    564 Services                   0      6,364 K
csrss.exe                      552 Console                    1      6,784 K
services.exe                   784 Services                   0      9,896 K
lsass.exe                      868 Services                   0     25,380 K
winlogon.exe                   916 Console                    1     14,744 K
svchost.exe                   1036 Services                   0     37,248 K
fontdrvhost.exe               1064 Services                   0      2,996 K
fontdrvhost.exe               1072 Console                    1      7,052 K
WUDFHost.exe                  1136 Services                   0      8,272 K
svchost.exe                   1232 Services                   0     18,516 K
WUDFHost.exe                  1248 Services                   0      9,144 K
svchost.exe                   1324 Services                   0     10,628 K
```

Figure 4. Data Exfiltration Server Sample Output

ii. Client Design and Implementation

With a successfully implemented C2 server, a client was needed to complete the toolset. Python was chosen as the language for client implementation due to its simplicity for prototyping and multi-platform properties. The client attempts to connect to the C2 via the given IP address on TCP port 25565. Upon successful connection, it checks to see if the "my_uuid.txt" file exists within the current directory, signifying that the client has previously connected to the C2 and was assigned a UUID. If the UUID file exists, its contents are read and the "set-uuid" message with the associated UUID is sent to the server. Otherwise, the "new-uuid" message is sent to the server and the response (a newly generated UUID) is written to the UUID text file.

Once a UUID has been acquired, the client enters its main infinite heartbeat loop, where the client sends the “heartbeat” message to the server every arbitrary amount of seconds. The server queries its database to see if a new command is available, responding to the client with “None” if no commands are present or sending the command to the client if it is present. The client then splits the string on the single space character and attempts to run the given command as a subprocess. If the command executes successfully, the output is Base64-encoded and returned to the server, although if there is no output for a command such as “calc.exe”, the string “Success” is instead encoded and sent to the server to ensure the server protocol is not stuck infinitely waiting and can perform the next query when the client sends its next heartbeat request. The client then sleeps for an arbitrary number of seconds before repeating the process.

c. Testing Mobile C2 Application and Client

During development of the C2, netcat was used to simulate a working TCP client and ensure that the protocol works successfully and reliably. Once the C2 and client were both in a functioning state, they were tested on an Android Virtual Device (AVD) emulating a Google Pixel 3a and Pixel 5 with TCP port 25565 forwarded to the host machine using Android Debug Bridge (ADB). Incremental changes were made to fix minor errors discovered during this phase of testing. Once fixed, the Mobile C2 application was installed on a physical Android device to test real-world usability of the app. On the physical device, it was possible to test the exfiltration capabilities against egress filtering in a simulated corporate environment using a router-based filter, bypassing the filter altogether by using a cellular network.

4. Conclusion

Throughout the planning, implementation and testing phase the initial possibility of bypassing egress filtering of a corporation to exfiltrate data by utilizing the cellular network on an Android device was found to be feasible. The biggest test was putting the C2 server on a physical Android Mobile device to also ensure that it can be used discretely and reliably as proposed for its usage in corporate environments for physical pentesting and attack simulation. In its current state, the application provides functionality to send, receive and exfiltrate information to and from the client. Future work for the application includes updating documentation and adding file upload and download capabilities to further data exfiltration.

Bibliography

D'Orazio, C. J., Choo, K. K. R., & Yang, L. T. (2016). Data exfiltration from Internet of Things devices: iOS devices as case studies. *IEEE Internet of Things Journal*, 4(2), 524-535.

Gardiner, Joseph, Marco Cova, and Shishir Nagaraja. "Command & Control: Understanding, Denying and Detecting-A review of malware C2 techniques, detection and defences." arXiv preprint arXiv:1408.1136 (2014).

Liu, Jing, et al. "Botnet: Classification, Attacks, Detection, Tracing, and Preventive Measures." EURASIP Journal on Wireless Communications and Networking, vol. 2009, no. 1, 13 Sept. 2009, jwcn-eurasipjournals.springeropen.com/articles/10.1155/2009/692654, 10.1155/2009/692654. Accessed 18 Sept. 2021.

Pieterse, H., & Olivier, M. (2013). Design of a hybrid command and control mobile botnet. *Journal of Information Warfare*, *12*(1), 70–82. https://www.jstor.org/stable/26487000

Red Team VS Blue Team in Cybersecurity | CrowdStrike. (2021, April 28). Crowdstrike.com. https://www.crowdstrike.com/cybersecurity-101/red-team-vs-blue-team/