Capture of Beacon Frame and Probing Requests for Identification of Frequently Requested Access Points

Authored by: Daniel Cervantes, University of Texas at El Paso

Pedro Estrada Jr, University of Texas at El Paso

# Abstract:

Network security analysts are tasked with gathering and compiling information pertaining to unknown wireless networks. Many different tools can be utilized to provide this information, which may require manual intervention by the network security analyst to conduct a successful assessment. This manual interaction with the software tools may require prior working knowledge of not only the tools, but the protocols encountered during the network security evaluation. Our research is intended to allow an analyst to conduct their assessment independent of the previous working knowledge of the network under analysis, but of the wireless protocols encountered within the test environment. We intend to utilize existing open source software and build upon this foundation. Our objective is to construct a file representation of the network under test which will includes access point information, clients, and a collection of beacon probe requests broadcast by the clients. This file will be fabricated through the dissection of IEEE 802.1x frames encountered during the examination. This information can then be utilized to create rouge access points for further security assessments.

# The following tools were used:

* TP-Link TL-WN821N wireless dongle
* RALINK USB Wifi RT5370 wireless dongle
* Github iSniff source code [1]
* Kali Linux
* VirtualBox VM software
* Python
* Scapy
* Git
* Wireshark
* PDFDump
* iPhone/iOS

# Methodology:

1. Start the Kali Linux virtual machine using VirtualBox
2. Open a new terminal and run the following commands, figure 1:
   * 1. **ifconfig**

Used to determine if the USB dongle has been detected and mounted. The USB dongle was reported as wlan0 and will be referred to as such for the duration of the document.

* + 1. **ifconfig wlan0 down**

This will bring the wlan0 interface down so that we may set it into monitor mode.

* + 1. **macchanger –r wlan0**

Changes the factory interface MAC address to a random value.

* + 1. **iwconfig wlan0 mode monitor**

Sets the USB dongle into monitor mode.

* + 1. **ifconfig wlan0 up**

This will restore the wlan0 interface, now in monitor mode.

* + 1. **wireshark &**

Wireshark is then used to verify the interface is in monitor mode and capturing non-IP 802.1x traffic

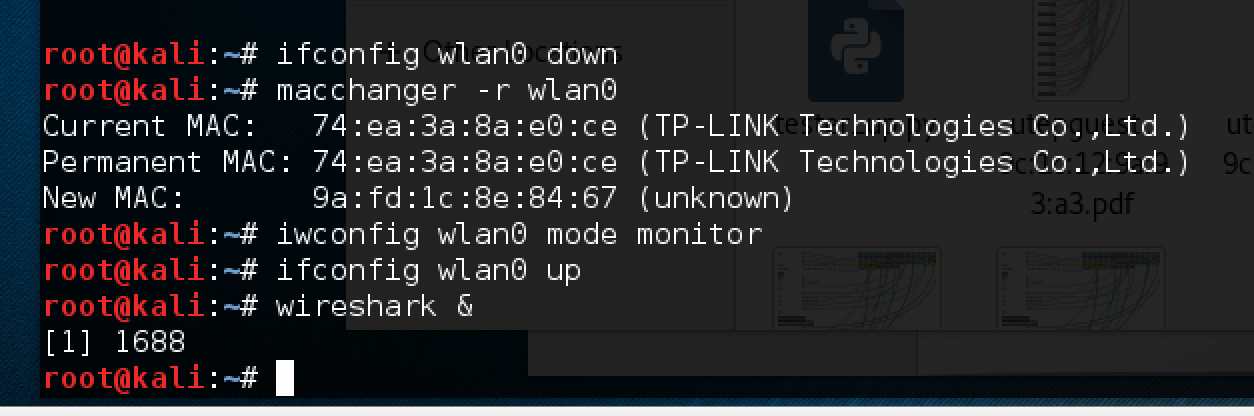


Figure Terminal commands

1. Find script online and load script (saved as Tester.py)
   1. **python Tester.py**
      1. verify script is working and capturing traffic.
   2. Analyze script and determine points for code injection
   3. Debug code
   4. Run analysis
   5. During analysis enable wireless devices with known MAC addresses, AP connections
   6. Check file results for accuracy
   7. Repeat insertion and testing in steps 3.2->3.7 until injection complete

# Frame Types:

There are three frame types that this research examines. These frames are briefly described below.:

1. Beacon Frame: Frame sent by the access point (AP) to announce its presence on the network.
2. Probe Request Frame: Client utilizes these frames to obtain information from access points.
3. Probe Response Frame: Station uses these frames to respond to probe request frames from a client.

# Scripts:

The first script called “Tester.py” was modified through code injection at points determined to be critical for our research. Figure X demonstrates how the default interface for packet sniffing was configured for wlan0 from the original value.

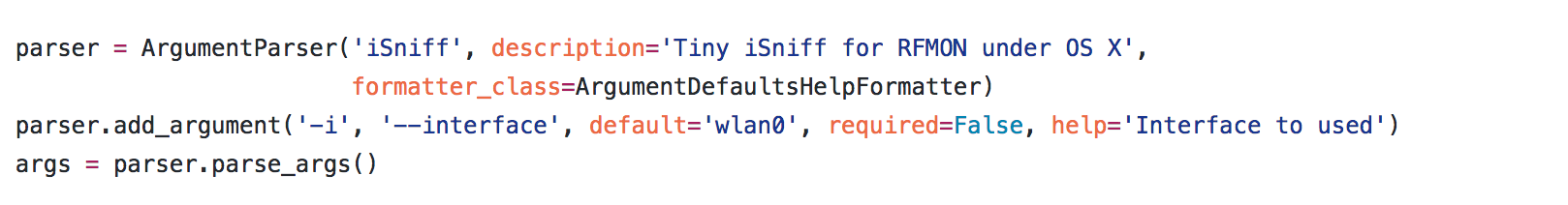


Figure default interface configuration

The main function of the script was modified to define several files names and add the names to a list for an automated log file generation, figure x.

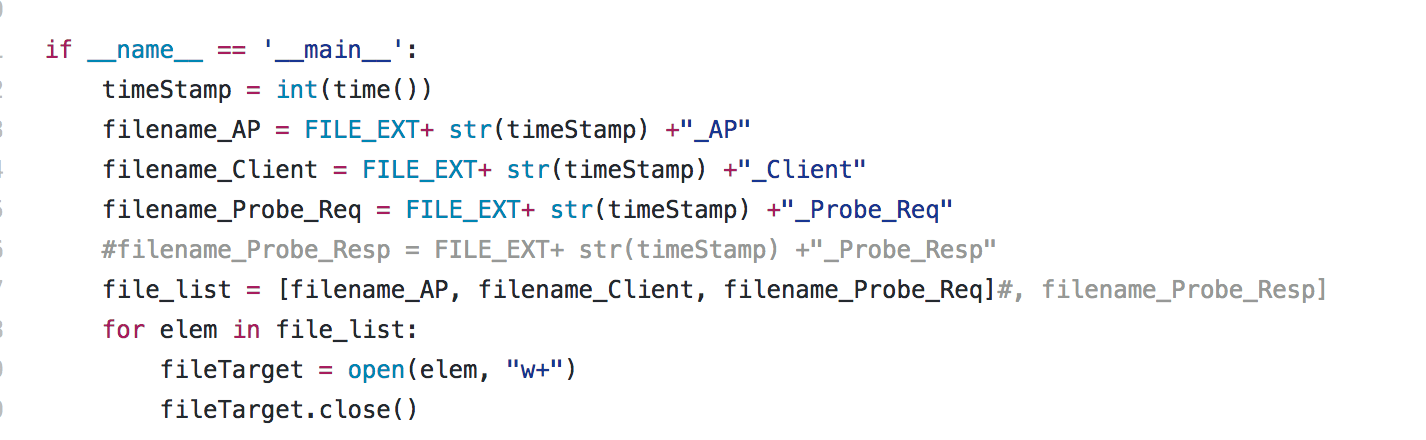


Figure Log file creation script

A new function was defined for identifying wireless network card manufacturers, figure x [3].

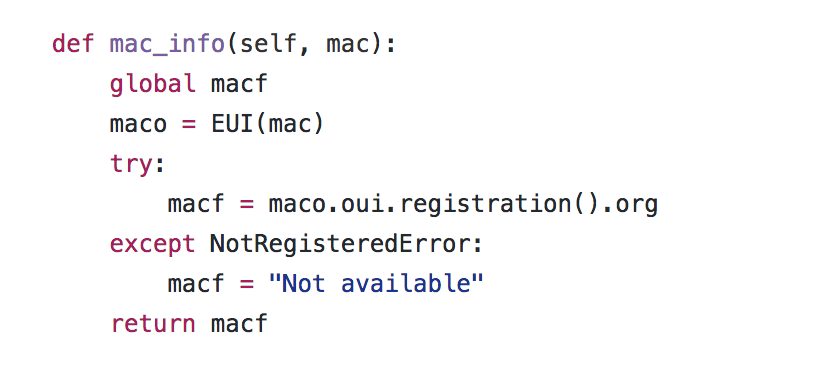


Figure Wireless network card identification

The log file updating procedures were defined in the following sections of the code, figure x and figure x.

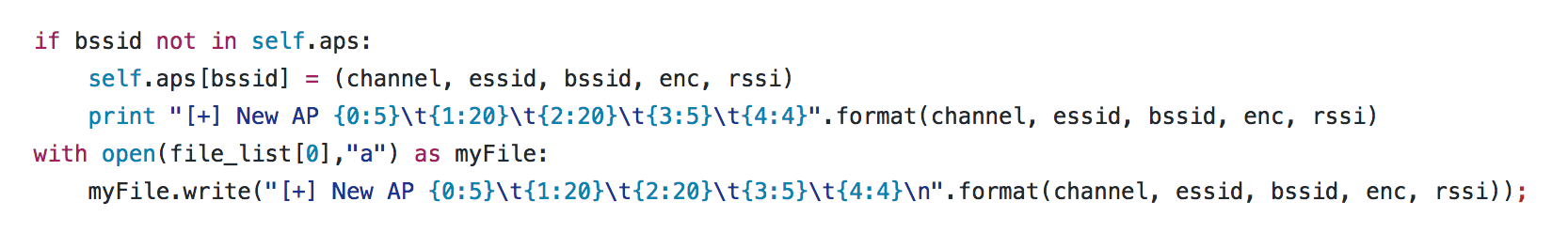


Figure Capturing AP information in a log file

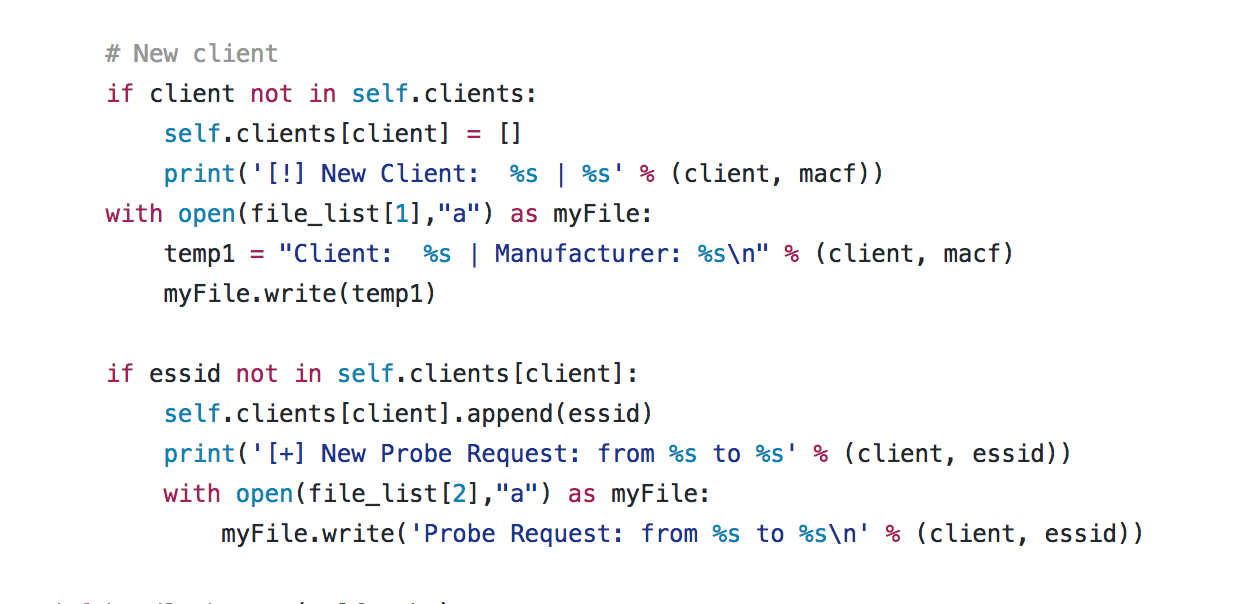


Figure Capturing client information and Probe Request in a log file

The second script called “Rogue.py” which was harnessed to create a rogue AP[4]. The code was refactored into a function and called through a main function. The main function also utilizes a parser to accept arguments, figure x.



Figure Rogue AP generator main function

# Experiment:

We walked around the first floor of the CS building for a few minutes to accumulate some log files. The four log files were created as follows, figure x.

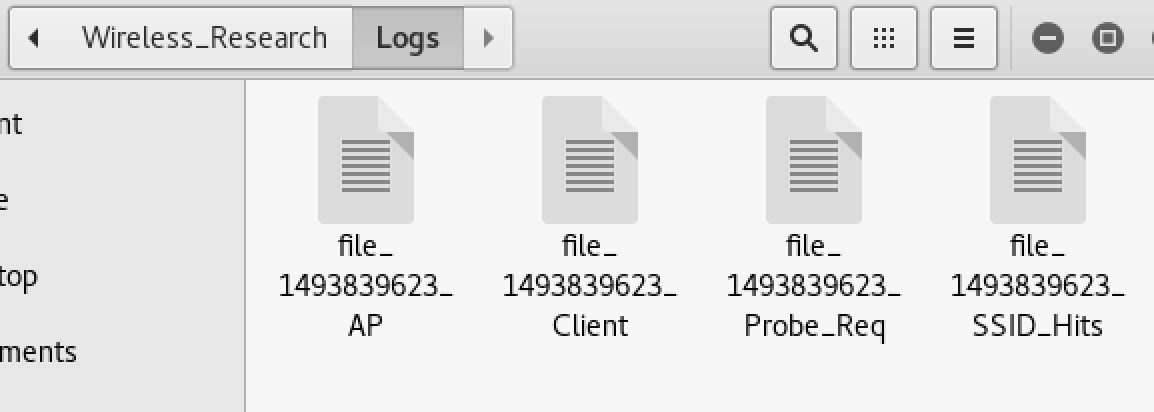


Figure Log files

The first file accumulates AP information, figure x.

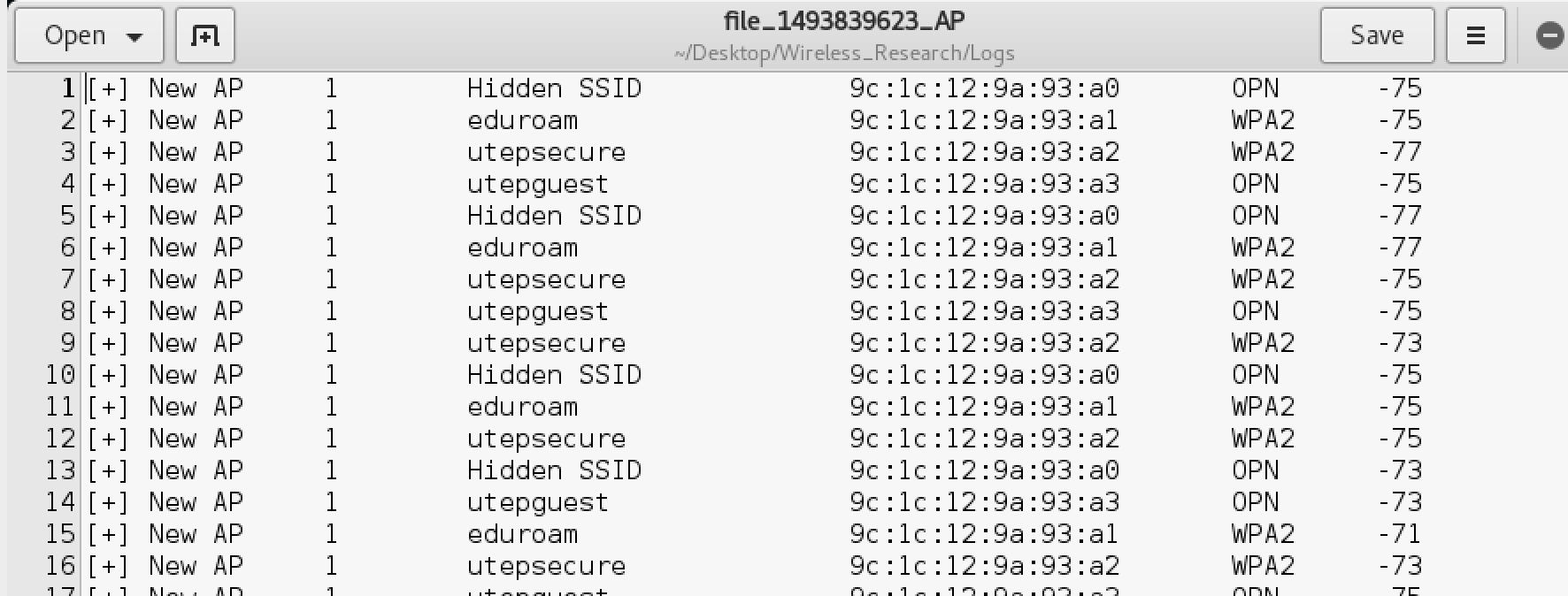


Figure AP information

The next file logs client information, MAC address and manufacturer, figure x.

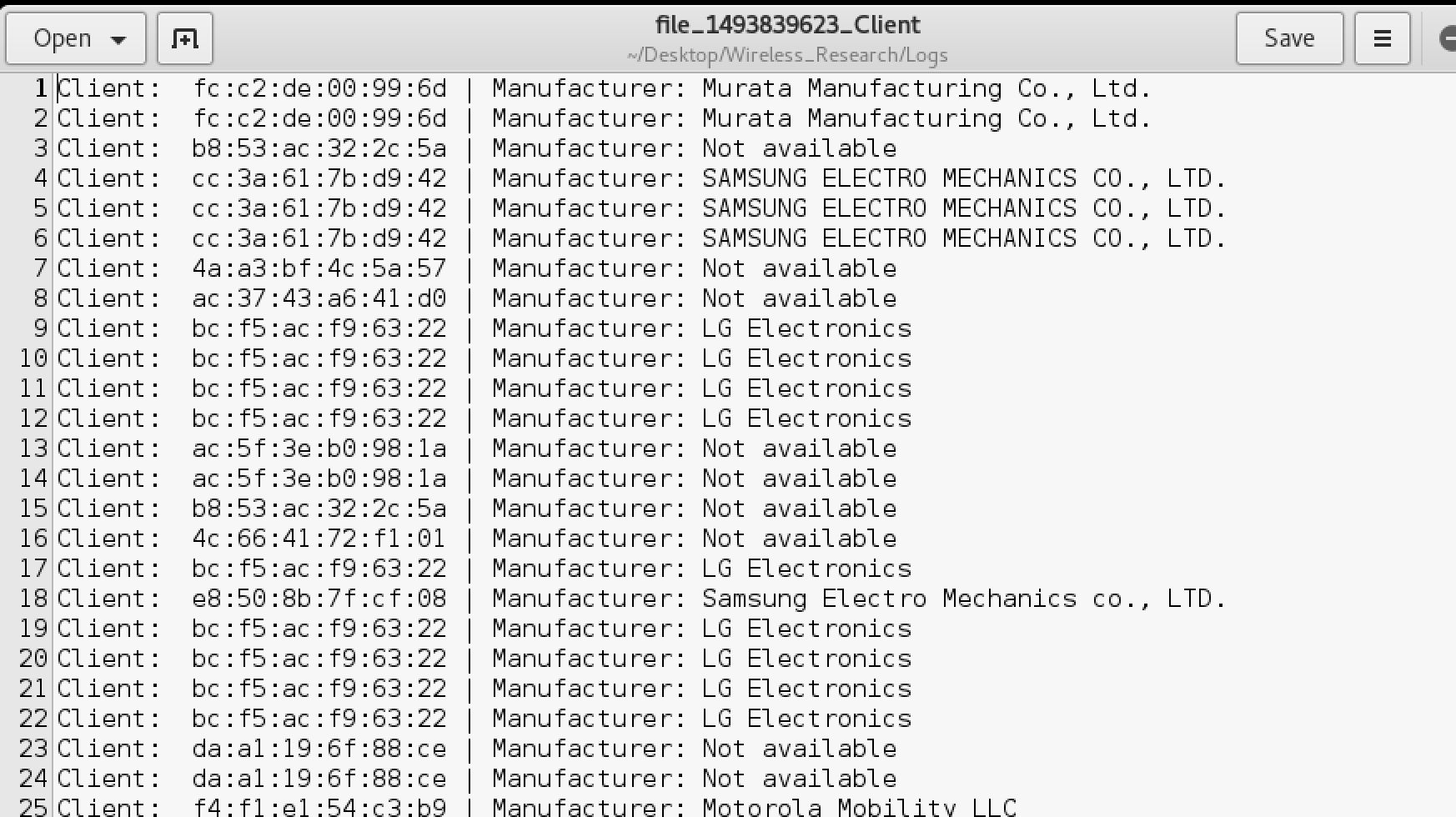


Figure Client information and wireless network card identification

The next log file contains the client probe requests, figure x.

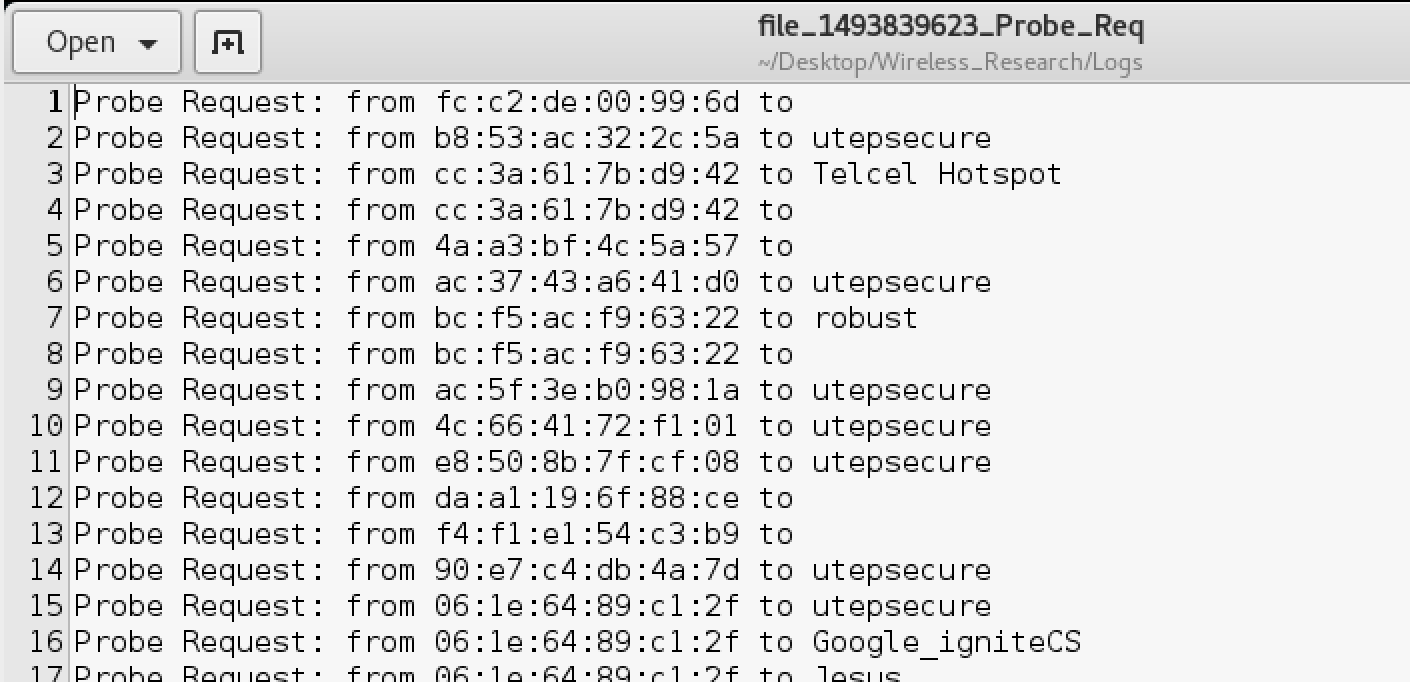


Figure Client Probe Request

Finally the last file contains the SSID probe requests, and a number of times each SSID was probed, figure x.

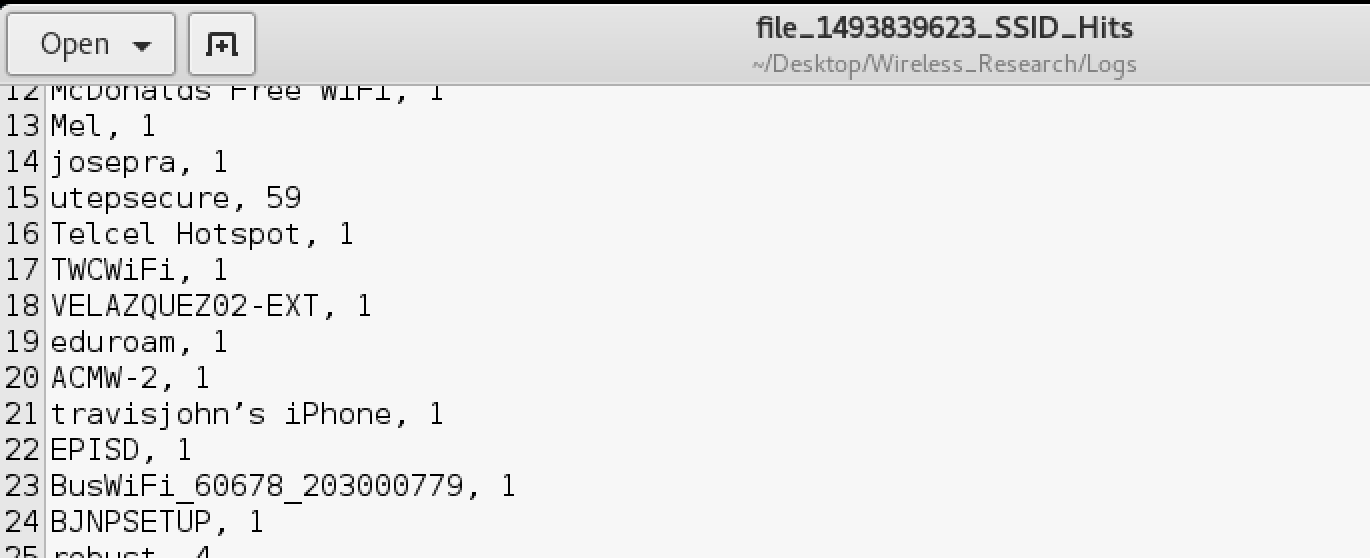


Figure SSID Hits

Running the Rogue.py script to generate a rogue point was successful. Figure x contains a screenshot of the rogue AP named, “utepSecureFakeAP.”

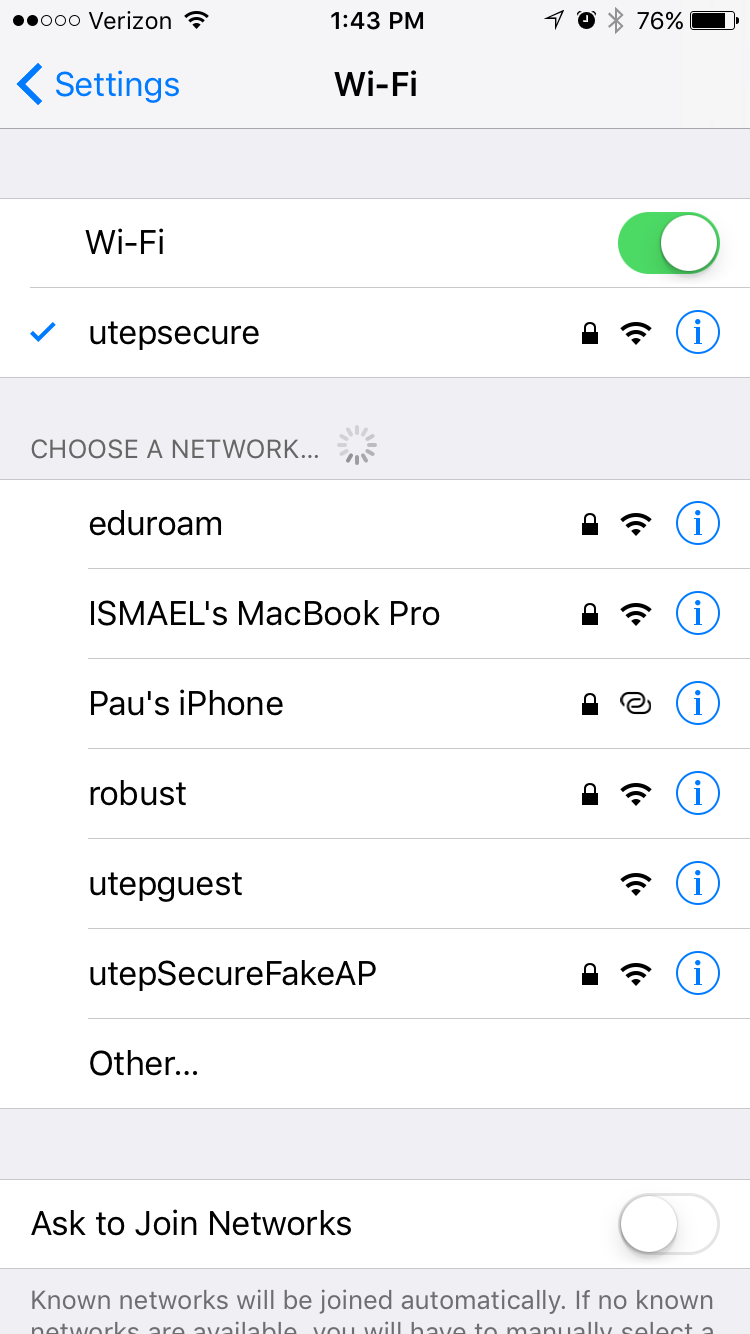


Figure Rogue AP screenshot

# Conclusion:

The experiment was successful in terms of meeting the objectives, through log file creation, and script modification. These scripts could be exported into a smaller form factor and could be carried in a pocket for further information gathering. Future work for this research can include creating shell scripts to tie the two scripts together utilizing the \*\_SSID\_HITS log file. Intel could be gathered, then in the script the metrics for the number of hits can be evaluated to determine the most suitable SSID for a rogue AP. Our research can also be extended to capture the handshake between the AP and client for evil twin generation.

Works Cited:

[1] <https://github.com/0x90/iSniff/blob/master/iSniff.py>

[2] <http://www.wi-fiplanet.com/tutorials/article.php/1447501/Understanding-80211-Frame-Types.htm>

[3] <https://github.com/drkjam/netaddr/issues/91>

[4] <https://www.4armed.com/blog/forging-wifi-beacon-frames-using-scapy/>