Stealing Credentials with Evil Twin Access Point

CS378 Final Project Paper

CS378: Ethical Hacking
Professor Chris Prosise

Group 10

Wesley Chung
Wenjie Yang
Viet Tran
Jesse Thaden

Spring 2018

Table of Contents

I. Introduction

The objective of this tool is to automate a way to steal Wifi passwords from unsuspecting victims. Instead of using a brute force dictionary attack to crack Wifi passwords, the tool provides an alternative method known as an “evil twin” attack. An “evil twin” is a fake access point that mimics a real access point. The tool sets up an “evil twin” such that when a victim uses the fake access point, their wireless traffic will be monitored and any time they try to access some HTTP website, their traffic will be redirected to a malicious website. The malicious website is used to steal credentials from the victim, such that the inputted credentials is written to a log file. This automation process is dependent on many tools, which are highlighted in the “Tools Used” section.

II. Steps of the Automation Process

1. Obtain a list of access points

Our tool first uses airmon-ng to start the interface in monitor mode, and then runs airodump-ng to get a list of access points. Our tool writes the output from the airodump-ng to a file. We then ask the user to type in the access point they wish to attack.

2. Impersonate the target access point

Our tool uses the access point name the user specifies to parse the airodump-ng output file to get the information that we want. We use the information about the real access point to create some configuration files to create the “evil twin”. We have our script automatically create the hostapd configuration file and the dnsmasq configuration file in order to set up our fake access point, as well as the DHCP server.

```
subprocess.call(['rm', 'hostapd.conf'])

with open('hostapd.conf', 'w') as f:
    f.write('interface=wlan0mon\n')
    f.write('driver=nl80211\n')
    f.write('ssid=' + name[1:] + '\n')
    f.write('hw_mode=g\n')
    f.write('channel=' + channel + '\n')
```

```
with open('dnsmasq.conf', 'w') as f:
    f.write('interface=wlan0mon\n')
    f.write('dhcp-range=192.168.1.2,192.168.1.30,255.255.255.0,12h\n')
    f.write('dhcp-option=3,192.168.1.1\n')
    f.write('dhcp-option=6,192.168.1.1\n')
    f.write('server=8.8.8.8\n')
    f.write('log-queries\n')
    f.write('log-dhcp\n')
    f.write('listen-address=127.0.0.1\n')

proc2 = subprocess.Popen(['dnsmasq', '-C', 'dnsmasq.conf', '-d'])
```

We then configure our IP tables via the iptables command line call in order to redirect traffic properly. Afterwards, victims are deauthed from the real access point in order to increase the likelihood they will connect to our fake access point.

```
# Configure iptables to reroute traffic
subprocess.call(['ifconfig', 'wlan0mon', 'up', '192.168.1.1', 'netmask', '255.255.2
55.0'])
subprocess.call(['route', 'add', '-net', '192.168.1.0', 'netmask', '255.255.255.0',
 'gw', '192.168.1.1'])
subprocess.call(['iptables', '--table', 'nat', '--append', 'POSTROUTING', '--out-in
terface', 'eth0', '-j', 'MASQUERADE'])
subprocess.call(['iptables', '--append', 'FORWARD', '--in-interface', 'wlan0mon', '
-j', 'ACCEPT'])
subprocess.call(['echo', '1', '>', '/proc/sys/net/ipv4/ip_forward'])
subprocess.call(['iptables', '-t', 'nat', '-A', 'PREROUTING', '-p', 'tcp', '--dport
', '80', '-j', 'DNAT', '--to-destination', ipAddr + ':80'])
subprocess.call(['iptables', '-t', 'nat', '-A', 'POSTROUTING', '-j', 'MASQUERADE'])
#subprocess.call(['iptables', '-t', 'nat', '-A', 'POSTROUTING', '-p', 'tcp', '-d',
'192.168.1.125', '--dport', '80', '-j', 'SNAT', '--to-source', '192.168.1.1'])
```

3. Steal credentials from the victim

Our tool steals credentials from the victim by hosting a malicious website on Apache. Our script moves all the files required to run our malicious website to the proper directory that Apache references to host a server locally (/var/www/html directory). Now the script keeps waiting until the victim submits the form on our malicious website, which writes the credentials to a log.txt file.

4. Log into the actual network and scan it

After the credentials are written to the log.txt file, our tool reads the log.txt file for the credentials. The credentials are verified by connecting to the real access point via nmcli, which is a command line tool for NetworkManager. After we are connected to the real access point, we perform an nmap scan internally.

III. Tools Used

1. Airmon-ng

Airmon-ng is a script that can be used to enable monitor mode on wireless interfaces. Our tool uses airmon-ng to enable monitor mode on our wireless interfaces, which allows us to scan the air for access points to attack.

2. Airodump-ng

Airodump-ng is used for packet capturing and displaying a list of detected access points. Our tool uses airodump-ng to write a list of detected access points to a file, which is later used to get important information from the target access point, such as BSSID, ESSID, and Channel Number.

3. Aireplay-ng

Aireplay-ng is used mainly for deauthentication. Our tool uses aireplay-ng to deauth victims from the real access point, which increases the likelihood of connecting victims to our “evil twin”.

4. Hostapd

Hostapd is a user space daemon for access points and authentication servers. Our tool uses Hostapd by automatically creating a configuration file that specifies which access point Hostapd should create.

5. Dnsmasq

Dnsmasq is a lightweight DNS/DHCP server. It is used to resolve DNS requests from and to a machine. It also acts as a DHCP server that allocates IP addresses to the victim. Our tool automatically creates a Dnsmasq configuration file that specifies configurations such as which interface to use, the dhcp-range, listen-address, etc. After some IP table configurations, Dnsmasq helps our tool monitor and redirect traffic.

6. Apache

Apache is a web server software that our tool uses to host a malicious website to steal the victim’s credentials.

7. Nmcli

Nmcli is a way to control the NetworkManager via the command line. Our tool uses nmcli to connect to the real access point after we have the stolen credentials.

8. Nmap

Nmap is a tool used for network exploration and security auditing. After our tool uses the stolen credentials to log on to the actual network, it performs an internal nmap scan of the network.

IV. How to use

a. Prerequisites

- *Kali* - Kali Linux is needed because it already has many tools installed that is used by the automation tool. For example, one of the scripts writes to the /var/www/html directory which Apache uses to locally host the malicious website.
- *Two wireless adapter that are compatible with Kali* - One wireless adapter is used to host the fake access point, and the other adapter is used to deauth the real access point.
- *Python3* - This tool runs Python3 scripts.
- *Other tools* - Make sure to have the tools mentioned in "Tools Used" installed.

b. Running the scripts

1. Launch fake access point

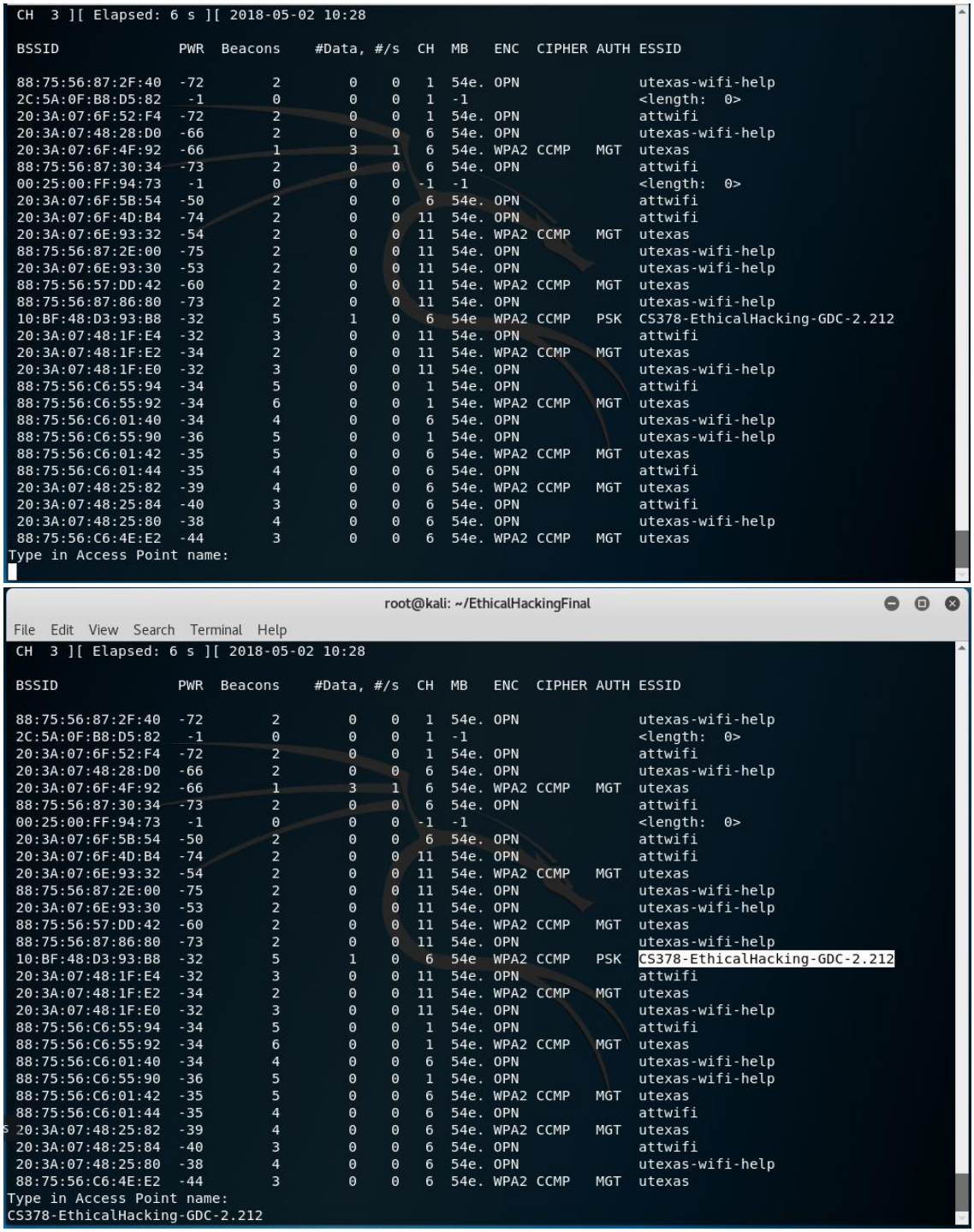
- Begin launching fake access point by running `hacking.py`:

```
root@jessethd:~/EthicalHackingFinal# python3 hacking.py
```

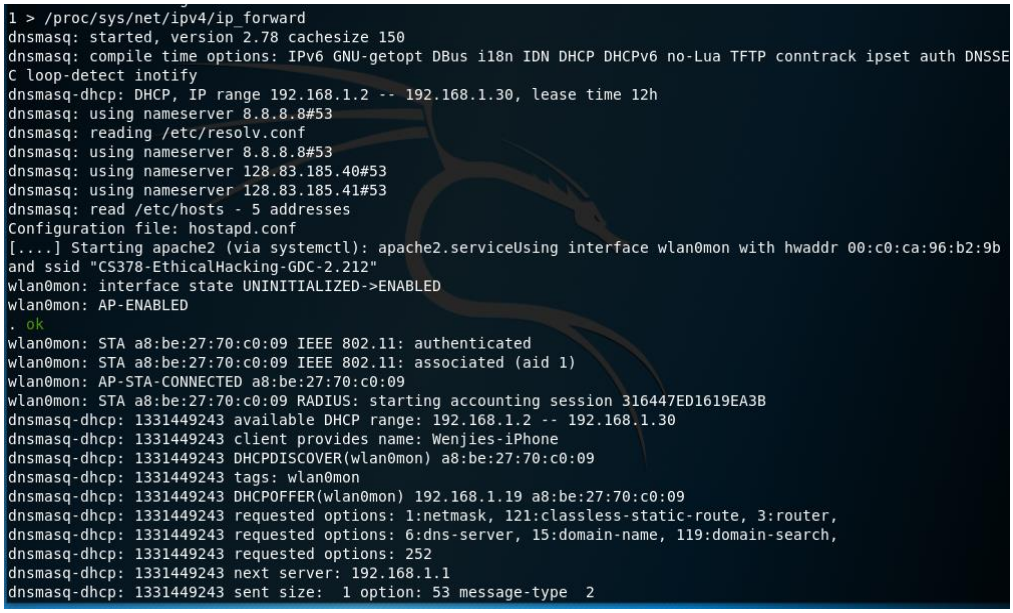
- The script begins by running `airmon-ng`, which starts up the wireless adapter in monitor mode as interface `wlan0mon`.

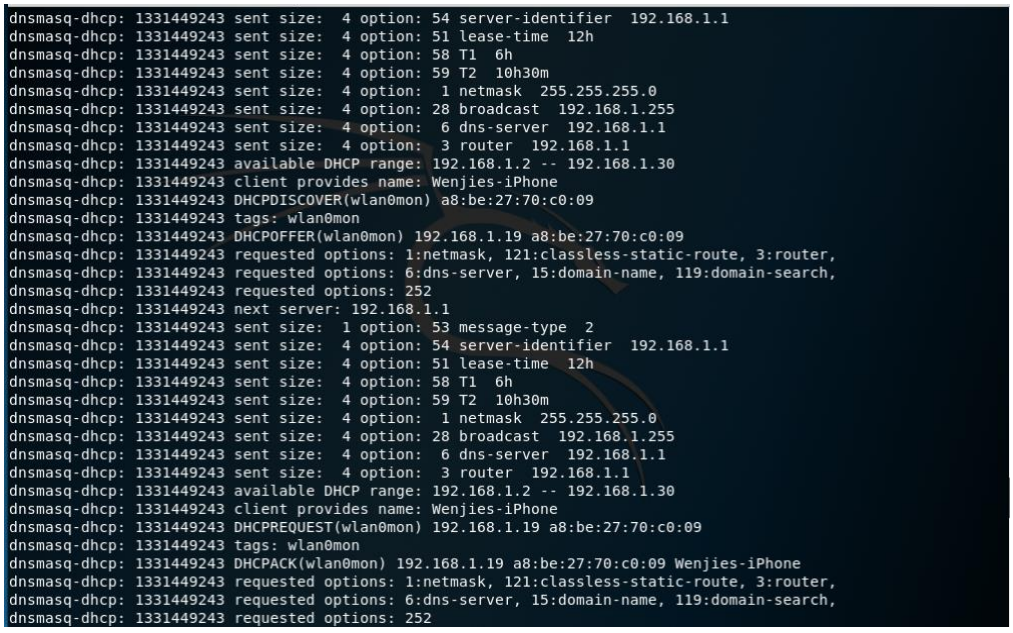
```
Found 3 processes that could cause trouble.
If airodump-ng, aireplay-ng or airtun-ng stops working after
a short period of time, you may want to run 'airmon-ng check kill'

  PID Name
  393 NetworkManager
  472 wpa_supplicant
  491 dhclient

PHY     Interface       Driver          Chipset

phy0    wlan0mon        rt2800usb       Ralink Technology, Corp. RT2870/RT3070
```

- The script then runs `airodump-ng`, scanning for nearby wireless access points for 10 seconds and writing the output to a file named `output-01.csv`. The user is prompted for the name of the access point displayed above that they wish to mimic.



- Once a name has been selected, the script begins the multi-step process of launching the access point. `hostapd` is started, writing its configuration into the `hostapd.conf` file in the local directory. This config file specifies the `wlan0mon` interface, the 802.11 physical layer, the 2.4 GHz frequency band, and the appropriate ESSID (name) and channel matching that of the selected access point. Next, the `iptables` are configured to help reroute all HTTP traffic to the Apache server hosted on our local machine. Finally, a local DHCP server is established through `dnsmasq`, with its configurations saved to config file `dnsmasq.conf`, also written in the local directory, and the Apache server is launched, loading up our malicious website to serve to all users of HTTP traffic on our network.





- All HTTP traffic on the server is monitored and reported. Once a WPA2 password is submitted to the malicious website (displayed below) by an unwitting client, it is printed to the terminal and written to the file `/var/www/html/log.txt`.

```
dnsmasq: query[A] espn.go.com from 192.168.1.19
dnsmasq: forwarded espn.go.com to 8.8.8.8
dnsmasq: forwarded espn.go.com to 128.83.185.40
dnsmasq: forwarded espn.go.com to 128.83.185.41
dnsmasq: reply espn.go.com is 35.168.187.154
dnsmasq: reply espn.go.com is 52.55.244.147
dnsmasq: reply espn.go.com is 54.81.98.150
dnsmasq: reply espn.go.com is 52.71.243.118
dnsmasq: reply espn.go.com is 34.194.75.200
dnsmasq: reply espn.go.com is 34.239.92.167
dnsmasq: reply espn.go.com is 34.197.11.167
dnsmasq: reply espn.go.com is 54.210.233.246
CS378-EthicalHacking-GDC-2.212
praetorian
```

- After capturing a password, the script terminates the `hostapd` and `dnsmasq` processes, shutting down the server.

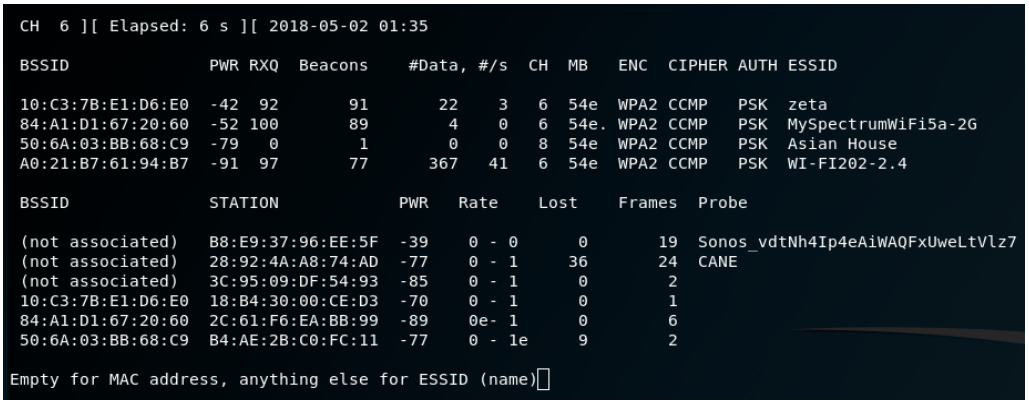
2. Deauthorize clients

- After launching the fake access point, run the `deauth.py` script to deauth clients on the real (mimicked) access point:

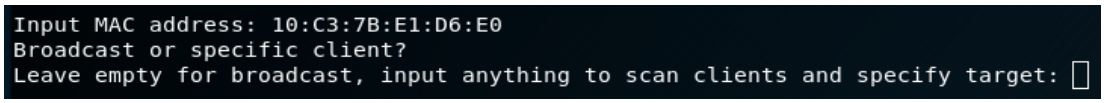
```
root@jessethd:~/EthicalHackingFinal# python3 deauth.py
```

- `deauth.py` will start `airmon-ng` and run `airodump-ng` for 10 seconds, scanning for all nearby access points and clients connected to those access points. Output is displayed to the screen and written to file `deauth_aps-01.csv`. Input an empty string to select target access point by MAC address, input any character to select target access point by

ESSID (name). Assuming fake AP is already running under the same ESSID as real AP, select by MAC address to specify correct target.

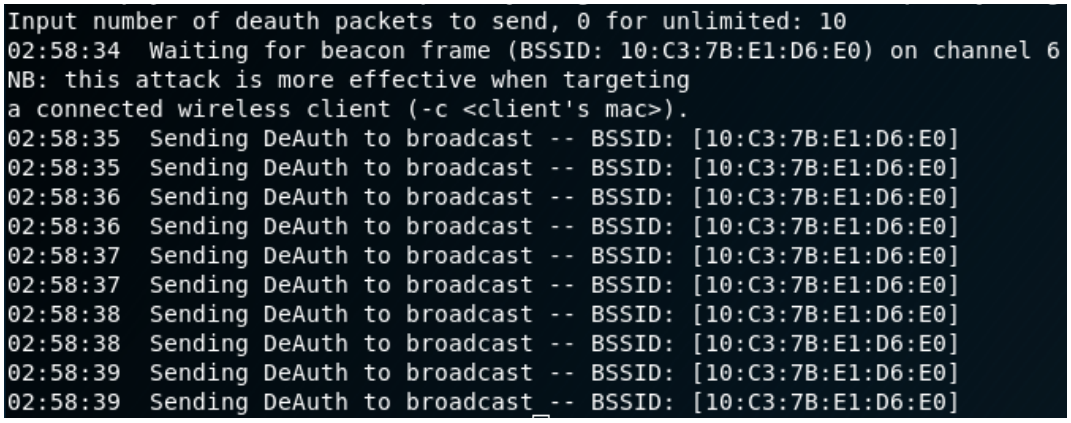


- Input empty string to send broadcast deauth packets to the target network, deauthorizing all clients connected to the network. Input any character to select a specific client to deauth. Sending broadcast deauths is recommended to maximize chance that a client will accidentally connect to the fake AP, but is very loud and may not be effective if a network is configured to ignore broadcast deauth packets.



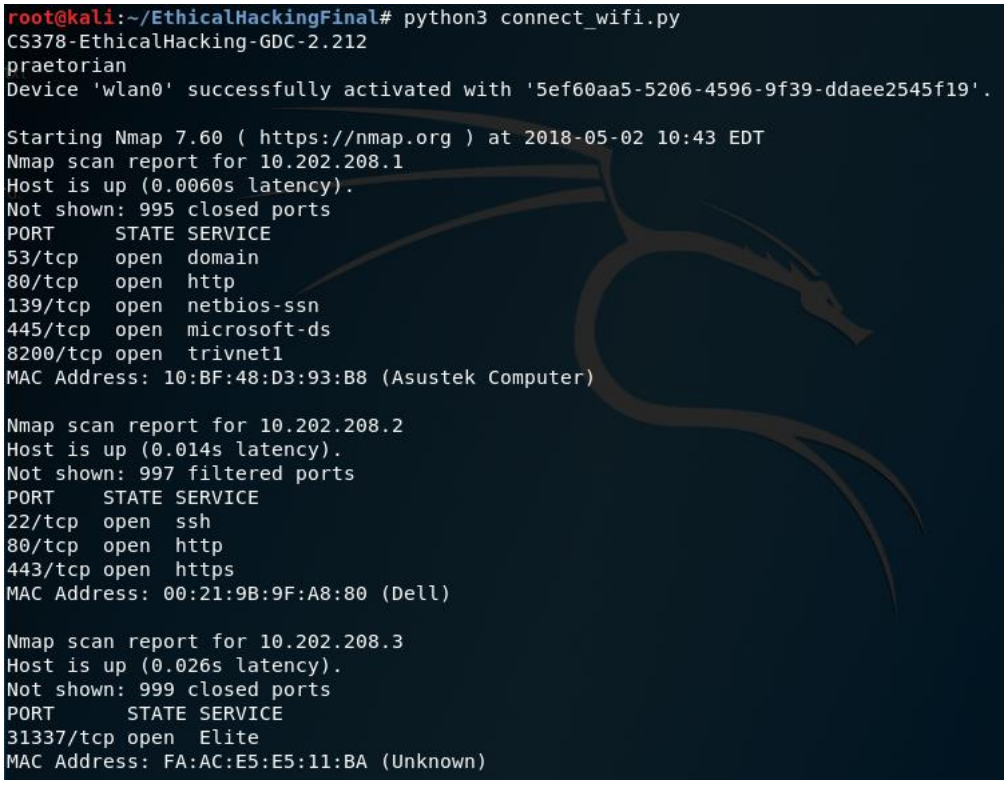
- Specify desired number of deauth packets to send to the targeted network. At least 10 recommended to ensure that packets reach all clients, or 0 to continuously send packets and effectively DOS the network, which may encourage connection to fake AP.

- Script will send requested number of packets to target AP and terminate.



3. Log into actual network and scan that network

- With the stolen credentials, run the connect_wifi.py script to automatically log into actual network and scan the network with nmap.



V. Problems encountered

Unfortunately, the design of our project does not cause the victim to auto-connect to our fake access point after being deauthenticated from the real access point. Auto-connect does not happen with our network even though it has the same name as the real network because the security scheme used is different (real network uses WPA, but fake one is open). However, making our fake network WPA protected is not an option because then the user wouldn't be able to connect to it.

A second major limitation to our tool is the need for two wireless adapters, which we couldn't figure out a way around. For some reason, when we use hostapd and run a deauth on the same wireless adapter, running a deauth causes hostapd to temporarily disable our "evil twin" access point. Initially, we tried to use airbase-ng instead of hostapd to host our fake access point, but airbase-ng wasn't consistently getting the fake access point to show up. Therefore, we resorted to making our tool use one adapter to deauth victims from the actual access point, and the other adapter to use hostapd to host the "evil twin" access point.

Another major problem that we encountered was allowing internet access to the victim. Ideally, what we wanted was to route the user to the internet after they enter credentials to our malicious website. By doing so, it will seem less suspicious to the victim. We couldn't get the rerouting to work, so the victim was unable to get internet access. We suspect it has something to do with how we are configuring our iptables, however we tried multiple configurations, and it still didn't seem to work. Our script even enables port forwarding, so we were unable to solve this issue properly.

The last major issue we came across was connecting to the actual access point with the credentials that were stolen. Our script initially performs an *airmon-ng start <interface>* to start the wireless adapter in monitor mode. When our wireless adapter is in monitor mode, it seems to be unable to send requests to the internet, and therefore is unable to connect to Wifi. Our team initially tried using wpa_supplicant by defining a configuration file that wpa_supplicant can use to manually connect to the internet. Unfortunately, it wasn't working properly, so we tried using nmcli instead. Nmcli worked properly, but is limited to only working when monitor mode is off (e.g. wlan0 and not wlan0mon). So our team solved this by either using *airmon-ng stop <interface>* to turn off monitor mode, or by manually reconnecting our wireless adapter to reset the mode. As a result, connecting to the actual access point with the stolen credentials was split into a different script file.

Overall, our team learned a lot from the problems that we encountered and from the project itself. We learned about the functionality of many new tools, as well as using them in a way that helps accomplish what we want. We also learned that it is very hard to circumvent a lot of the modern securities in place today, and it requires quite a bit of knowledge and creativity to create a tool that is practical.

VI. Ideas to improve functionality

Unfortunately, there are current limitations to our automation tool. The tool can be expanded on much further to improve its practicality and versatility. Below is a list of ideas that could be implemented in the future to further enhance our tool's functionality:

- *Use a database:* Currently, our tool writes the stolen Wifi password of the real access point to a log file. However, when you re-run our tool, the tool re-creates that log file. If a user of our tool wants to store passwords of multiple access points they wish to attack, then it would be possible with a database. Instead of writing to a log file, we can store the access points' name and password, to some database so that the user can store information about multiple networks.
- *Redirect HTTPS traffic:* Currently, our tool only redirects HTTP traffic. So if the victim happens to manually put in an HTTPS tag to the website they are trying to access, they won't be redirected to our malicious website. Most modern websites use HTTPS, so finding a way to properly redirect HTTPS traffic is important to improving the practicality of our tool. Perhaps using SSLStrip or getting some sort of valid SSL certificate could make this possible.
- *Internet Access:* It would be great if our tool could redirect the victim to the internet after they enter their credentials to our malicious website. This way, it will be more deceptive to the victim.
- *A way to handle differences in signal strength:* Right now, there is no sure way to make sure that our fake access point has a stronger signal than the real access point aside from being closer to the victim. Finding a reliable way to handle differences in signal strength will make our tool more consistent.
- *Expand on the Malicious Website:* The malicious website that we crafted to steal credentials is set up to look like a router page. However, in real life, that doesn't have too much practicality outside of working on people with that specific router. We can definitely expand our ability to steal credentials by crafting other types of sites. For example, we can use some sort of captive portal instead to make the website more convincing. We also don't have to limit our tool to just stealing Wifi passwords, we can also expand into other phishing scams (e.g. creating a fake Facebook page and steal credentials of accounts). Adding all of this extra functionality to our tool will help it be more deceptive and flexible.
- *Increase Ease of Use:* Our automation tool could definitely be improved to be more user friendly. Our tool currently uses three scripts: one script for deauthentication, one script for running hostapd, dnsmasq, and setting up Apache, and the last script for connecting to the real network and performing an nmap scan. If we can compact some of the scripts, then we can lower the amount of steps required in the "How to Use" section.