Introduction

Unlike wired networks, where a physical connection to the network infrastructure is required to eavesdrop on network traffic and activity, wireless networks use radio transmitters to broadcast traffic to any compatible device within range. Although the ease of access offered by wireless networks is one reason why they have been so widely adopted, it also makes them inherently insecure. By design, wireless networks are easily discoverable via beacons that advertise the network's SSID and capabilities.

To further compound this risk, during the early days of Wi-Fi, most vendors shipped their products to run "out of the box" without any default security controls. Although this may have streamlined the set-up process for end-users, it also led to the rise of an attack technique called *wardriving*. Wardriving refers to the process of using a laptop to search for unsecured Wi-Fi networks, often from the safety of a moving vehicle (hence "driving"). The term itself is a play on a much earlier technique called *wardialing*, where an attacker would automatically scan a list of phone numbers in search of modems that might provide a vector to other systems.

Since then, the war- suffix has been recycled many times and added to many other modes of motion, leading to terms like warjogging, warcycling, and warwalking. It was also applied to the term *warchalking,* a related activity that refers not to the mode of motion, but the activity performed upon discovery. Warchalkers signal vulnerable networks by marking areas with special drawings to indicate the security implementation (for example, mesh, closed, open, WEP). Inspired by hobo drawings, these symbols made it easy for those in the know to identify vulnerable networks anywhere someone had previously identified one - on sidewalks, in bathroom stalls, and on the side of buildings.

In this lab, you will make use of Mininet-WiFi, a wireless network emulation tool. You will use Mininet-WiFi to run several wireless network emulations, and then interact with various devices within these emulations to perform wireless network discovery, association/authentication, and decloaking. You will also harden a wireless access point (WAP, or just AP) by reviewing and adjusting various security configuration options in a generic AP web GUI (graphical user interface).

Lab Overview

SECTION 1 of this lab has three parts, which should be completed in the order specified.

1. In the first part of the lab, you will explore several common wireless security parameters in a generic access point web dashboard.

2. In the second part of the lab, you will participate in a wardriving scenario, wherein you will

make use of a wireless network scanner to collect data on all APs within range.

3. In the third part of the lab, you will make several changes to a wireless access point configuration to improve its security posture.

SECTION 2 of this lab allows you to apply what you learned in **SECTION 1** with less guidance and different deliverables, as well as some expanded tasks and alternative methods. You will learn more about the hidden wireless security control, as well as the tools that can be used to defeat it.

Finally, you will explore the virtual environment on your own in **SECTION 3** of this lab to answer a set of questions and challenges that allow you to use the skills you learned in the lab to conduct independent, unguided work - similar to what you will encounter in a real-world situation.

Learning Objectives

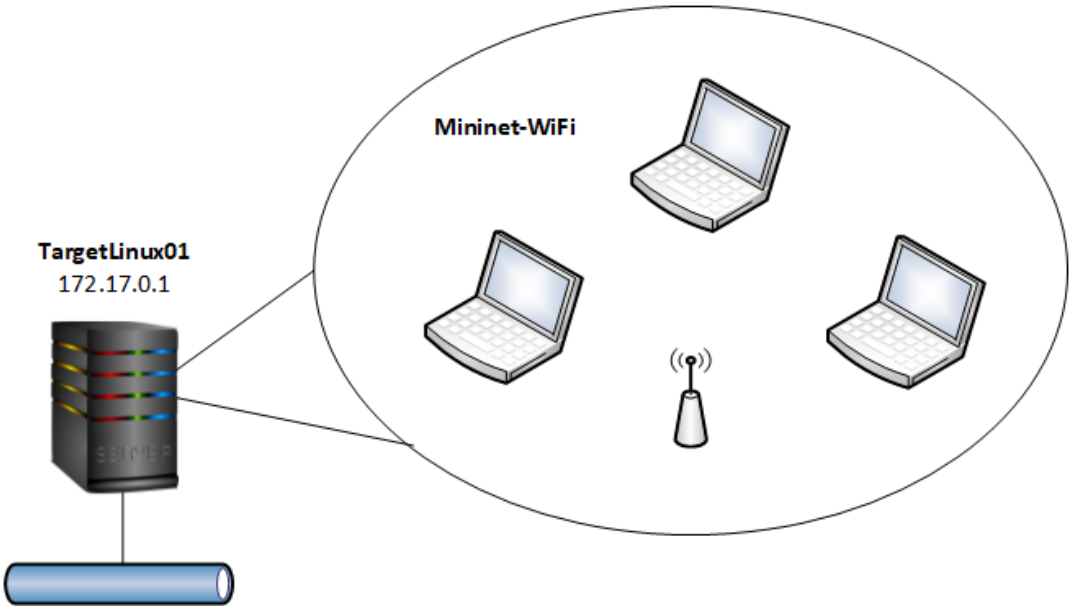
Upon completing this lab, you will be able to:

1. Configure and secure a Wi-Fi access point.
2. Scan the available Wi-Fi spectrum and identify prospective targets.
3. Understand the configuration flaws that enable wardriving techniques.
4. Understand the controls available to enhance WLAN security and prevent eavesdropping.
5. Evaluate, test, and compare viable solutions to WLAN eavesdropping.

Topology

This lab contains the following virtual machines. Please refer to the network topology diagram below.

- TargetLinux01 (Lubuntu 20)



Tools and Software

The following software and/or utilities are required to complete this lab. Students are encouraged to explore the Internet to learn more about the products and tools used in this lab.

- Mininet Wi-Fi
- LinSSID
- Kismet

Deliverables

Upon completion of this lab, you are required to provide the following deliverables to your instructor:

SECTION 1

1. Lab Report file, including screen captures of the following:

- Wireless networking configuration in GHostAPd Status screen
- Allow List in the Access Control Lists section of GHostAPd
- Log line corresponding to sta1's association with ap1 in the GHostAPd Log view.
- Multiple WLANs discovered in LinSSID
- New security mode and transmit power values for simplewifi on the GHostAPd Status page
- Relocated ap1 with 75% transmission power in the Mininet-WiFi Graph
- Both WPA2-PSK networks in LinSSID

2. Any additional information as directed by the lab:

- Record the GPS (x and y graph) coordinates of the ap1 access point

SECTION 2

1. Lab Report file, including screen captures of the following:

- List of five networks detected by Kismet
- SSID: youcantseeme probe request in your Node: sta1 Kismet window
- <youcantseeme> network details in your Node: sta2 Kismet window
- WPA2-enabled *Canteen* network as reported by the Status page
- Ap1's new location and signal range in the Mininet-WiFi Graph window
- New configuration values for the Canteen network on the Status page
- Configured ACL parameters, and any Attached Devices, as shown in the MAC Filtering page
- Sta1-wlan0 interface being denied authentication in the GHostAPd Log

2. Any additional information as directed by the lab:

- None

SECTION 3

1. Lab Report file, including screen captures of the following:

- Four discovered networks in Kismet
- Response to your curl/GET request
- Decloaked *TheGatesofHeck* network in Kismet.

2. Any additional information as directed by the lab:

- Document the SSID of the open hidden network you discovered.
- Document the SSID of the WEP-encrypted network you discovered.

Section 1: Hands-On Demonstration

Note: In this section of the lab, you will follow a step-by-step walk-through of the objectives for this lab to produce the expected deliverables.

1. **Review** the **Tutorial**.

   Frequently performed tasks, such as making screen captures and downloading your Lab Report, are explained in the Cloud Lab Tutorial. The Cloud Lab Tutorial is available from the User menu in the upper-right corner of the Student Dashboard. You should review these tasks before starting the lab.

2. **Proceed** with **Part 1**.

Part 1: Review a Wi-Fi Access Point Configuration

Note: For most users with a home Wi-Fi network, wireless access points (typically a combination modem-router) can be configured by a technician from the ISP during the initial on-site set-up process. This means that many users will never encounter their router's web interface. However, for security purposes, it is important to recognize that just because a user may have no need to access their Wi-Fi access point's web interface, an attacker would not be interested in it.

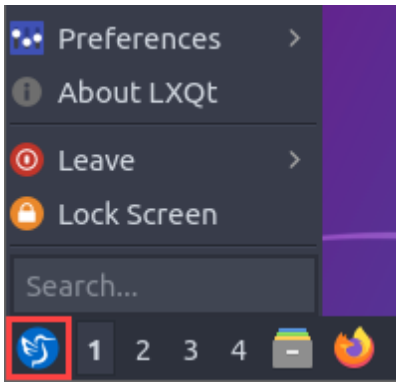
As an additional risk, many modern Wi-Fi access points are still configured with a default username and password. For example, if you Google "netgear default admin password" you can quickly confirm that all Netgear routers use a default administrator account with the username "admin" and password "password." On that note, it is worth disclaiming that although the custom web interface used in this lab does not require any authentication, this is not typical of any actual configuration interfaces, and is just a matter of simplicity for this lab. In the real world, most devices will require some type of authentication before granting access to their control board.

In the next steps, you will learn about some basic controls available for configuring a Wi-Fi access point through a web-based graphical user interface (web GUI), with options commonly available on consumer-grade devices. You will be exploring an emulated wireless network topology that you will run on TargetLinux01 (172.17.0.1).

You will begin the lab on the TargetLinux01 machine (172.17.0.1). User credentials are provided below for reference.

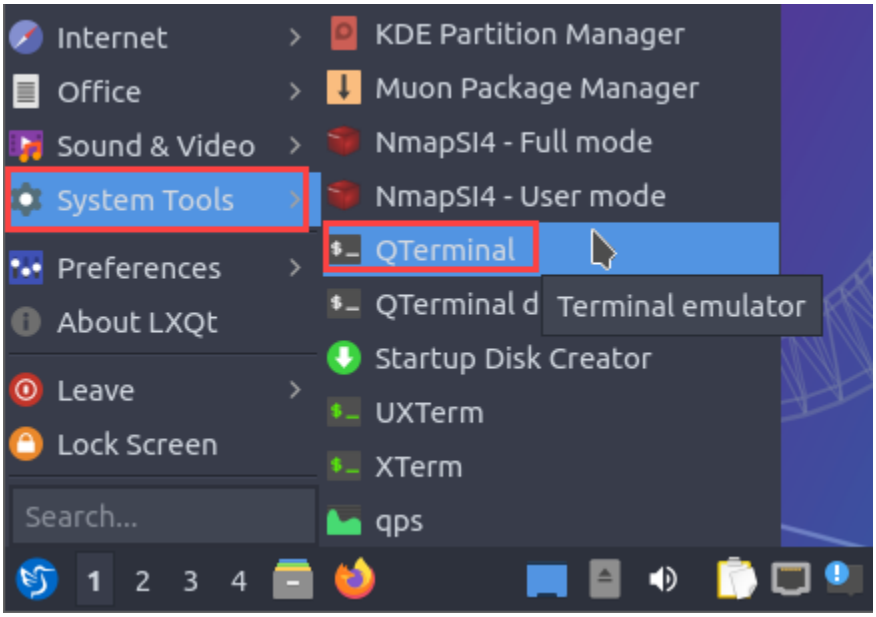
- Username: **`wifi`**
- Password: **`wifi`**

1. On the TargetLinux01 desktop, **click** the **Lubuntu icon** in the bottom-left to open the application menu.



Lubunutu icon

2. From the application menu, **navigate** to **System Tools > QTerminal** to launch Lubuntu's default terminal.



QTerminal

3. At the command prompt, **type** `cd Topos` and **press Enter** to change your current directory to /Topos.

```
wifi@TargetLinux01:~$ cd Topos
wifi@TargetLinux01:~/Topos$
```

Change your current directory

4. At the command prompt, **type** `sudo python wardriver.py` and **press Enter** to run this Python application with root (highest) privileges.

   When prompted for a password, **type** `wifi` and **press Enter**. Your password input will not be displayed to the screen, but rest assured that it is still being received by the console.

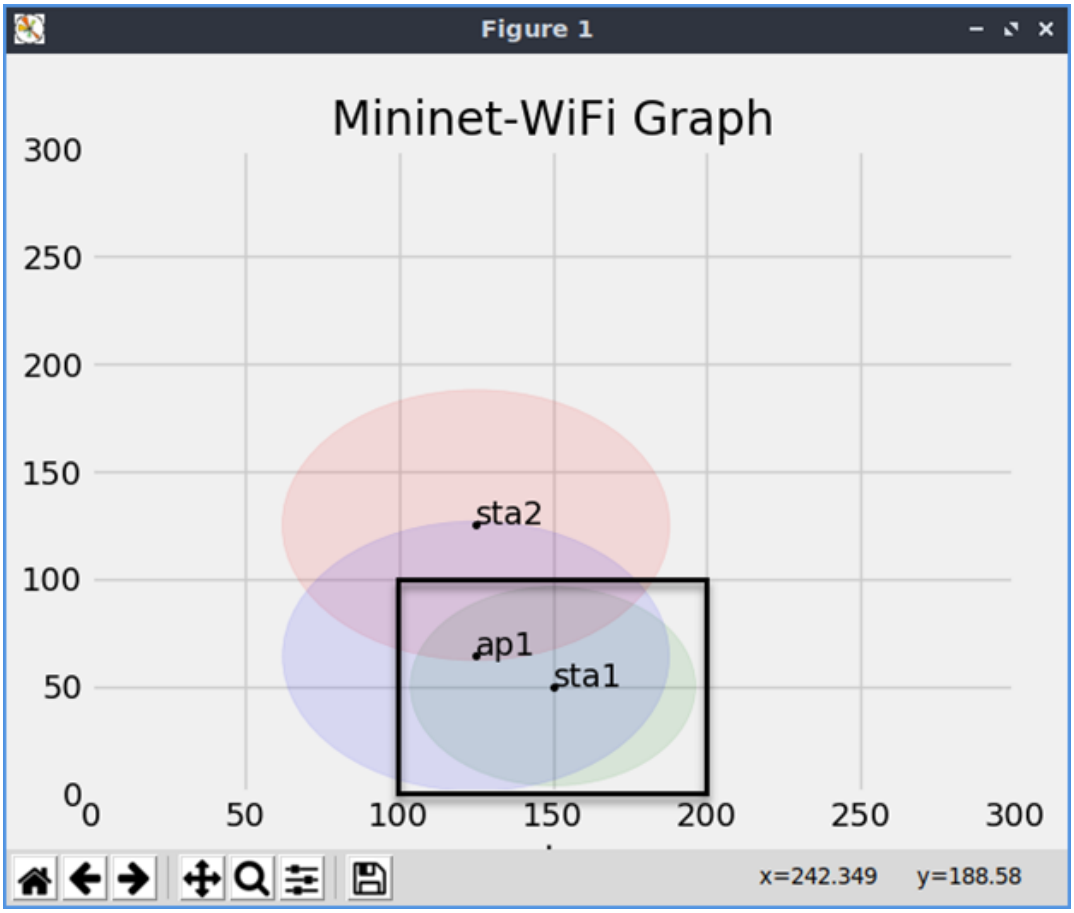
```
wifi@TargetLinux01:~/Topos$ sudo python wardriver.py
*** Creating nodes
*** Configuring Propagation Model
*** Configuring wifi nodes
*** Connecting to wmediumd server /var/run/wmediumd.sock
*** Starting network
*** Replaying mobility
*** Starting socket server
*** Running CLI
*** Starting CLI:
mininet-wifi>
```

Launch Mininet-WiFi

Note: The file you just executed describes a wireless network topology that will be built using Mininet-WiFi, an open-source wireless network emulator, the mechanics of which are outside the scope of this lab. The topology described in this setup consists of the following virtual hosts, which all reside in and are accessible locally on the TargetLinux01 machine:

- **Ap1**: 10.0.0.254/24
- **Sta1**: 10.0.0.2/24
- **Sta2**: 10.0.0.1/24

You will be greeted with a scene showing the wireless station, sta1, move from the left to the right side of the screen, then reverse to pull into range of ap1's signal. In this scenario, sta1 plays the role of a wardriving individual in search of local access points they can pick up from the street, while ap1 is the wireless access point for a home network, and sta2 is a legitimate device belonging to the administrator of that network. In this context, the perimeter of the home where ap1 and sta2 are located looks like this:



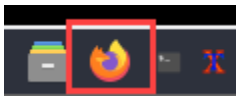
Mininet Graph

You will interact with this graph view more later, but your current aim is to inspect the wireless network configuration, so what you really want is a web browser. In the next steps, you will close this graph, then access the GHostAPd web GUI to determine whether the wardriving sta1 station had good

reason to double back.

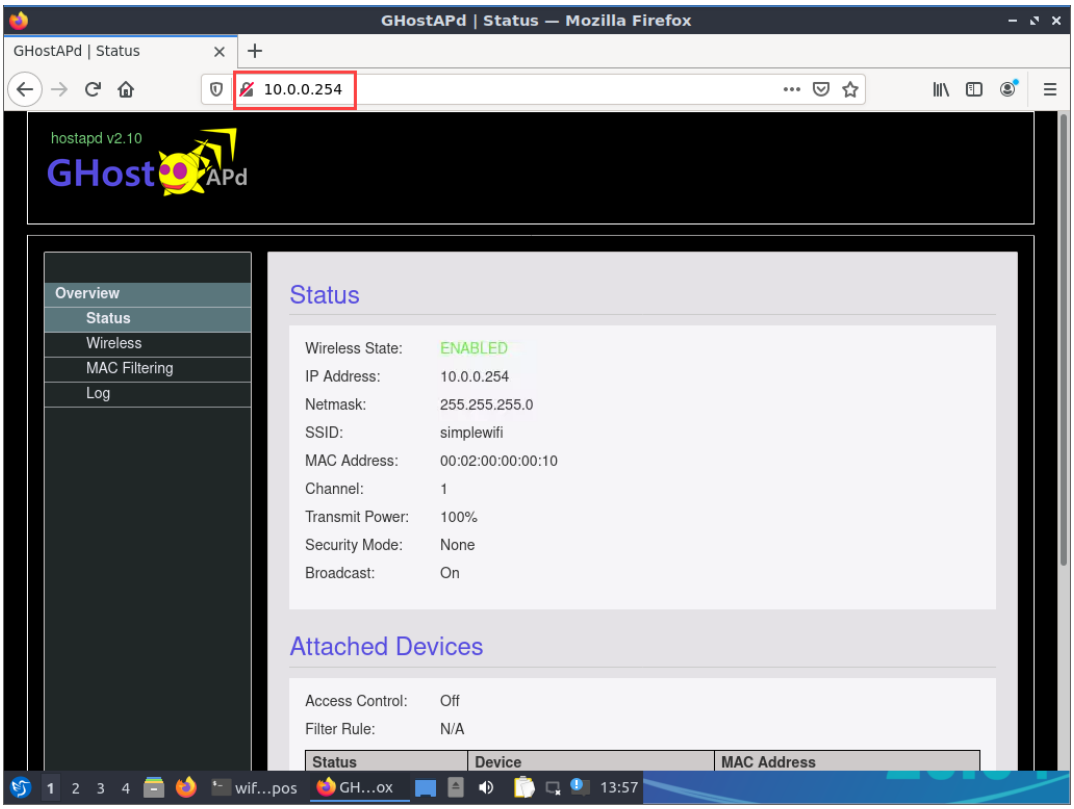
5. **Close** the **Mininet-WiFi Graph window**.

6. From the TargetLinux01 taskbar, **click** the **Firefox icon** to launch the Firefox web browser.



Firefox icon

7. In the Firefox navigation bar, **type** `10.0.0.254` and **press Enter** to navigate to GHostAPd, the WAP front-end for this deployment.



GHostAPd page

Note: Welcome to GHostAPd, the custom front-end for this wireless network. This interface provides a graphical way of interacting with Hostapd, a popular daemon available for implementing wireless access point functionality on Linux operating systems. It allows you to convert compatible network interface cards into access points and authentication servers, as has been done with ap1 in this wireless topology. Not all of the Hostapd functionality is exposed through this interface, though you will find most of the common configurable parameters provided on most generic access points. In the next steps, you will explore those parameters by examining the current configuration reported on the Wireless Status page.

8. In the Status section, **verify** that the **SSID is set to simplewifi**.

Note: The Service Set Identifier (SSID), or wireless network name, usually represented in natural language, is a human-friendly label for a wireless network. The alternative human-unfriendly label is the Basic Service Set Identifier (BSSID), which is the MAC address of the access point.

SSIDs are similar to DNS names (for example, google.com), which spare humans from having to remember the IP addresses for all their favorite websites.

9. **Verify** that the **Channel** is set to 1.

Note: Selecting an available and suitable channel is a very important step, because selecting the same channel as an existing WLAN in the vicinity may produce interference that adversely affects both WLANs. To discover nearby WLANs and look up their respective channels, you can use Wi-Fi discovery tools such as NetSpot, LinSSID, and Kismet.

10. **Verify** that the **Transmit Power** is set to 100%.

Note: This setting is intended to mirror the default configuration of many real access points, which are set to full channel power out of the box in the interest of maximizing RF coverage. However, this is not necessarily an advisable setting from a security perspective, because it may make your network more readily accessible to users outside the confines of your property – plus, it may annoy your neighbors. In a real-world situation, you would be well advised to use a power setting that does not propagate your signal beyond your property borders.

11. **Verify** that **Broadcast SSID** is set to On.

Note: In theory, disabling this setting prevents the access point's SSID from being included in the broadcast, which makes it less likely to be detected by tools such as NetSpot, LinSSID, and Kismet. Unfortunately, these tools can still detect a hidden WLAN SSID whenever a device that is already configured to connect to the hidden SSID initiates an exchange with the access point. For the purposes of this lab, you will leave this unchecked.

12. **Verify** that **Security Mode** is set to None.

Note: Get out your warchalk, looks like this access point is wide open – anyone can freely authenticate (prove their identity) and associate (pursue a connection) with it.

Disabling Wi-Fi security is necessary to making the access point vulnerable to the wardriving demonstration in Part 2. In the real world, you would want to ensure that WPA2/PSK security is enabled on any access point.

13. **Make a screen capture** showing the **wireless networking configuration in** GHostAPd **Status** screen.

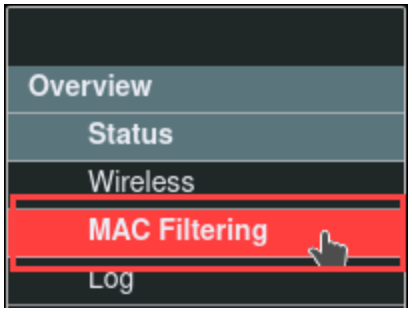
Note: The Attached Devices section at the bottom of this page also offers some valuable information: the IP addresses, DNS names, and MAC addresses of all currently attached wireless clients. It was presumed that sta2 would be connected, because this is its operator's own personal network, and sure enough you can see it there at 10.0.0.1/24 and 00:02:00:00:00:11. The real question was whether the wardriver connected as well, and it appears they have: sta1: 10.0.0.2/24 and 00:02:00:00:00:12. However, note that although the primary goal of wardriving is to locate and map vulnerable wireless access points to GPS coordinates (often through a platform like WiGLE), some actors are also interested in piggybacking – that is, connecting to and using the network without explicit authorization.

The MAC address is the most interesting data point here, because the IP address is only used once the device has successfully connected to that network – after all, it is a Layer 3 (Network) address, and therefore requires a network, which is exactly what the wireless client is seeking. Naturally, you obtain access to the Layer 3 network by authenticating and associating with the access point at Layer 2 (Data Link), where the 802.11 WLAN standards operate and MAC addresses are used as client identifiers.

This area also displays any access control lists and rules being enforced, which implement protection by governing which MAC addresses are permitted to connect to the network. Although MAC filtering can provide additional protection, that protection is easily overcome by MAC spoofing – assigning a different MAC address to your network card than the one issued by the manufacturer. A nefarious actor can scan the airwaves for network traffic, locate a valid MAC address for the network, and then assign that address to their own network card before pursuing a connection with the AP.

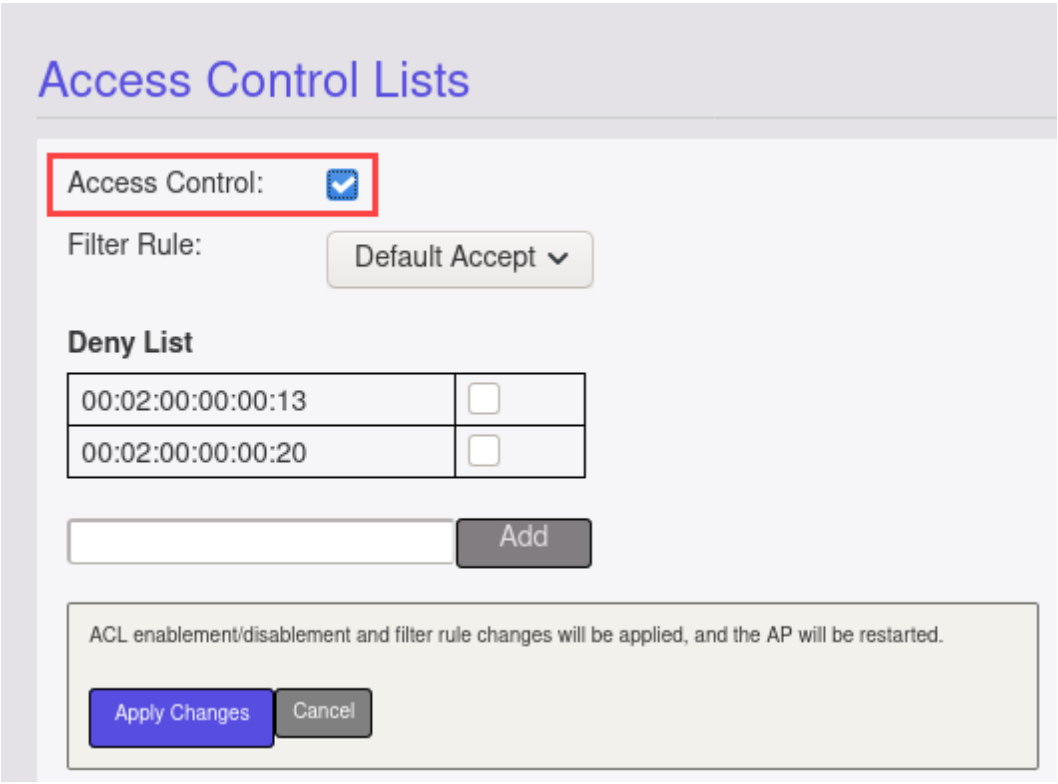
Hostapd implements MAC filtering through host allow and deny lists. This functionality is exposed by the GHostAPd web GUI in the MAC Filtering section, which you will explore in the next steps.

14. From the GHostAPd navigation menu, **click** the **MAC Filtering link** to open the Access Control List (ACL) configuration page.



MAC Filtering

15. In the Access Control Lists section, **click** the **Access Control checkbox** to reveal the ACL configuration options.



Access Control Lists

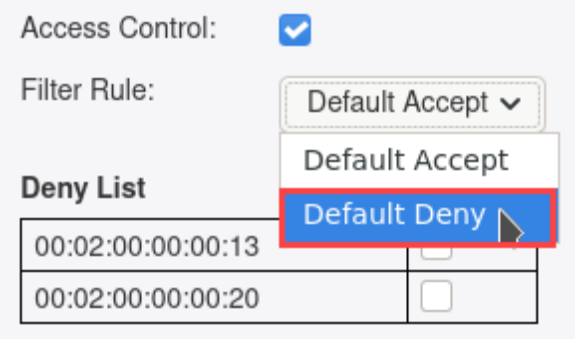
Note: Access Control Lists (ACLs) can be implemented in one of two filter modes:

- **Default Accept:** Accept connections from any MAC address except for those specified in the deny list.
- **Default Deny:** Deny connections from any MAC address except for those specified in the allow list.

As you can see, you are able to select your desired mode from the Filter Rule menu, and then manage it in the table below. Presently, the Block List is being shown because the Default Accept filter rule is selected. If Default Accept was selected, then the Deny List would be displayed instead. It is best practice to run in Default Deny mode, which forces you to allow each host manually, ensuring that only explicitly authorized devices (MAC addresses) can pursue a connection to the network.

You should also note the dialog box that appears at the bottom of the Access Control Lists section, which indicates that your selections have not been applied. You will not make any changes to the AP in this section – it is merely exploratory – so you should refrain from using the Apply Changes button at this time. You will return to this page later in the lab and use the Default Deny setting to harden the network.

16. In the Access Control Lists section, **select Default Deny** from the Filter Rule menu.

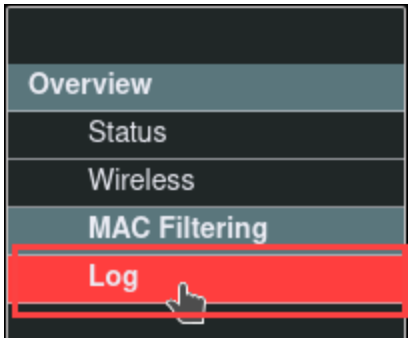


Filter Rule menu

Note: You should notice the table below changes to one called Allow List, with a single MAC address. The example MAC addresses here is arbitrary, as are the ones you saw in the Allow List, but on a real network they would be indicative of other devices that regularly connect to the network. In fact, the first 3 octets of a MAC address (00:03:04 in this case) are reserved for manufacturers, and therefore can be used to identify the type of device being used (assuming the MAC is not spoofed).

17. **Make a screen capture** showing the **Allow List in the Access Control Lists section of GHostAPd**.

18. From the GHostAPd navigation menu, **click** the **Log link** to view the wireless driver logs.



Log

Note: The Log is a simple page with a steady stream of information about your wireless device drivers. Here you can see changes in interface states, authentication events, and other wireless transactions conducted with the AP. The most recent log entries are shown at the bottom, which means that by scrolling down from the top, you can get a clear order of the events.

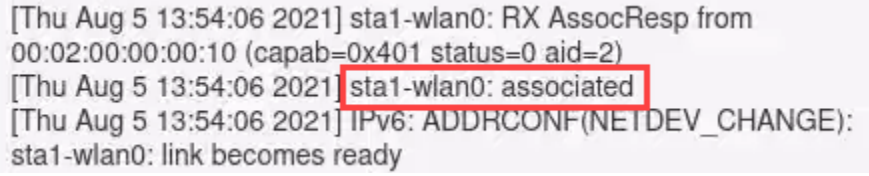
Following the log, you can identify moment the ap1-wlan1 link (ap1's wireless interface) became ready, which would have been right after you started this wireless topology. A couple lines down from this, you can see the moment sta2 (sta2-wlan0) initiated authentication with the AP, and follow that exchange right up to the actual association, *sta2-wlan0: associated*.

Note that even though this access point is wide open, authentication is still performed – this is called null authentication, and consists of a simple "can I connect?", "yep" exchange. This form of authentication is equivalent to a bouncer who is responsible for moving the ropes aside, but will move them for anybody.

You should also be able to identify the moment that sta1 connected too, shortly after the sta2

connection.

19. In the GHostAPd log, **locate** the line containing **sta1-wlan0: associated**, then **highlight it**.



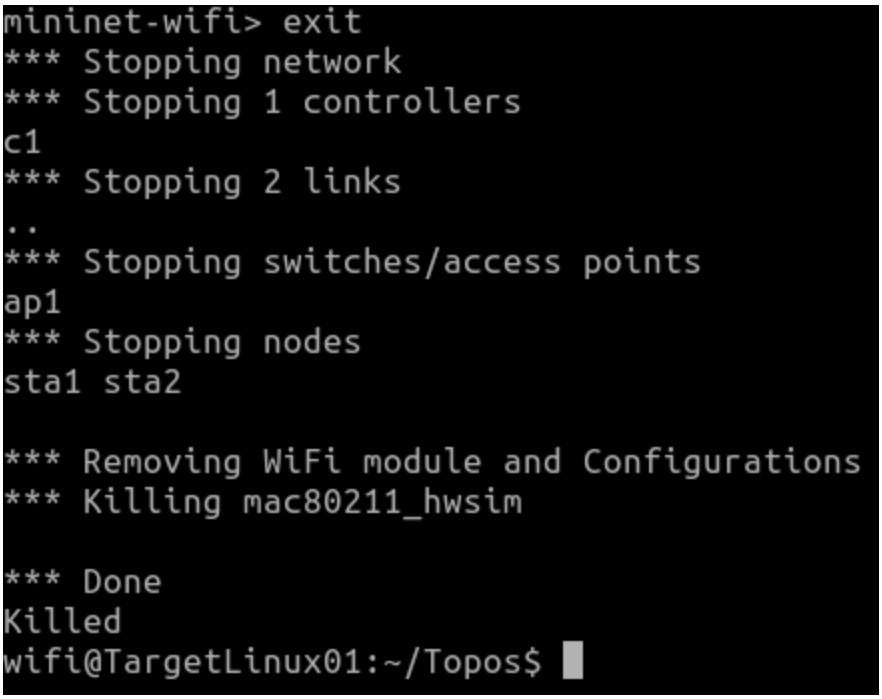
sta1-wlan0: associated line

20. **Make a screen capture** showing the **log line corresponding to sta1's association with ap1 in the GHostAPd Log view**.

21. **Close** the **Firefox browser window.**

Note: The next part requires a new Mininet-WiFi topology. You will follow the steps below to end and clean up the current emulated topology in preparation for your new topology. In the event you encounter an irrecoverable error or would simply like to reset the positions of the nodes and begin the emulation anew, you can use these steps at any time to reset the Mininet-WiFi topology you are working in.

22. At the command prompt, **type** `exit` and **press Enter** to shut down the current emulated wireless topology.



Shut down the current topology

23. At the ~/Topos$ command prompt, **type** `sudo mn -c` and **press Enter** to perform a thorough clean-up of all residual configuration data from previous topologies.

    If prompted for a password, **type** `wifi` and **press Enter**.

```
wifi@TargetLinux01:~/Topos$ sudo mn -c
[sudo] password for wifi:
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd ovs-controllerovs-t
estcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd ovs-controllerov
s-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
***  Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.

*** Removing WiFi module and Configurations
wifi@TargetLinux01:~/Topos$
```

Perform a clean-up

Note: Once the script has finished running, you will be returned to the ~/Topos$ command prompt. You are now ready to spin up a new wireless topology for Part 2.

Part 2: Perform a Wardriving Attack with LinSSID

Note: In Part 1 of this lab, you reviewed the configuration for a Wi-Fi access point with wireless security disabled. Disabling wireless security is widely regarded as a terrible idea, but it was not long ago that this was the default configuration for new out-of-the-box access points. These permissive default settings – intended to facilitate ease of use for a relatively new consumer technology – are what enabled the rise of wardriving in the first place. Fortunately, the danger of leaving wireless networks unsecured has since been recognized, and although wardriving still persists as an attack technique, the attackers using it are far less successful than they were 20 years ago. However, this is no reason to disregard it as a viable attack vector. An attacker only needs to get lucky once to breach a network.

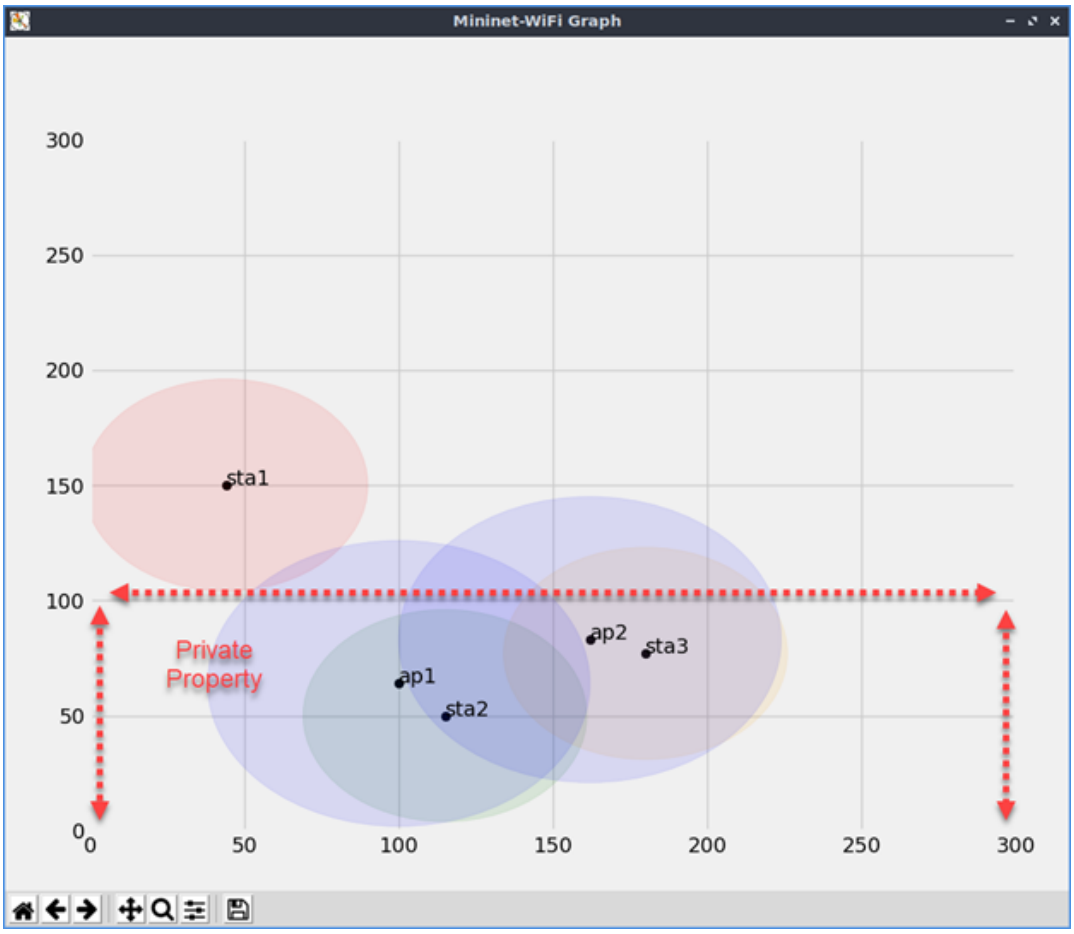
In the next steps, you will use Mininet to instantiate another wireless network. In this new network, you will play the part of the wardriver looking for open access points to map using some common wardriving tools and techniques. Once found, you will inspect their values in LinSSID, and then finish mapping them by recording their GPS (graph) coordinates.

1. At the command prompt, **type** `sudo python wardriver-multi.py` and **press Enter** to start a new wireless network emulation.

   If prompted for a password, **type** `wifi` and **press Enter**.

Note: This network topology bears some resemblance to the first, with a few adjustments. First, you will find the wardriver, sta1, is back, but did not place itself in range of any access points this time. You will see that sta2 is still at home, connected to their wireless network. You will also see some obvious changes, like the additional access point called ap2, as well as a new station, sta3, which is connected to the ap2 network. These will serve as additional targets for your wardriving exercises.

In the next steps, you will assume the role of the wardriver and run a wireless discovery tool to search for nearby vulnerable access points. In this graph scenario, the premise is that all squares below the horizontal line y=100 represent private property, and so any unwelcomed advances below this line would constitute trespassing.

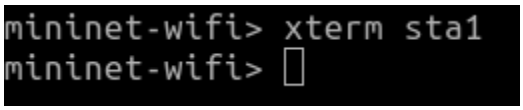


Mininet graph

Also, note that these circles suggest a uniformity not present in a true wireless environment, where real-world conditions would cause great variance in the signal range. That said, this wireless topology has been designed to emulate imperfect conditions through signal fading (signal attenuation/reduction due to obstacles, distance, and random fluctuations), so some of this fuzziness is baked into the emulation.

2. **Minimize** the **Mininet-WiFi Graph window**.

3. At the command prompt, **type** `xterm sta1` and **press Enter** to open a new terminal window from the sta1 device.

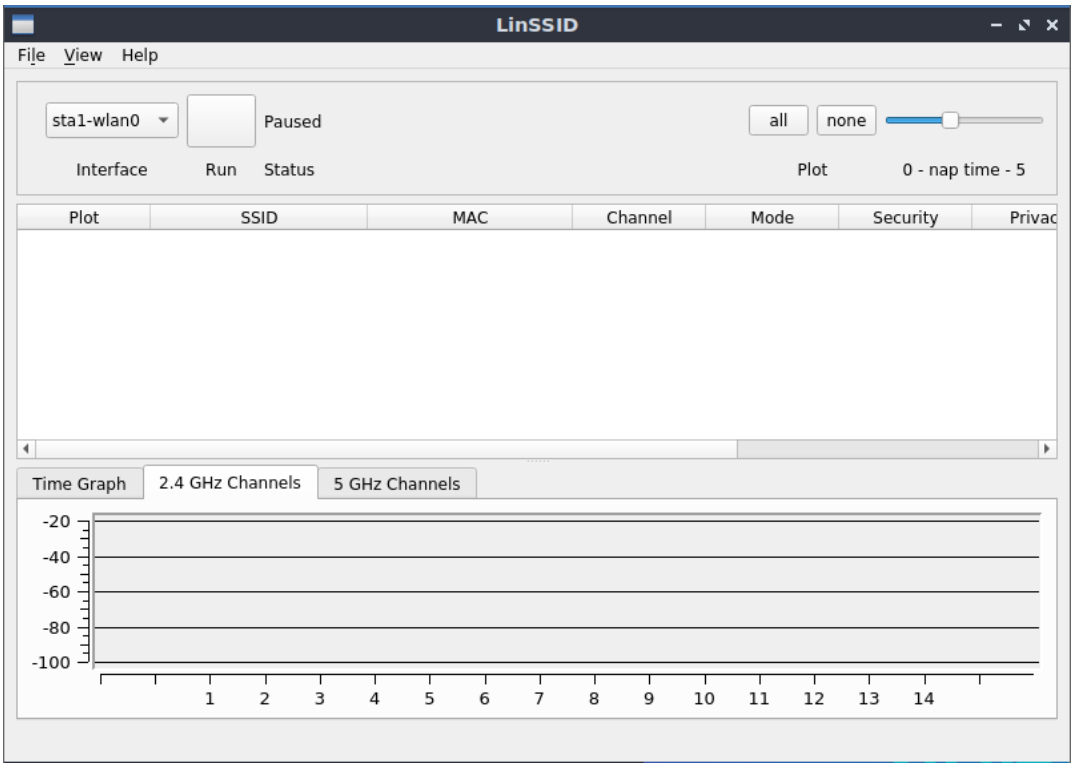


Open a new terminal window

4. At the sta1 terminal window, **type** `linssid` and **press Enter** to launch LinSSID from sta1.

Launch LinSSID

Note: LinSSID is the tool you will use to perform your wardriving attack. LinSSID includes a GUI that makes it very easy to discover WLANs. Access points will broadcast their SSID(s) and the WLAN capabilities via beacons roughly once every 100 milliseconds (10 times per second). These beacons advertise the networks to potential clients, which is how LinSSID and other tools are able to find them.

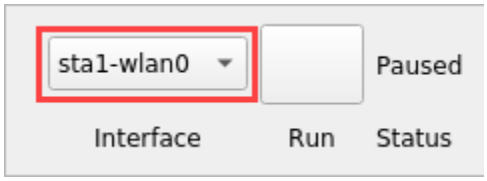


LinSSID

From the LinSSID GUI, you can see the relevant information about any discovered WLANS, including the SSID, Channel, and Privacy & Cipher. The goal of wardriving is to find open WLANs that can be accessed and potentially compromised. The easiest way to do this is to locate WLANs that do not have wireless security enabled.

If the LinSSID window appears partially off-screen, **hold** the **alt key**, and then **left-click-and-hold** on any blank space within the window and drag it into view.

5. In the LinSSID window, **verify** that **sta1**-wlan0 is selected from the Interface menu.



Interface menu

6. In the LinSSID window, **click** the **Run** button to begin scanning wireless networks using the sta1-wlan0 interface.

Run LinSSID

Note: LinSSID will scan for and display relevant information about any discovered WLANs, notably the SSID, Channel, and Privacy & Cipher. If you go to the top toolbar and select View, you can select additional attributes to display, such as the access point's vendor, mode, or the time it was first/last seen.

You are unlikely to pick up any networks from your current position. However, you may not actually need to move an inch for this to change, because there are other ways of expanding your wireless field of view, such as increasing your antenna gain.

Wardrivers typically employ omnidirectional antennas during their drives in order to get 360-degree coverage of the area (in contrast with a parabolic or dish antenna, which has high directivity). A truly omnidirectional antenna (or *isotropic* antenna), one that radiates a signal in a perfect 360-degree sphere (equally sensitive to signals in all directions), does not exist – at least, not anywhere else but in theory. In contrast, their real-world manifestation, dipole antennas (or more commonly *omnidirectional* antennae, when the distinction is clear) – the jointed black plastic sticks you see on most consumer-grade wireless network devices – radiate in more of a donut pattern, which are obviously imperfect implementations of their theoretical counterpart. This imperfection results in a narrower but more concentrated beam, which provides wider horizontal coverage of the area, but reduced vertical coverage (effectively squishing the donut). This quality is referred to as gain.

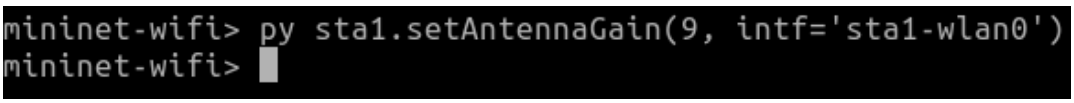
The perfect omnidirectional (isotropic) antenna has a gain of 0dBm, because it radiates uniformly in all directions. In other words, there are no relatively stronger or weaker areas in the radiation pattern. This ideal model is then used as a reference point for gain measurement in real-world antennae. For example, basic dipole antennas will have a standard gain of 2.15 dBi, or 2.15 decibels per isotropic – in other words, "2.15 decibels more power than the isotropic antenna." You may also see the measurement dBd, which is the same concept, but instead uses a real-world dipole antenna for the reference base unit (dBd = dBi – 2.15), instead of its ideal hypothetical version.

It is common for wardrivers to use antennas with high gain, although this is not strictly better. In metropolitan environments with tall buildings, it is advisable to use an antenna with lower gain so that the beam is radiated at higher angles, thereby covering more vertical space. In an area with a less impressive skyline – perhaps a suburb populated with bungalows – you might opt for a higher gain antenna, possibly a 7 or 9 dBi model, to steal some radiated energy from the higher signal beam angles to intensify the lower ones.

You can presume the graph scenario represents a residential area with low houses and some good views of the sky. In the next steps, you will simulate exchanging your current antenna for a higher gain antenna in hopes that this change will provide LinSSID with its first hit.

7. **Restore** the **wifi@TargetLinux01:~/Topos terminal window**.

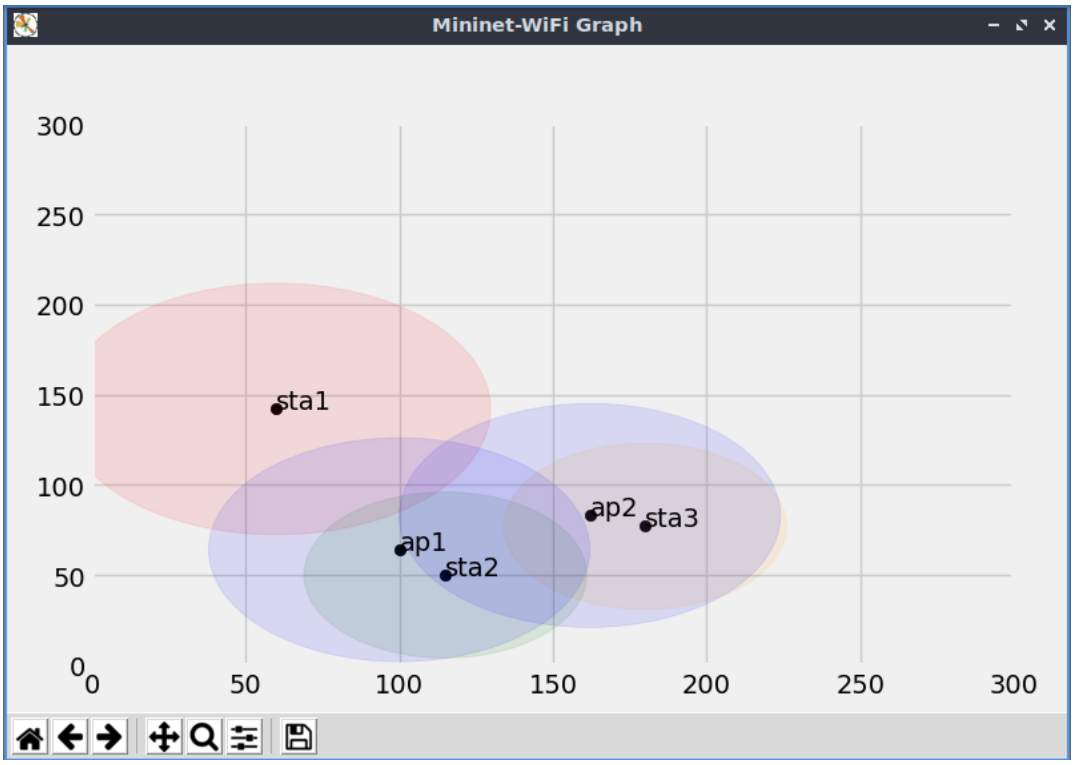
8. At the mininet-wifi> prompt, **type** `py sta1.setAntennaGain(9, intf='sta1-wlan0')` and **press Enter** to change the antenna gain to 9 dBi on the sta1-wlan0 interface.



Change the antenna gain

Note: This step simulates swapping in a new, higher-gain antenna by utilizing the Mininet-WiFi's setAntennaGain() function. Given that 9dBi is on the higher end, this should increase your horizontal reach by a good margin.

9. **Restore** the **Mininet-WiFi Graph window** to review your antenna gain change in the graph.

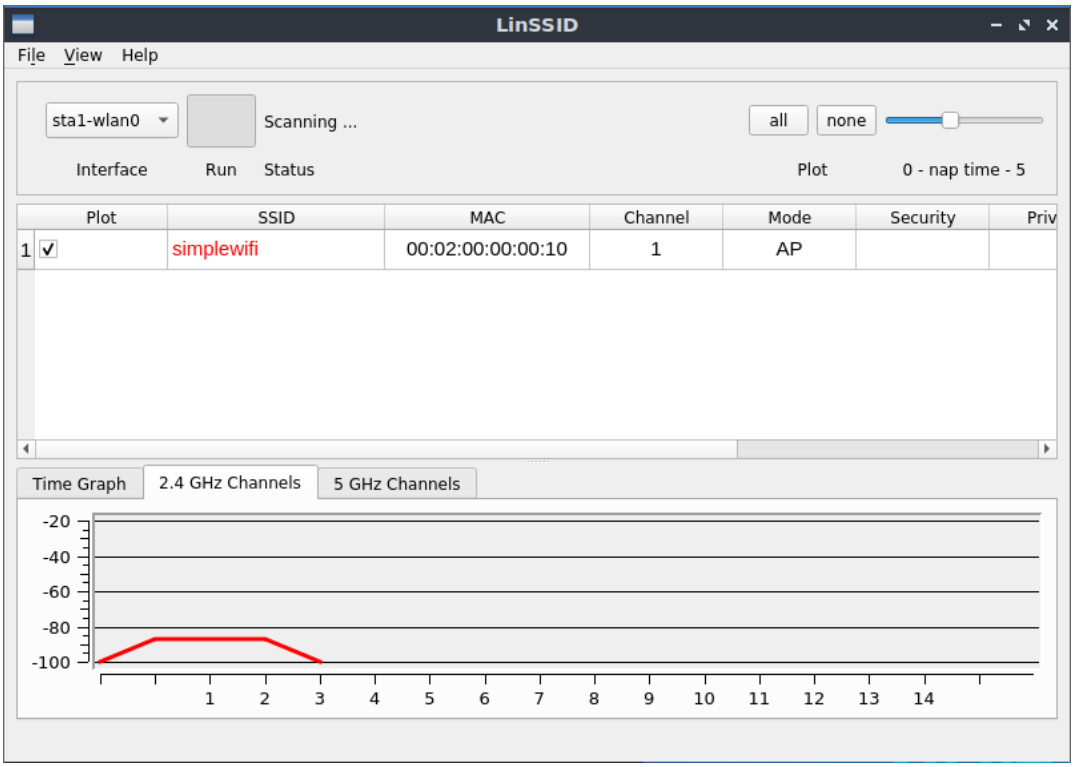


Updated graph

10. **Make a screen capture** showing **sta1's increased antenna gain as displayed in the Mininet-WiFi Graph**.

11. **Minimize** the **Mininet-WiFi Graph window**.

12. **Restore** the **LinSSID window**.



LinSSID results

Note: Got one! The network called *simplewifi* may look like the same one that you reviewed in Part 1, but the context is different here: you are on an official (pretend) war driver. This means that you should inspect this network to confirm that its security mode remains unchanged (disabled), and then map it like any good wardriver.

If no network is discovered after a minute, click the Run button to pause your scan. Then, click the Run button again to resume it, and resume waiting for discovery.

13. In the LinSSID window, **inspect** the **Privacy column value** to determine the encryption implementation on this network.

    In this case, the fact that no values are listed for the networks indicates this network is unprotected.

Note: Modern encryption algorithms are very difficult to crack, which is why the objective of wardriving is to locate WLANs with wireless security disabled. Although a lack of security controls is an important consideration when selecting a target, it is not the only one. Signal strength is also important, because

there is little to be gained by connecting to a WLAN with a weak signal – if that were the intention.

14. In the LinSSID window, **inspect** the **Signal Level column header**.

Note: Signal level is measured in dBm. With Wi-Fi, the usable signal range is typically from -30 dBm (excellent) to -97 dBm (very poor). However, because Wi-Fi devices are rarely located directly next to the access point, the typical healthy signal range is considered between -55 dBm and -70 dBm. Given that interference is baked into this emulated environment, you can also observe signal fluctuations over time via the Time Graph in the lower half of LinSSID. You can also sort the current signal strength by channel in the 2.4 GHz and 5 GHz channel tabs.

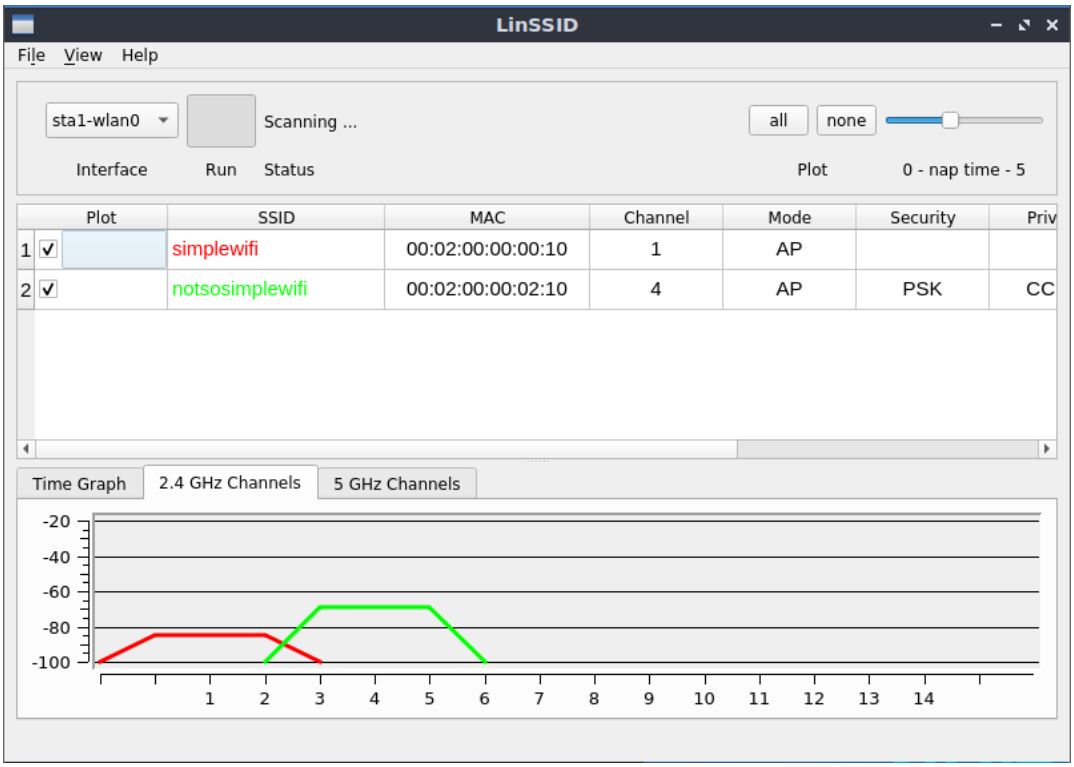
In this case, the signal is in the “very poor” range – you know it exists, but good luck getting a connection. And this is fine, because you are still just in wardriving mode, and so are only interested in collecting APs and their coordinates. But in that effort, it may be worth getting a little closer to the perimeter – maybe you can populate LinSSID with some additional networks.

15. **Restore** the **wifi@TargetLinux01:~/Topos terminal window**.

16. At the mininet-wifi> command prompt, **type** `py sta1.setPosition('150,105,0')` and **press Enter** to center sta1 between the two property buildings and pushed right up to the perimeter.

Move the sta1 device

17. **Restore** the **LinSSID window**.



Updated LinSSID results

Note: After a few moments, your LinSSID view should populate with an additional wireless network: *notsosimplewifi*. You may also see a small improvement in your signal to the *simplewifi* network, because you are slightly closer to it in your new position. However, the relative change in distance was fairly small, and the random signal fading in the environment, as well as the additional interference from sta2 when coming from the right side, makes the difference negligible – or at least difficult to appreciate right away. Additionally, you would still want to get closer if you had any intention of maintaining a reliable connection to the network, because your signal likely still falls outside the range of a healthy wireless network connection.

This new network, *notsosimplewifi*, provides a pretty good signal. It also delivers on the promise of its name, because it appears to have some security options enabled. The new values include:

- **Channel:** On the 2.4GHz Wi-Fi band, there are 14 channels (only 11 of which are designated in North America), but only 3 channels (1, 6, and 11) do not overlap with any other channels. For this reason, you will find the majority of APs on one of these channels, making them among the busiest. In this case, *notsosimplewifi* is operating on channel 6, whereas *simplewifi* is on channel 1 – both non-overlapping.

- **Security:** PSK indicates authentication is performed via a pre-shared key – a secret shared outside of the authentication channel between the AP and any trusted wireless clients. It looks like you are dealing with Wi-Fi Protected Access 2 Personal (WPA2-PSK or WPA2-Personal), which is no good for your wardriving efforts. If you were interested in sticking around to crack this, you better hope they have some really old firmware, or a really weak PSK.

- **Privacy & Cipher:** CCMP, or the Counter Mode Cipher Block Chaining Message Authentication Protocol. This mouthful is the message authentication and encryption mechanism implemented by WPA2, which uses the AES block cipher algorithm.

There is no point in spending any more time with the *notsosimplewifi* WLAN, because it appears to be fairly well guarded, but at least you found one new vulnerable AP worth mapping. Now you just need to record the GPS coordinates specifying its location. LinSSID can provides this data, assuming you have a compatible GPS card with you, but there is no such card in this emulation, so you will have to settle for graph coordinates.

18. **Make a screen capture** showing the **multiple WLANs discovered in LinSSID**.

19. **Close** the **LinSSID window**.

20. **Minimize** the **Node: sta1 window**.

21. **Restore** the **wifi@TargetLinux01:~/Topos terminal window**.

22. At the mininet-wifi> command prompt, **type** `py ap1.position` and **press Enter** to display the GPS (graph) coordinates of the unsecured access point, ap1.

    The third coordinate refers to the z-axis. There are no 3D graphs in this demonstration, so it will always be zero.

23. **Record** the GPS (x and y graph) coordinates of the ap1 access point.

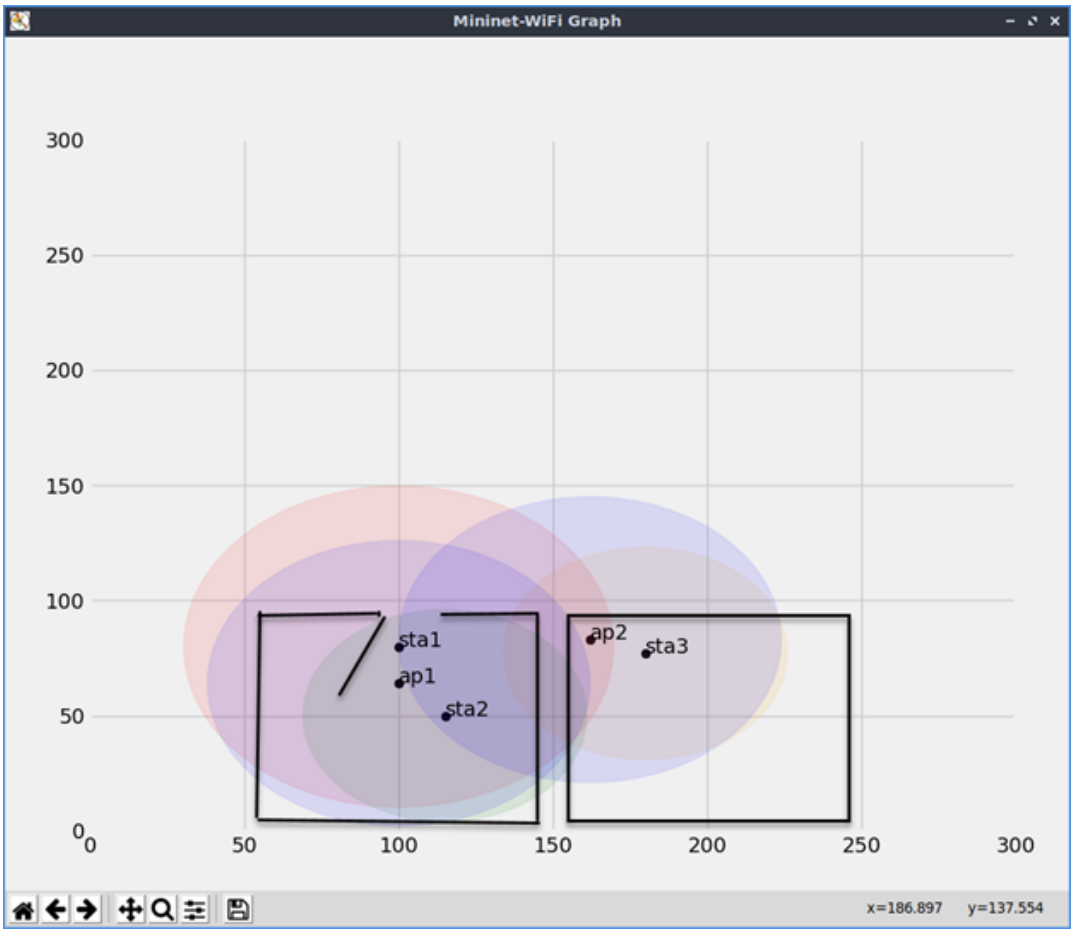
Part 3: Secure the Access Point

Note: In this part of the lab, you will learn how to mitigate a wardriving attack by implementing some basic security controls on an access point. While your previous wardriving run was a thrill, you have decided to help out the owner of the *simplewifi* network and show them how to secure their network more effectively.

Suppose that the network admin for *simplewifi* was using old hardware that only supported WEP, and opted not to configure that, which is why the network is currently exposed. Lucky for them, you happen to have some an extra access point that supports 802.11i (WPA2), which saves you having to begrudgingly deploy WEP as a last resort. In the next steps, you will walk them through the configuration so that you only have to do it once. First, you will relocate your laptop within the property lines to get a better signal, then you will access the access point's web GUI to improve the device's security posture.

1. At the mininet-wifi> command prompt, **type** `py sta1.setPosition('100, 80, 0')` and **press Enter** to enter the property and stop just north of the ap1 access point.

2. **Restore** the **Mininet-WiFi Graph window** to review your changes in the graph.



New Sta1 position

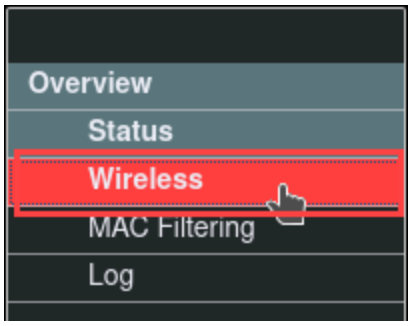
3. **Minimize** the **Mininet-WiFi Graph window**.

4. From the TargetLinux01 taskbar, **launch** the **Firefox web browser**.

5. In the Firefox navigation bar, **type** `10.0.0.254` and **press Enter** to open the wireless access point's web GUI.

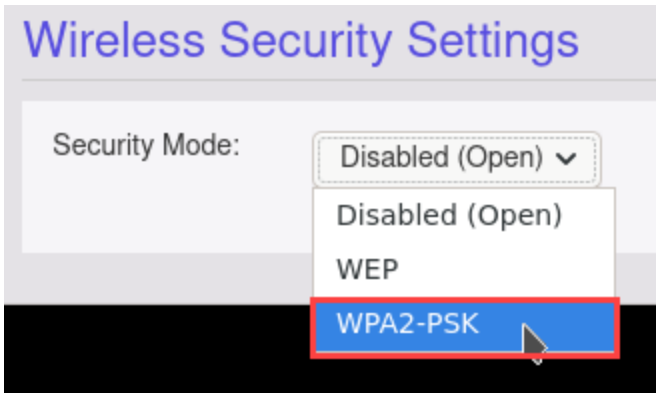
Note: Okay, so now you are back in GHostAPd, but this time you are going to make some changes. First, you will add some encryption.

6. From the GHostAPd navigation menu, **click** the **Wireless link** to open the Wireless Network and Security Settings page.



Wireless menu

7. In the Wireless Security Settings section, **select WPA2-PSK** from the Security Mode dropdown menu.



Security Mode

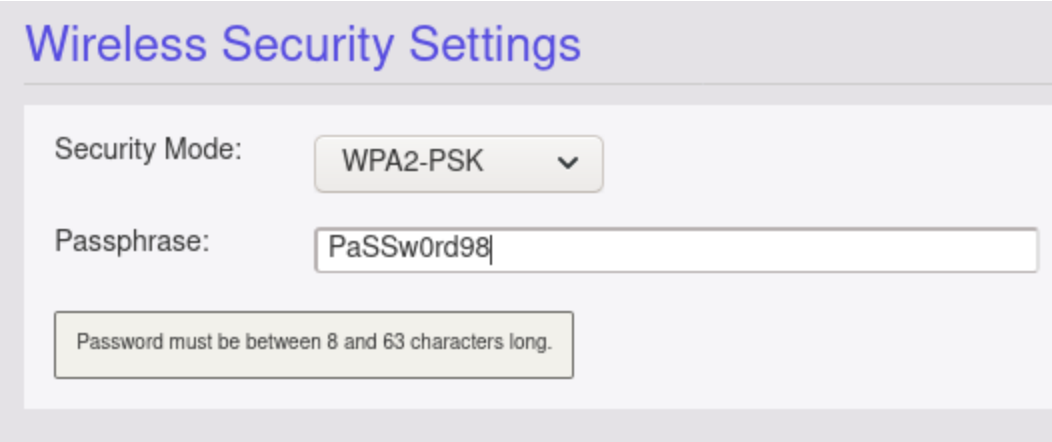
Note: Wi-Fi Protected Access 2 (WPA2), often used as shorthand for the IEEE 802.11i wireless security standard, was ratified in 2004 and put an end to the woes created by its problem-riddled predecessor, WEP. WPA2 is still the most widely-deployed wireless encryption mode today (circa 2021), with the majority of these being WPA2-Personal implementations, the flavor most often used in SOHO (small office/home office) networks. WPA2-Personal uses a simple pre-shared key to

authenticate clients and secure the network, and for that reason is also commonly referred to as WPA2-PSK.

The popularity and longevity of WPA2 are for good reason, as WPA2 provides a cryptographically formidable solution through its use of the Advanced Encryption Standard (AES) block cipher algorithm, established by the National Institute of Standards and Technology (NIST) within the secure CCMP encryption protocol (CCMP, an acronym you may remember from LinSSID earlier in this section).

Unsurprisingly, WPA2 did produce some vulnerabilities over the years, so having updated firmware is still of great importance, but ultimately the strength of a WPA2-Personal network lies in the length of the key used to secure it. With that being said, we will use a simple key in this scenario to spare you from typing out a long string of random characters.

8. In the Wireless Security Settings section, **type `PaSSw0rd98`** in the Passphrase field to assign a password to the access point.



Passphrase

Note: Although this password makes use of upper- and lowercase letters and numbers, it is much too short, contains a common dictionary word with a common substitution (o = 0), and has what is likely a "year" value appended to it (birthday, anniversary, favorite Windows OS, etc). You should always use better passwords when using real devices.

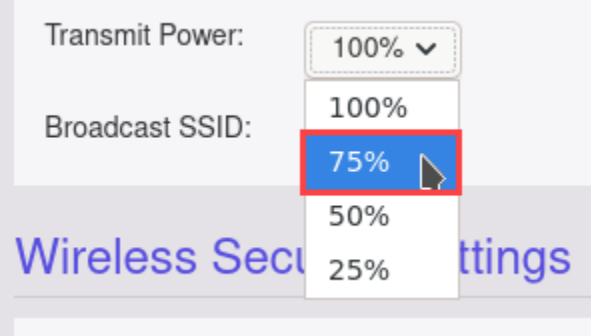
Before locking in your new security mode, there is one more change that may be worth making. The network administrator has ap1 perched at the edge of the property, right near the road, and is transmitting its signal on full blast. If instead they relocated that AP to the middle of the property, you

could probably tweak the transmit power so that very little signal escapes the property border. As a wardriver, you know that you could still locate this AP with a high-gain antenna. However, this would make the network much harder to piggyback on. Wi-Fi communication is bidirectional, after all – if the AP cannot communicate with the wireless station, it doesn't matter that the piggybacker's high-gain antenna has range of 3 kilometers.

Really, the primary piggyback deterrent and true security MVP here is the WPA2 protection. Most potential attackers will take off once they discover this, because it is not worth the time, and a breach is not guaranteed. However, it is always important to have multiple layers of security. For a WPA2-Personal home network, implementing some involved or bleeding-edge security solution is probably not worth the trouble, but tweaking your transmission power is no power-user move, and also a pretty neighborly thing to do. Most access points will transmit at full power (100mW or 20dBm), but there's no point to blasting wireless beacons into your neighbor's living room. You should confine those broadcasts to your own space and cut your noise pollution!

In the next steps, you will turn the transmit power down to 75%, apply your changes, and then move the ap1 access point to the center of the property to find out if your calculations were correct.

9. In the Wireless Network Settings section, **select 75%** from the **Transmit Power menu**.



Transmit Power

10. At the bottom of the page, **click** the **Apply Changes button** to reload the access point configuration with your new specifications.

    After a few moments you will be redirected to the Status page. Here you can confirm your values were successfully applied.

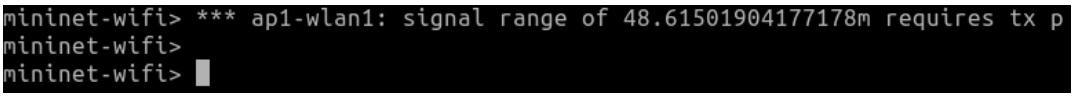
Apply Changes

11. **Make a screen capture** showing the **new security mode and transmit power values for *simplewifi* on the GHostAPd Status page**.

Note: In the next steps, you will simulate moving the access point. Hopefully these changes will keep the broadcast range mostly, if not entirely, off the street.

12. **Restore** the **wifi@TargetLinux01:~/Topos terminal window**.

    If you see output in the console (specifically , "...signal range of…"), **press Enter** to clear it and restore the command prompt.



Restore the command prompt

13. At the command prompt, **type** `py ap1.setPosition('100,50,0')` and **press Enter** to place the access point in the middle of the property.

14. **Restore** the **Mininet-WiFi Graph window**.

Note: If you successfully relocated the access point and adjusted its tx (transmission) power to 75%, you should find that the signal range now fits snugly within the confines of the owner's property.

You may notice that you could recommend similar changes to the owner of ap2, whose current set-up is not very neighborly right now, but one problem at a time.

15. **Make a screen capture** showing the **relocated ap1 with 75% transmission power in the Mininet-WiFi Graph**.

16. **Restore** the **Node: sta1 terminal window**.

17. At the command prompt, **type** `linssid` and **press Enter** to launch the LinSSID wireless scanner.

18. In the LinSSID window, **click** the **Run button** to begin scanning for wireless networks.

Note: After a few moments you should see both wireless networks populate in the LinSSID display. More importantly, you should notice that *simplewifi* is now reported as using PSK for Security and CCMP for Privacy and Cipher.

19. **Make a screen capture** showing **both WPA2-PSK networks in LinSSID**.

Note: This concludes Section 1 of the lab.

Section 2: Applied Learning

Note: SECTION 2 of this lab allows you to apply what you learned in **SECTION 1** with less guidance and different deliverables, as well as some expanded tasks and alternative methods. You will learn more about the hidden wireless security control, as well as the tools that can be used to defeat it.

1. If you completed Section 1 of this lab, you will need to reset the virtual environment before beginning Section 2.

   To reset the virtual environment, complete one of the following options.
   a. **Click Options > Reset Lab** to restore all virtual machines to their base state. This will take several minutes to complete. If you do not see the desktop after five minutes, **click Options > Reload Lab** to reload your lab connection.

   b. **Click Disconnect**, then **select Discard Changes** to end your lab session without creating a StateSave. If you previously created a StateSave, delete the StateSave at the launch page, then start a new lab session.

2. **Proceed** with **Part 1**.

Part 1: Perform a Wardriving Attack with Kismet

Note: Access points advertise their WLANs by broadcasting beacons that are sent out for all who care to listen. At one point, blocking the SSID broadcast and instead pre-configuring authorized Wi-Fi devices with a profile for the SSID was considered to be a good security control. Today, this approach is no longer considered a viable security control, due to the fact that when a Wi-Fi station associates with an access point, it must still broadcast a network probe. If an eavesdropper were to be monitoring the RF range at the time, they would be able to identify the hidden SSIDs during the association process.

In this part of the lab, you will learn about the tools and techniques that wardrivers use to detect hidden SSIDs. You will do so using one of the oldest and most popular tools in wireless security – Kismet. Kismet is practically synonymous with wardriving. Along with NetStumbler, it was one of the original wardriving tools of choice when Wi-Fi first started gaining traction in the early 2000’s, but Kismet has survived and is still in use today. Kismet has many advanced features that make it popular, but it can be more difficult to operate than tools like LinSSID. One key feature of Kismet is its ability to surface hidden SSIDs. Although hiding SSIDs by restricting broadcast transmissions is superficially a good security control, it does not stand up to competent attacks performed using tools like Kismet.

In the next steps, you will instantiate another emulated wireless environment using Mininet-WiFi. You will assume the role of a wardriving station in motion. You will be tasked with getting Kismet running so that you can begin compiling a list of all networks in the area. Upon discovery of a hidden network, you will attempt to decloak the network and verify its identity.

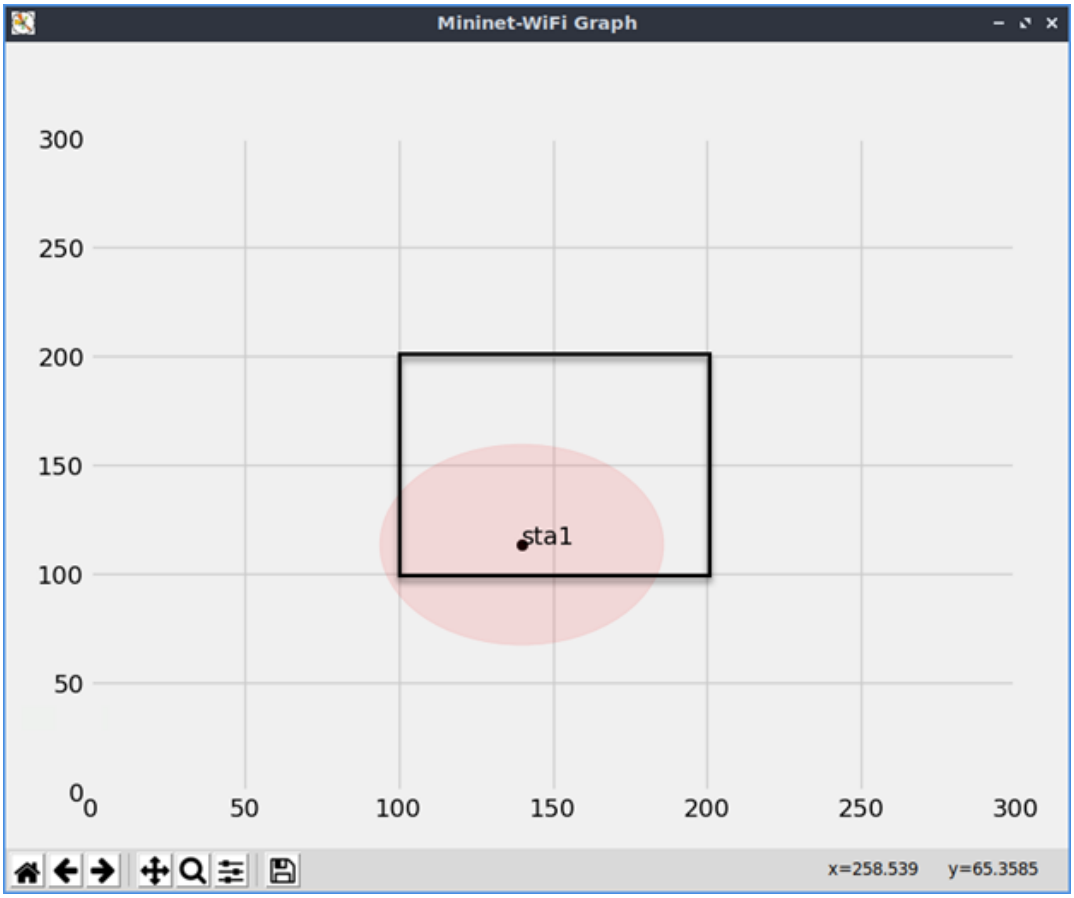
1. From the TargetLinux01 applications menu, **launch** a new **QTerminal session**.

2. At the command prompt, **execute** `cd Topos` to change your working directory to Topos.

3. From the Terminal window, **execute** `sudo python wardriver-s2.py` to build and bring online a new emulated wireless topology.

   When prompted for a password, **use** `wifi`.

Note: Right now, sta1 looks like someone who is lost but reluctant to stray too far. The Mininet-WiFi Graph shows your wardriving avatar randomly scurrying about this x/y neighborhood. It is, in fact, moving about pseudo-randomly, and within the confines of a smaller subspace on this graph (100 < x < 200, 100 < y < 200). That subspace is a square that looks like this:



Mininet graph

4. **Minimize** the **Mininet graph window**.

5. At the command prompt, **execute** `py sta1.setAntennaGain(9, intf='sta1-wlan0')` to switch your wardriving weapon to a 9 dBi omnidirectional antenna.

6. **Execute** `xterm sta1` to open a console on your wardriving machine.

7. **Maximize** the **Node: sta1 window**.

Note: You will launch Kismet in the next step. If you do not maximize the terminal window now, it will not respond to any window resizing during runtime. That being said, this is optional.

8. At the command prompt, **execute** `kismet` to launch the Kismet wireless device detector.

Kismet

Note: Kismet has already been configured to use the sta1-wlan0 interface, according to a specification in the /etc/kismet/kismet.conf file, which is why you did not need to specify it here. There are many configurables in this file that you can use to alter Kismet's behavior, such as a wireless mode inclusion/preclusion list and parameters for GPS integration.

9. At the Kismet startup screen, **press Enter** to select the Start option and run a wireless scan using Kismet.

Note: The following steps use the keyboard to navigate. However, you may also click the various

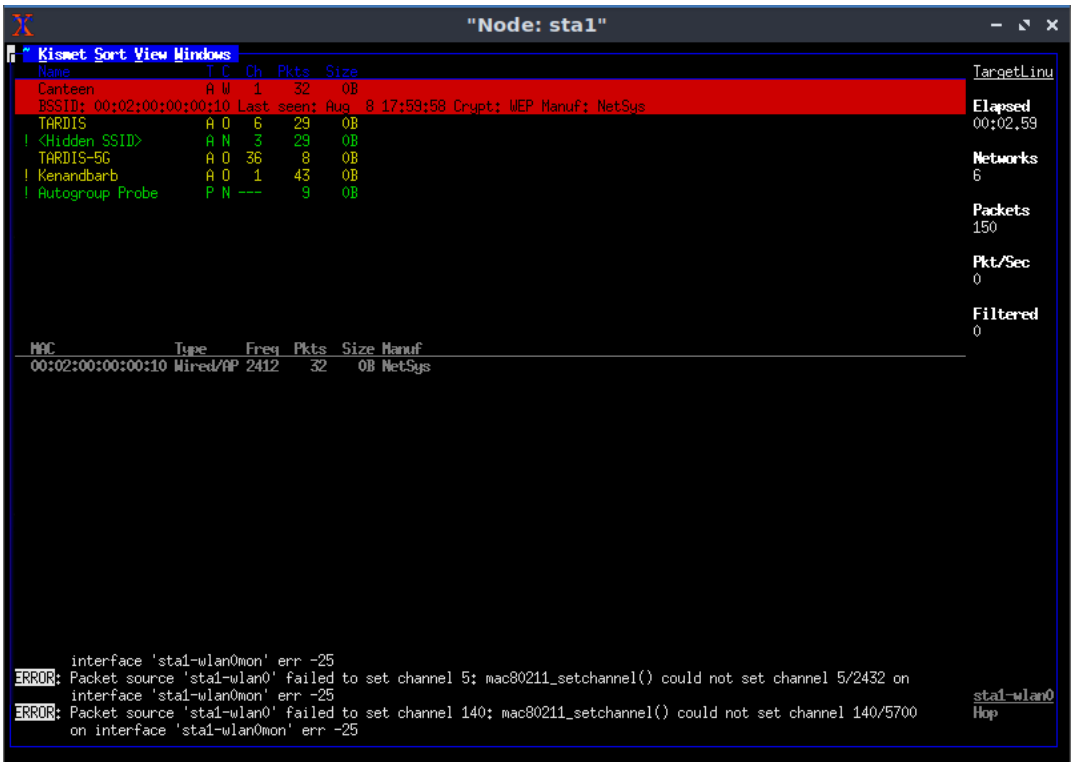
items in the Kismet GUI if you prefer, as you would expect to do in any typical GUI.

If presented with a dialog about potentially interfering processes, **press Enter** to acknowledge and dismiss it.

10. **Use the Tab key** to select the Close Console Window option, then **press Enter**.

    You should see a list of five networks discovered by Kismet, one of them with a hidden SSSID. This may take anywhere from 30 seconds to ~2 minutes to populate, so do not worry if you they fail to appear immediately.

    Once your Kismet view is populated with all five networks, proceed to the next step. The order of networks in your lab may not match the screenshot below.



Fully populated Kismet view

Note: You should see pretty varied array of networks this time around! And Kismet is kind enough to color code these for you according to their security mode: red for WEP, yellow for WPA/WPA2, and green for no security.

You should see two TARDIS networks: one broadcasting a 2.4GHz network (*TARDIS* on channel 6), the other broadcasting a 5 GHz network (*TARDIS-5G* on channel 36). Note that 5 GHz networks provide faster speeds than do 2.4 GHz, but cover a smaller range, and their higher-frequency waves are not nearly as effective at passing through obstacles. In any case, both flavors of this network are locked down with WPA2-PSK, so there is not much point in looking further. You can confirm this yourself by using the arrow keys to highlight one of the TARDIS networks and looking for “Crypt: WPA PSK AESCCM” in the expanded information. You also have *Kenandbarb*, but that, too, is locked down with WPA2.

Then there’s *Canteen* on channel 1, which is using WEP (Wired Equivalent Privacy), a weak encryption mode that provides a *slightly* better security posture than a wide-open access point. WEP is riddled with security flaws due to its poor implementation of the RC4 cipher, and the pre-shared key on networks using it can be cracked within minutes with modern techniques.

And finally, you have an entirely open network, a perfect find – save for the missing SSID. It seems the admin opted to keep their network completely unprotected, but nevertheless went through the trouble of hiding their SSID (removing it from the AP’s wireless broadcasts so that the name is no longer advertised). If they think this is an acceptable security measure, they are dead wrong. There are numerous ways of obtaining the SSID for a network, so a hidden SSID is a trivial obstacle. That is not to say it should not be done, but rather that it should not be relied upon as a singular security solution. In the next steps, you will attempt to learn the SSID for the mysterious network.

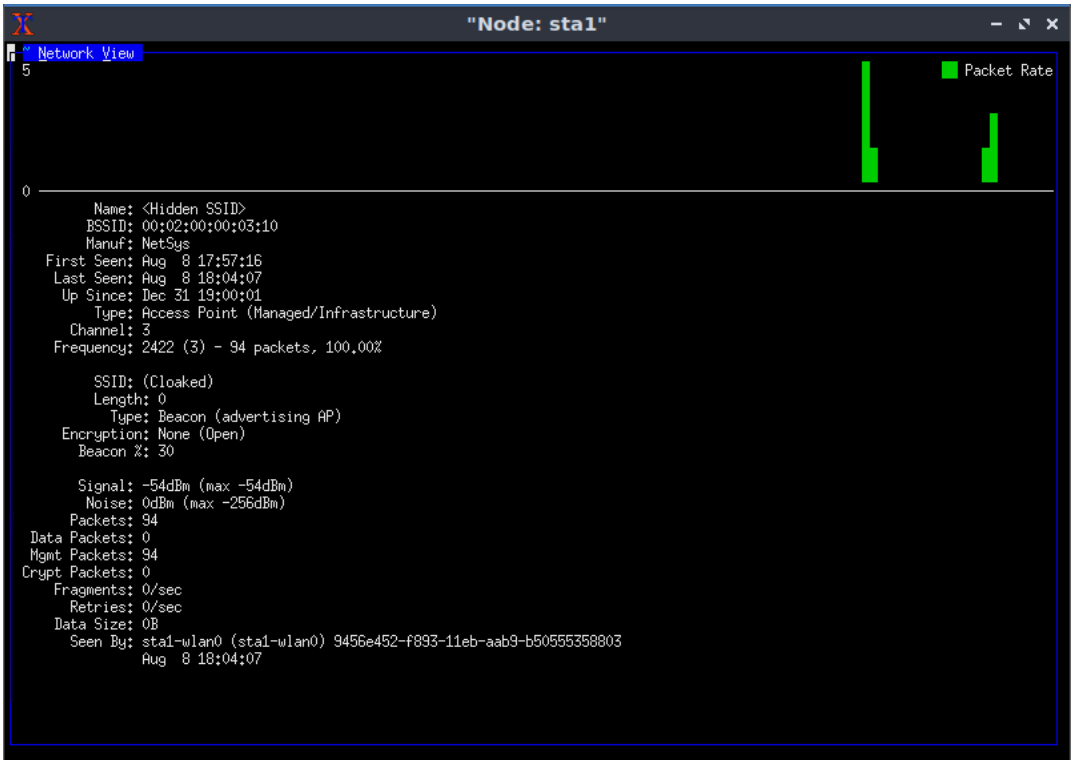
11. **Make a screen capture** showing the **list of five networks detected by Kismet**.

12. If necessary, **use the Arrow keys** to highlight the <Hidden SSID> network in Kismet’s network view.

Note: The alphanumeric values in the green row provide some basic information, such as the type (T) of device (A for access point), the cryptographic/cipher (C) implementation (N for no encryption), as well as the BSSID (MAC address) and a more detailed cryptographic breakdown (None in this case).

Additional information can be discovered by drilling down into a specific network, as you will do now.

13. **Press Enter** to open a new window displaying additional information about the hidden access point.



Additional information for the hidden SSID

Note: It appears that there is nothing too helpful here. *SSID: Cloaked* tells you what you already know – or rather, reminds you of what you do not know – which is that the SSID name remains a mystery. Time to do some de-cloaking.

14. **Press Esc** to highlight the Network menu in the top left of the Kismet window, then **press Enter** to expand the menu.

15. **Press w** to close the window and return to the main Kismet view.

16. **Use** the **Arrow keys** to navigate to the *Autogroup Probe* network row.

17. **Press Enter** to open a new Kismet window with additional information on this network group.

Note: This network group displays unanswered probe requests. Probe requests are how wireless clients scan the network for available WLANs. They are also used to contact hidden networks the client wishes to connect to, which means that in these cases, they are continually revealing the "secret" SSID in their probes.

You should find that one of these probes is for a network called *youcantseeme* (SSID: youcantseeme). There is only one hidden network in your list, and none of them have this name, so it would appear you have found your hidden network. Admittedly, so this is a little cheap, because the probe request was sent from your own wardriving machine (sta1 at 00:02:00:00:00:10), in an attempt to connect as it walked around collecting beacons – and it should not already know the name of this SSID. Additionally, because this was an *unanswered broadcast* probe, Kismet is unable to connect it to any specific AP/BSSID – only you can, due to your admirable deductive prowess.

In some cases, you may even get an answer to your probe request, and therefore already have the <Hidden SSID> decloaked in your Kismet view. In this case, congratulations, you are able to confirm your hypothesis early, and the remaining steps in this part can mostly be considered a victory lap.

While sta1 continues roaming the square collecting wireless networks, you are going to take over another wardriving (well, piggybacking) machine. From here you will attempt to connect to the hidden network using the SSID discovered in the probe request, thereby confirming its identity. To do this, you will need to bring sta2 into the picture.

You will take control of sta2 by first repositioning it near the *youcantseeme* AP and then connecting via another xterminal. Then, it will be time to see if you *can* see them.

18. **Make a screen capture** showing the **SSID: youcantseeme probe request in your Node: sta1 Kismet window**.

19. **Execute** the **ctrl-c hotkey combination** to end your Kismet session.

20. **Close** the **Node: sta1 terminal window**.

21. If necessary, **restore** the **wifi@TargetLinux01:~/Topos terminal window**.

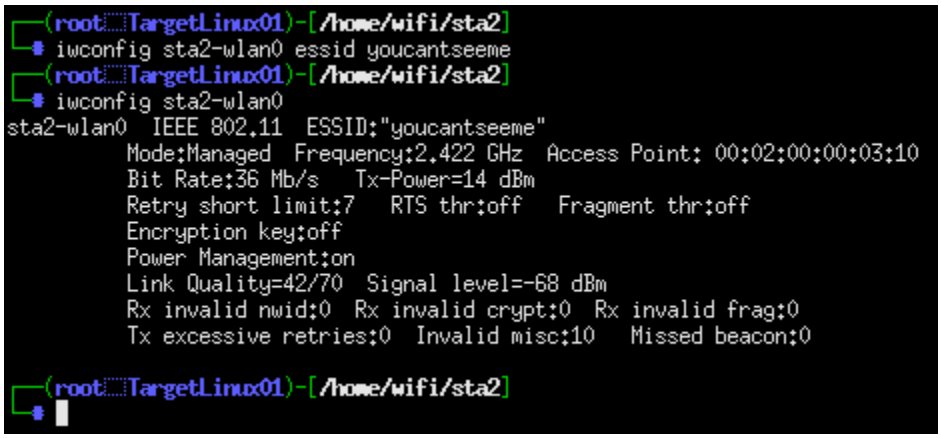
22. At the command prompt, **execute** `py sta2.setPosition('190,90,0')` to relocate sta2 from off-screen to a position near the hidden access point.

23. **Execute** `xterm sta2` to open a terminal for sta2.

24. In the Node: sta2 terminal window, **execute** **`iwconfig sta2-wlan0 essid youcantseeme`** to attempt a connection to the hidden network.

25. In the Node: sta2 terminal window, **execute** **`iwconfig sta2-wlan0`** to check the wireless interface configuration to determine if your connection was successful.

    If successful, you should see **ESSID: "youcantseeme"** in the first row of the output. Association may take several seconds, so you may need to reissue the command again to retrieve the updated configuration.



Review the wireless interface configuration

Note: In the next steps, you will set up a dedicated monitoring interface (to avoid tying up your primary one) and run Kismet on it. Assuming that *youcantseeme* and this hidden network are one and the same, Kismet should report this network as de-cloaked.

26. In the Node: sta2 terminal, **execute** **`iw sta2-wlan0 interface add mon0 type monitor`** to create a new virtual interface (**`interface add mon0`**) on top of your existing wireless interface (**`sta2-wlan0`**) and put this new interface in monitoring mode.

Note: Normally your wireless interface will only see traffic destined for itself, dropping the rest. In monitor mode, your interface will now pay attention to all packets, no matter the destination, and without even needing to be associated with the network!

Kismet had already configured a monitoring interface in your previous run on sta1, but it also took your

primary interface offline to prevent interference. In this case, you are opting to create the virtual device beforehand, so that your primary interface (sta2-wlan0) is not interfered with.

27. In the Node: sta2 terminal, **execute** `ip link set mon0 up` to bring your new monitoring interface online.

28. **Execute** `kismet -c mon0` to launch Kismet using your new virtual monitoring interface.

    Remember to maximize the terminal window! You may also notice some “Packet source ‘mon0’ failed to set channel” messages piling in immediately. These are artifacts of the environment that can be safely ignored.

29. In Kismet, **start** the **capture session**, then **close** the **Console Window**.

    The hidden network should already be highlighted. If it is not, use the arrow keys to select it.

Note: It will take up to a minute for the SSID to update, but after a short time you should see the SSID switch from <Hidden SSID > to <youcantseeme>. Notice the use of brackets to indicate this is a de-cloak.

The lower half of the screen displays all clients connected to the network currently selected. In this case, you can see the BSSID of the AP itself, with a Type of Wired/AP; and you can see your own MAC address, because you are currently a client of this network.

30. **Press Enter** to view additional information about this access point.

Note: Look back to that *SSID: (Cloaked)* row you identified previously – top row of the second block. You should notice a line below it reading *Probable Decloak: youcantseeme*. This is Kismet’s way of saying “I think I *can* see you.”

Looks like a win! Although a little sleuthing was needed to be done to determine the SSID in this case, it should be clear that it is not a good defensive device on its own. The airwaves are usually ripe with traffic containing identifying information about nearby networks. Probe request are one example, but another popular target are authentication exchanges: because the network SSID is required from the

client to connect to the network, it can often be extracted from these exchanges.

31. **Make a screen capture** showing the **<youcantseeme> network details in your Node: sta2 Kismet window**.

32. **Press ctrl-c** to end your Kismet session.

33. **Close** the **Node: sta2 terminal window**.

Note: In the next steps, you will shut down Mininet and perform a clean-up.

34. If necessary, **restore** the **wifi@TargetLinux:~/Topos terminal window**.

35. At the command prompt, **execute** `exit` to shut down the current emulated wireless topology.

36. At the command prompt, **execute** `sudo mn -c` to perform a thorough cleanup of all residual configuration data for previously running topologies.

    If prompted for a password, **type** `wifi` and **press Enter**.

Part 2: Harden the Access Point

Note: In this part of the lab, you will harden another access point, but this time you will be pulling out all the stops – transmit power, hidden SSID, and even some MAC filtering. Although enabling encryption on your network device should be the first step you take towards improving its security posture (if it does not come with encryption preconfigured), there is value in paying attention to some of the smaller gains available. For example, reducing your transmit power: it is neighborly, and a good way to prevent attackers from whispering to your AP from outside the walls of your home. Such steps are often overlooked because they offer little gain in comparison to a full cryptographic overhaul, but their importance should not be underestimated. Particularly in small networks with one or two security controls (a firewall and a password, maybe), a tweak to transmit power and a hidden SSID may present *just* enough of an inconvenience to turn away some potential attackers. After all, even the most sophisticated cryptographic implementation (barring any quantum cryptography) is only considered secure because it would take an impractical amount of time to crack it. If you are not a

public figure or someone with widespread recognition, then simply being too inconvenient to be worth the attention or effort is often enough to remain untroubled and to preclude inclusion of your network on any wardriver's list.

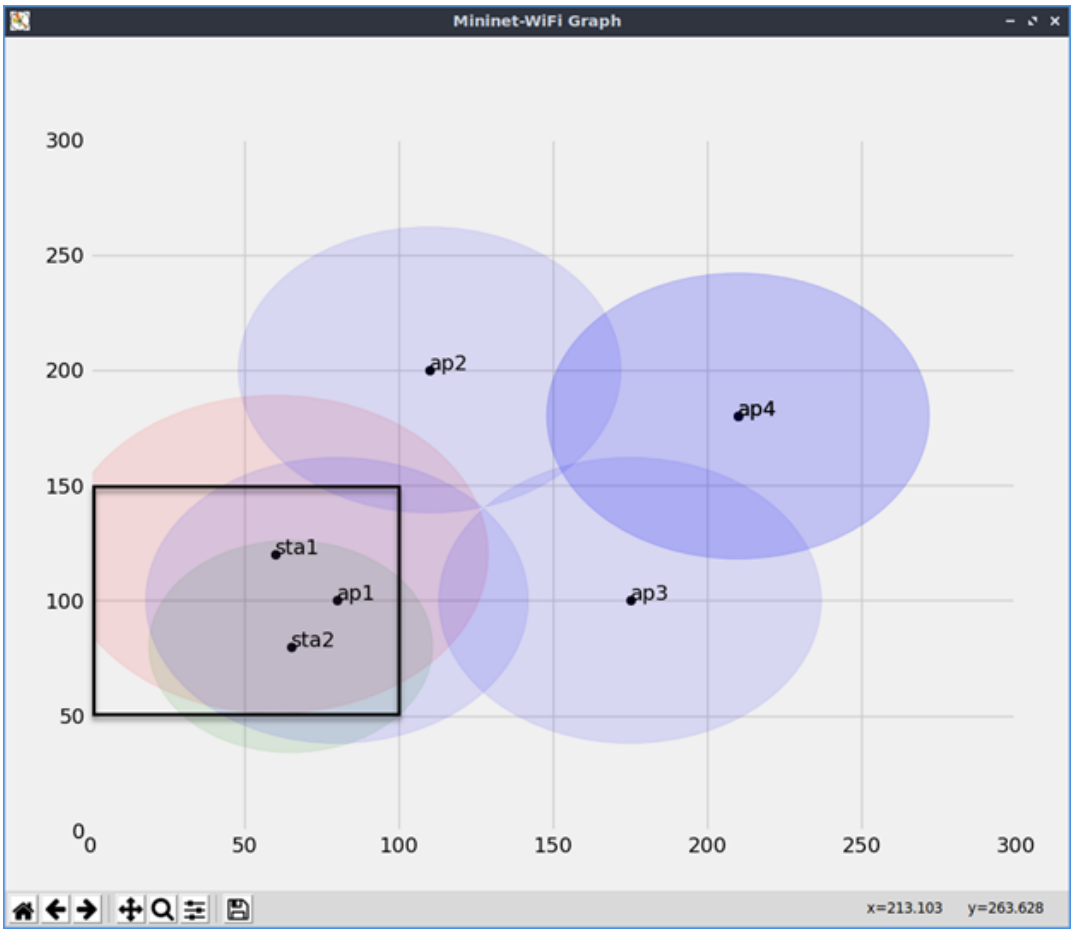
In the next steps, you will once again perform an act of goodwill, and double back to offer your security services to the administrator of one of those networks you peeked at in Part 1. You will access the wireless admin interface for a WEP-enabled device and apply several security controls, such as an improved encryption mode, and MAC filtering, and perform some tests to confirm they were configured properly.

1. At the command prompt, **execute** `sudo python wardriver-s2_static.py` to build another emulated wireless network topology.

Note: It may not seem familiar at first, given the number of blue circles on the screen, but this is that same block you were looking at in the previous part. You have ap1, or *Canteen*, with its WEP encryption; ap2, or as you know it, *Kenandbarb*, which is sealed with WPA2-PSK; the now-ironic network, *youcantseeme*, is at ap3; and the TARDIS/TARDIS-5G pair, each under the protection of WPA2-PSK, is represented by ap4.

As you may have gathered, based on where sta1 and sta2 are located, the target for hardening is ap1, or *Canteen*. They were apparently security conscious and attempted to apply encryption to their network, but unfortunately they selected WEP, a weak implementation, which means they are hardly safer than the hidden-but-wide-open *youcantseeme* network. In this case, both sta1 and sta2 are devices under your control.

You are going to make some important changes to harden this network. First, get some better encryption – WPA2-PSK sounds like a good choice. Second, hiding the network's SSID was a neat trick, so you will do the same for a slight boost in AP obscurity. Third, put that AP in the middle of the property and turn down the transmit power so that the signal is unlikely to reach outside of the property borders.



Mininet graph

2. From the TargetLinux01 taskbar, **launch** the **Firefox web browser**.

3. Using Firefox, **navigate** to **`http://10.0.0.254`**.

4. From the GHostAPd menu, **navigate** to the **Wireless Network and Security Settings page.**

5. On the Wireless Network and Security Settings page, **configure and apply the following changes** to the access point:

   Transmit Power: 75%
   Broadcast SSID: unchecked

Security Mode: WPA2-PSK
Passphrase: strongpassword

Note: Once again, remember that the supposedly strong password used in this lab is nothing of the sort.

After applying your changes, you should be looking at the Status page. That page should reflect all the changes you have just made to the access point. Transmit power and Security mode should report 75% and WPA2, respectively, and Broadcast should be reported as Off.

6. **Make a screen capture** showing the **WPA2-enabled *Canteen* network as reported by the Status page**.

7. **Restore** the **wifi@TargetLinux01:~/Topos terminal window.**

   If necessary, **press Enter** to clear the “ap1-wlan1: signal range…” output and reclaim the mininet-wifi> command prompt.

8. At the mininet-wifi> command prompt, **execute** `py ap1.setPosition('50,100,0')` to relocate the AP to the specified coordinates.

8. **Restore** the **Mininet Graph window**.

9. **Make a screen capture** showing **ap1’s new location and signal range in the Mininet-Wifi Graph window**.

Note: Hold on now – there also seems to be a MAC filtering option on this AP. That should help improve this network’s security posture by selectively allowing or denying hosts based on the MAC address of their wireless network interface card (wNIC). Why not use it to add another layer of inconvenience between strangers and your network?

Similar to hiding your SSID, MAC filtering is a relatively trivial security approach, and is easily overcome by MAC address spoofing, in which an attacker assigns a different MAC address to their card than the one issued by the card manufacturer. If an attacker knows a valid MAC for that network, they need only define that address for their card to get access. They may also take other measures to take full control of that identity, such as de-authenticating the legitimate client that holds that MAC address or forging some traffic control packets to silence its radio – whatever might prevent it from

getting back on the network and calling out the imposter.

In the next steps, you will implement MAC filtering through an access control list (ACL). You will then test this implementation to verify that it is working as expected.

10. **Restore** the **Firefox browser**.

11. From the GHostAPd menu, **click MAC Filtering** to navigate to the Access Control Lists page.

Note: This area also includes the same Attached Devices section as seen on the Status page. As expected, there are currently no attached devices, because a new encryption mode and PSK has recently been applied, kicking off any devices attached via the previous WEP configuration. After enabling an ACL here, you will connect some devices to the network to test your configuration, at which point some attached devices will appear in this table.

12. **Click** the **Access Control checkbox** to expand all ACL options.

    As noted in the dialog box, ACL has not been enabled until you apply your changes. You will do that after selecting your rule.

13. **Select Default Deny** from the Filter Rule menu.

Note: This selection should change the table below to the Allow List. MAC filtering works by either allowing or denying hosts based on their MAC addresses and the default rule set on the filter. If the rule is Default Deny, then only hosts in the allow list are permitted to join the network. If the rule is Default Allow, everyone except hosts in the deny list are permitted to join the network. As an addition to a SOHO (small office/home office) network, it can provide another layer of inconvenience armor that makes breaching your network more trouble for an attacker than its prospective worth. However, that inconvenience applies to you as well, who will now be required to maintain it. MAC filtering is not very scalable, due to the associated administrative burden of managing larger lists as the network grows, which makes it ill-suited to any sizable enterprise networks.

Here you are taking the typical, best-practice ACL route: Default Deny. It is much more efficient and human-error-resistant to block all devices by default, and then make exceptions on a case-by-case basis. By explicitly allowing each device that requests access to your network, you can keep close track of and exercise a tighter control over each device on your network, meaning you are less likely to

inadvertently allow any bad actors access to the network.

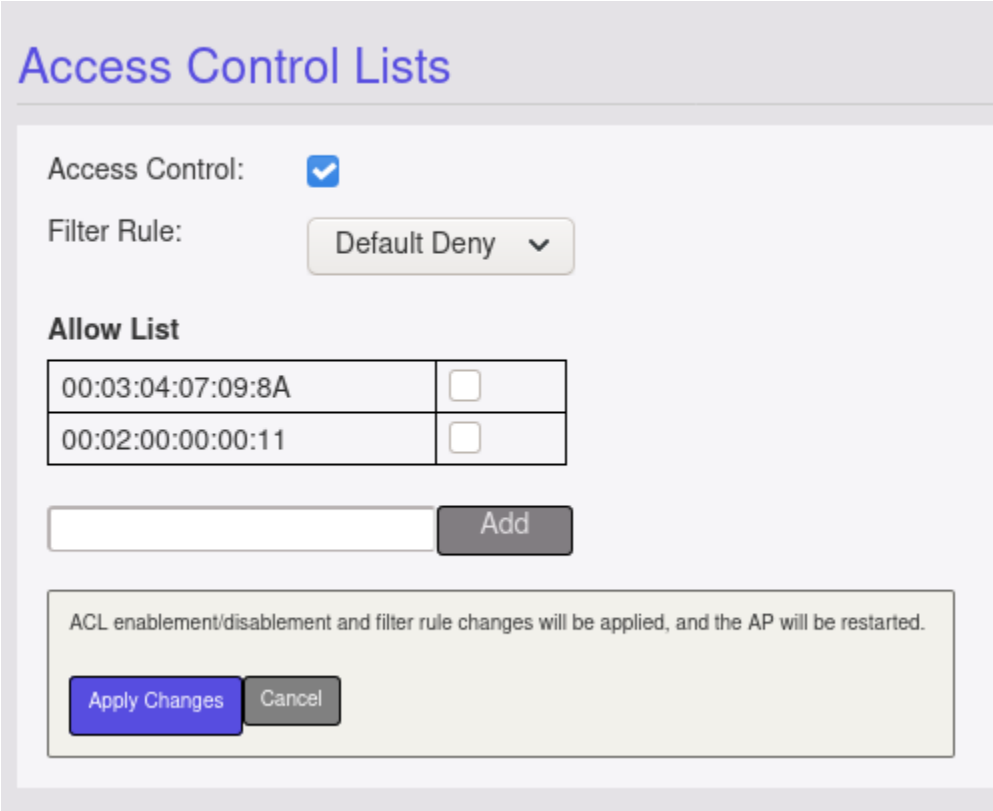
14. In the Add input field, **type** `00:02:00:00:00:11` and **click Add** to add your own (sta1's) MAC address to the Allow List.

Note: You will be deposited back on the Access Control Lists page. The Access Control box will be unchecked because you did not actually apply your changes – you merely added a rule to a list. To prevent being locked out of your device, it is wise to first configure the control list and confirm your entries before actually bringing the mechanism online.

In this case, as you may have noticed, the GHostAPd interface is accessible by you regardless of sta1's proximity to the access point, so you are not at risk of locking yourself out of the UI in this lab. Additionally, it is not protected by a login, which is another vulnerability that you would not (or should not) see in the real world. This independent access is purely a result of the emulated environment – in a real-world scenario, such access controls could naturally affect your ability to continue accessing the device's GUI, so due consideration should be exercised before deploying any restrictive filtering patterns.

15. On the Access Control Lists page, **expand** the **ACL options**, then **select** the **Default Deny** rule to display the Allow List.

    You should confirm that sta1's MAC address, 00:02:00:00:00:11, is now present in the Allow List. The other MAC address in the Allow List is a dummy and may be ignored for the purposes of this lab.



Access Control Lists

Note: If you accidentally added the wrong MAC address, or maybe the right MAC address to the wrong list, you can delete your entry and try again, using the following steps.

In the appropriate list, **select** the **checkbox** to the right of the address. This will change the Add button to a red Del button. **Click Del** to remove the address. You may then proceed again from step 12 to repeat the MAC address entry process.

16. **Click Apply Changes** to enable ACL and restart the AP.

    This time you are locking in your changes – ACL enabled with a Default Deny filter rule, using your curated allow list.

17. **Make a screen capture** showing the **new configuration values for the Canteen network on the Status page**

.

Note: You are almost there, now you just need to test your new implementation. You have added sta1 as an allowed host, but you have not added your other device, sta2. If both try to connect, and only sta1 is successful, that would be pretty strong evidence for a functional implementation (because sta1 was added to the allow list).

In the next steps, you will attempt to connect to the WPA2-enabled *Canteen* network from both sta1 and sta2. After your attempt, you will monitor the Attached Devices section in the GHostAPd to determine which station, if either, is successful.

18. **Restore** the **wifi@TargetLinux01:~/Topos** terminal window.

    If necessary, **press Enter** to clear the dialog.

19. At the mininet-wifi> command prompt, **execute** `xterm sta2` to open a terminal to sta2.

20. In the Node: sta2 terminal, **execute** `wpa_passphrase Canteen > /root/wpa.conf` to configure a wpa passphrase for the simplewifi network and store this configuration in the /root/wpa.conf file.

    You will be expected to enter a passphrase at the blank line.

21. **Type** `strongpassword` and **press Enter** to specify the passphrase for the WPA2-enabled *Canteen* network.

Note: Wpa_passphrase takes your passphrase, converts it to a PSK, and then stores it along with the SSID in wpa.conf in the expected format.

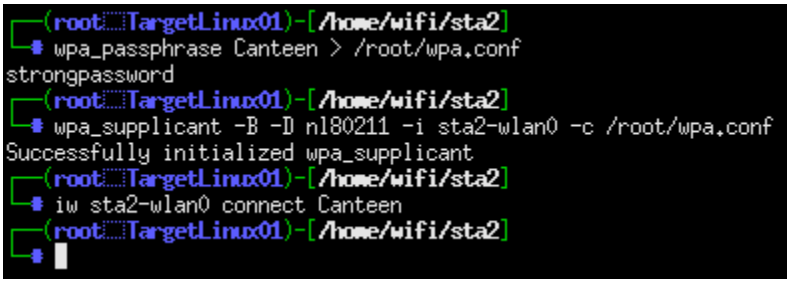
In the next steps, you will use wpa_supplicant, a well-known software wireless authentication client. Wpa_supplicant will make login requests to WPA networks on your behalf. It will also actively maintain your connection in the background and may automatically reassociate after a disconnection.

22. In the Node: sta2 terminal, **execute** `wpa_supplicant -B -D nl80211 -i sta2-wlan0 -c /root/wpa.conf` to start the 802.11 wireless supplicant.

Note: This command specifies to initialize wpa_supplicant as a daemon (**-B**, run it in the background), using netlink 802.11 drivers (**-D**), sta2's primary wireless interface (**sta2-wlan**0), and the WPA network config file you populated in the previous step.

Once you see "Successfully initialized wpa_supplicant" and the command prompt returns, you may proceed to the next step.

23. **Execute** `iw sta2-wlan0 connect Canteen` to attempt a manual connection to the network.



Connect to the Canteen network

24. **Minimize** the **Node: sta2 terminal window** and **restore** the **wifi@TargetLinux01:~/Topos** terminal.

25. At the mininet-wifi> command prompt, **execute** `xterm sta1` to obtain a terminal for sta1.

26. **Repeat steps 20-23,** replacing all instances of sta2 with sta1, to attempt a connection to the *Canteen* network from the sta1 client.

27. **Minimize** the **Node: sta1 terminal window** and **restore** the **Firefox web browser**.

28. From the GHostAPd Status page, **navigate** to the **MAC Filtering page**.

Note: No surprise, you should find sta1 connected just fine, but no sign of sta2. It would appear to be working!

If you do not see any attached devices, confirm the correct SSID and passphrase were used in the previous steps, and that you have allowed the correct MAC address (00:02:00:00:00:11).

29. **Make a screen capture** showing the **configured ACL parameters, and any Attached Devices, as shown in the MAC Filtering page**.

Note: In the next steps, you will perform one more test before concluding this section of the lab. You will need to confirm that sta2 did not just malfunction, and that it was indeed the MAC filtering that kept them off the network. To run this test, you can simply delete sta1's MAC address from the allow list and see if they end up getting disconnected from the network.

30. **Click** the **checkbox** beside the 00:02:00:00:00:11 MAC address in the Allow List.

    The gray Add button will now be replaced by a red Del button.

31. **Click** the **Del button** to delete sta1's MAC address from the Allow List.

Note: Sta1 should no longer be present in the Attached Devices table, confirming that your MAC filtering is working correctly. Overall, this is a massive improvement over the weak WEP solution that was previously in place. In the next steps, you will check the access point's log files to confirm that wpa_supplicant is busily attempting to reassociate sta1 with the network.

32. From the GHostAPd menu, **navigate** to the **Log page**, then **scroll down** and **locate** the line ***sta1-wlan0: 00:02:00:00:00:10 denied authentication (status 1)***.

    You should find this line immediately following the authentication attempt: "sta1-wlan0: send auth to 00:02:00:00:00:10"

33. **Make a screen capture** showing **sta1-wlan0 interface being denied authentication in the GHostAPd Log**.

Note: This concludes Section 2 of the lab.

Section 3: Challenge and Analysis

Note: The following exercises are provided to allow independent, unguided work using the skills you learned earlier in this lab - similar to what you would encounter in a real-world situation.

Before beginning Section 3, you will need to complete the following steps.

1. If you completed Section 2 of this lab, you will need to reset the virtual environment before beginning Section 3.

   To reset the virtual environment, complete one of the following options.
   a. **Click Options > Reset Lab** to restore all virtual machines to their base state. This will take several minutes to complete. If you do not see the desktop after five minutes, **click Options > Reload Lab** to reload your lab connection.

   b. **Click Disconnect**, then **select Discard Changes** to end your lab session without creating a StateSave. If you previously created a StateSave, delete the StateSave at the launch page, then start a new lab session.

2. From the TargetLinux01 applications menu, **launch** a **QTerminal window**.

3. **Execute** `cd Topos` to change to the Topos directory.

4. **Execute** `sudo python wardriver-s3.py` to bring this section's emulated wireless topology online.

   If prompted for a password, **enter** `wifi`.

Part 1: Perform Data Gathering and Analysis with Wardriving

You have recently been hired by Corporation Techs, a company that specializes in IT outsourcing. The company has asked you to analyze a small remote professional office located in the outskirts of town near a new residential development. Some employees have noted strange networks popping in and out of existence in the common area between their building plaza and a long stretch of chic townhouses and coffee shops. Given your experience as a wardriver, you figure you can gather some valuable data by just walking around and getting a lay of the land, and then making some comparisons later to see if you can obtain proof of the employees' claims. You also realize that the same unencrypted networks you seek through wardriving are also the most dangerous ones for passers-by.

Like little digital whirlpools, they take advantage of the fact that most clients are running daemons (like wpa_supplicant) that automatically connect to networks the client is authenticated for. Given that an open network performs null authentication, devices will often connect to these networks without prompt. If this open network were not a hotspot gifted by your ISP, and instead a rogue access point with the intention of siphoning your data, you may just have spilled some tea into a dangerous little data sink.

Alright, time to look for danger. You have come equipped with a 9dBi antenna, and plan to set up a separate monitoring interface, mon0, to start Kismet on. After that, you will simply move around using a small program that accepts x,y coordinates. Whether this moving around is warjogging, warcycling, wardriving, or unashamed warteleporting, is up to you. Your aim is move to several places on the map to gather all access points in the area within your Kismet view.

Once you have collected and enumerated the security mode of each access point in the area, you will take notes and make a screen capture so you have a reference for later captures in this area. And make sure to keep your Mininet-WiFi Graph open (it can be minimized, just not closed) as you work your way through the challenge so that you may consult it periodically as you move around the map. Now get to it!

1. First, **open a terminal** for the **sta1 wireless station**. Next, **create** a new virtual monitoring interface called **mon0** from your primary **sta1-wlan0** interface, then bring your new monitoring interface online.

2. **Maximize** your **Node: sta1 terminal window**, then **run Kismet** on the mon0 interface using the **`-c`** option (**`kismet -c <virtInterface>`**). **Start Kismet** and **close** the **Kismet console window**.

3. **Restore** the **wifi@TargetLinux01:~/Topos window** and **execute** **`sh /move.sh`** to start a program that will make position changes easier.

   This program lets you change sta1's position by entering x,y coordinates (for example, 250,50). The idea is to first inspect the Mininet-WiFi Graph to determine where you want to move, then restore the -> prompt and enter that location's corresponding coordinates. You can either approximate the location using the axis values (300x300 grid, in units of 50), or you can determine a location more precisely by hovering over a spot in the Mininet-WiFi Graph and inspecting the x and y coordinates at the bottom right of the window.

   Use this program to move around the map, in between inspections of Kismet for new networks. And make sure to give Kismet enough time in each position (10-20 seconds) to register all broadcasting networks in the area.

Once you have collected all four networks in your Kismet view, move to the next step.

4. **Make a screen capture** showing the **four discovered networks in Kismet**.

5. **Document** the SSID of the open hidden network you discovered.

6. **Document** the SSID of the WEP-encrypted network you discovered.

Part 2: Investigate Potential Rogue Access Points

There is a total of four networks are in this area right now. Your wardriving reflex kicked in, so you recorded a WEP AP you found, but you are not too interested in that right now. What you *are* interested in is the hidden unencrypted access point you discovered. You have consulted with employees who confirm it is the only one they do not recognize as a regular find in their device's Wi-Fi scans.

You figure that if it is a rogue access point intending to siphon data, it is doing a poor job by attempting to hide itself. In that case, it would only be useful if it were using an SSID you already have saved – a technique you know as AP masquerading. In such cases, the attacker may attempt to intercept the credentials used by your wireless supplicant as it automatically attempts to associate you to this familiar network nearby.

All things considered, the name of this network suggests that the admin doesn't want anybody stopping by, so perhaps this is just another misguided person attempting to secure their network by merely "hiding" it from view. However, there is one interesting word in the SSID: GET. This capitalization may be a reference to the HTTP GET method, which browsers use to request the contents of a web page from a web server. The 8080 suffix may also be a reference to a common alternative port number used for web traffic. Hmm…, maybe this person is a little more tech-savvy than you expected, and is playing some kind of game with you. Perhaps this is the beginning of a recruitment process, whereby some super-secret government organization is attempting to locate some scrappy genius for a new cyber initiative. Maybe a ton of these things are planted throughout the world, each with a different SSID and a different clue, and they are the beginning of a gauntlet all prospects must pass through to get to an interview, and…. okay, you're getting ahead of yourself. But clearly you are going to follow this thread to its end.

You resolve to connect to the hidden network. Once there, you will use curl, a command-line data transfer utility, to perform a GET request on the network gateway, hopefully retrieving some secret messages that might be stored there. Minimize your Kismet window and get to it.

1. **Move sta1** to **location 225,75** on the Mininet-WiFi Graph to place it within range of the

<GEToutofmyyard8080>

2. **Execute** `q` or `exit` to terminate the move.sh program

3. At the mininet-wifi> prompt, **open** another **terminal** for the sta1 machine.

4. Use the **`iwconfig`** utility to connect to the open *GEToutofmyyard8080* network using sta1's primary interface (sta1-wlan0). The following syntax is expected:

   ```
   iwconfig <primaryInterface> essid <targetNetworkSSID>
   ```

5. **Execute** `iwconfig <primaryInterface>` to confirm you have connected successfully. It may take several seconds to complete.

   Alright, you have connected, and you have an IP address, as well as a default gateway. That default gateway has an IP of 10.50.0.254/24, and is presumably the address this stranger would have you send a GET request to (over port 8080, they seemed to specify). That is, if you are interpreting this all correctly… No point in turning back now — go GET some answers.

6. Use the **`curl`** command to issue a GET request (the default if no options are specified) to the gateway on the port 8080.The following syntax is expected:

   ```
   curl <ipaddress>:<port>
   ```

7. **Make a screen capture** showing the **response to your curl/GET request**.

Part 3: Connect to Hidden WPA2 Access Points

Well, what a ride – it looks like you have grabbed some coordinates for another access point, as well as an SSID – something about the gates of heck? What is this all about? And no kidding it is off the grid, the Mininet-WiFi Graph neighborhood is 300x300, and you are headed to x=450? This is starting to feel like some glitch in a video game. Perhaps this is how you get Mew.

Alright, it is getting late, so you best just head to those coordinates, confirm there's an AP there, connect to it to capture a de-cloak, and then get home for well-deserved rest. You can figure out what the heck this all means later.

The passphrase has already been entered into your wpa.conf file, so you can skip the wpa_passphrase step and go straight to associating with the AP.

1. At the mininet-wifi> prompt, **execute** `sh /move.sh` to start up the sta1 positioning program again.

2. **Move** to the **coordinates** specified in the response to your GET request.

3. Consult Kismet to **confirm** a new hidden network was discovered.

4. **Restore** the **second sta1 terminal** (the one not running Kismet).

5. **Use** the **`iw` command** to connect to the **TheGatesofHeck network** using your primary wireless interface, **sta1-wlan0**. The following syntax is expected:

   ```
   iw <primaryInterface> connect <targetNetworkSSID>
   ```

6. **Use** the **`iwconfig` command** to verify you are connected to the network.

7. **Restore** your Node: sta1 Kismet window to **confirm** the SSID has changed from <Hidden SSID> to <TheGatesofHeck>.

8. **Make a screen capture** showing the **decloaked *TheGatesofHeck* network in Kismet**.

Note: This concludes Section 3 of the lab.