Introduction

All wireless computers are both radio transmitters and receivers. Your phone, your computers, your smart watches – all of them compete for access over the airwaves to resources on your network and beyond. If you enjoy playing games or streaming video, you can probably appreciate the squeeze experienced when too many devices are connected to the network. No, it is not your Internet that sucks (but it might be that, too) – it is the fact that you have eight devices smashing electrical signals into each other in the same room, oscillating through the airspace in an attempt to enable fast and meaningful conversations between your devices.

Given the nature of the medium, it is not surprising to find that these airwaves are a hotbed for hacking activities. Blasting your personal information through the wireless surf-zone is going to perk some nearby ears (or antennae) – especially if that information is being conveyed in clear-text – so anywhere within range of your access point's signal is a great place to set up camp and begin hoovering up transient data. In fact, if you are connecting wirelessly, you should treat any information you transmit as though it is also being sent to some ill-intentioned party – like an email that always has badguy@illgetyou.com carbon-copied (CCd) on it. This does not mean that badguy@illgetyou.com gets all your meaningful information; it simply means that you can assume they will receive whatever you send. So, how about sending meaningless info? At least, meaningless to someone who does not have the proper code (or key) to unlock it.

The world's first Wireless Local Area Network (WLAN), ALOHAnet, went up in Hawaii in 1971. It was extremely cost-prohibitive, and thus used only when direct cabling was either extremely difficult or impossible. Fast-forward to 1997, after years of industry-specific solutions and protocols were developed in attempts to standardize this new implementation, and along comes IEEE 802.11, the Institute of Electrical and Electronics Engineer's (IEEE) official standard for WLANs. Fast-forward one more year, to 1998, and Ethernet's younger and hipper sibling arrives to the party, based on that fancy new 802.11 standard: Wi-Fi.

The 802.11 standard also came with the Wired Equivalent Privacy (WEP) security specifications, which aimed to provide wireless clients the same protections as those connected to the network over cables. Mind that not all wired connections, or hard-wired devices, are constrained to their tubing or casing – many emit sounds and electromagnetic radiation, which can be used to infer information about the operations taking place (known as side-channel attacks, many more of which are detailed in the National Security Agency's (NSA) TEMPEST specification, which specifies the performance and prevention of spying on systems through such information leakage). Such threats could be mitigated in degrees through things like fiber connections, cable shielding, and Faraday cages, but because the purpose and nature of 802.11 technology was to literally radiate information to nearby devices through the open air, throwing an access point into a Faraday cage would defeat its purpose. Instead, it must compete using some informational shielding: encryption. In any case, WEP, a name with big aspirations, came up quite short. It failed to make the airborne data meaningless enough to third parties, and soon wireless networks all over the world were vulnerable to any proximal nefarious actor with a little network know-how and the right software. WEP ultimately turned out to not provide equivalent, or even satisfactory, security – sure, it was better than nothing, but not by much. Engineers scrambled to apply a stopgap measure, eventually landing on WPA, which was a temporary implementation used to buy them time while they constructed the truly secure iteration: WPA2. WPA2 was a massive improvement to both WEP and WPA, and provides a virtually unbreakable network

given the right inputs (a sufficiently long key).

In this lab, you are going to take a crack-walk through history (a short, but long-winded one). First, you will begin by analyzing traffic on an open network using Wireshark. This will provide a baseline against which you can compare the encrypted traffic later in the lab. You will then explore your first encrypted packets by enabling WEP on the access point and performing another review in Wireshark. Finally, you will capture traffic on this WEP network, and use this captured data to ultimately crack the WEP key and decrypt the information in Wireshark. In Section 2, you will perform similar activities with the aim of cracking a WPA2 network. This will require the adoption of some new techniques, such as handshake captures and password file generation, and demonstrate that even “secure” wireless encryption solutions are vulnerable if they are inadequately configured.

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 examine wireless traffic on an unencrypted network.

2. In the second part of the lab, you will apply Wired Equivalent Privacy (WEP) encryption to the network and re-examine the traffic in Wireshark.

3. In the third part of the lab, you will break WEP encryption by collecting data to crack the passphrase and using it to the decrypt network traffic in Wireshark.

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 apply Wi-Fi Protected Access II (WPA2) to a wireless network and examine the network traffic in Wireshark. You will then break into a WPA2-encrypted network by collecting data during wireless client authentication, perform a dictionary attack to crack the passphrase using the collected data, and then use the passphrase to decrypt the network traffic in Wireshark.

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

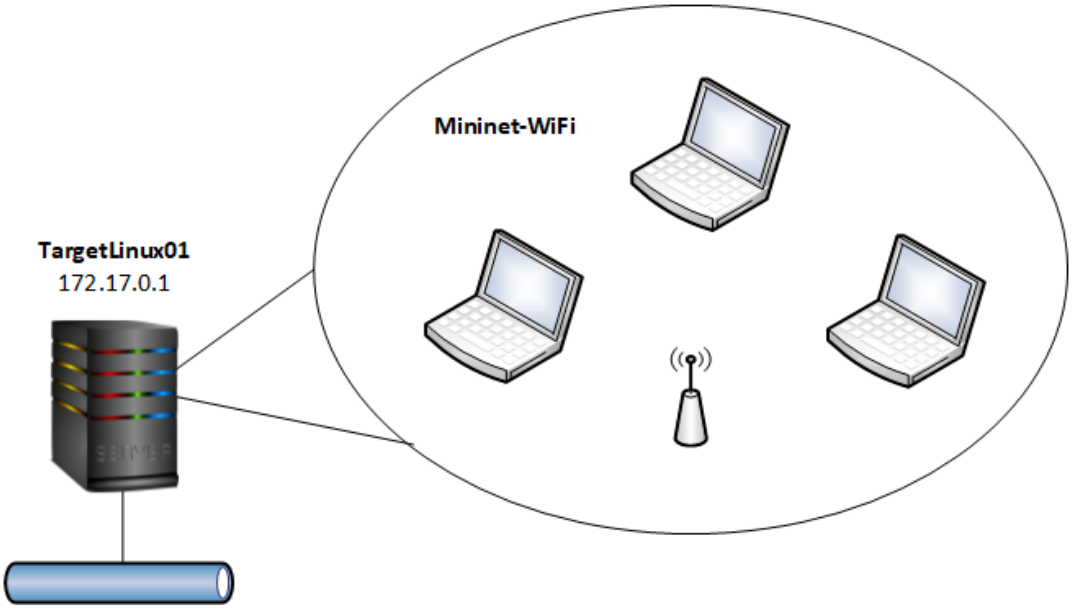
Upon completing this lab, you will be able to:

1. Use Wireshark to analyze wireless traffic.
2. Encrypt wireless traffic using WEP and WPA-PSK.
3. Crack WEP passwords using Aircrack-ng.
4. Create a dictionary file for a brute force/dictionary attack.
5. Break a simple WPA2 passphrase.

Topology

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

- TargetLinux01 (Linux: 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-WiFi
- Wireshark
- Aircrack-ng suite
- CUPP
- CeWL
- John the Ripper
- Docker

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:

- HTTP headers in the Packet Bytes pane
- WEP mode enabled on the GHostAPd Status page
- WEP mode enabled and both sta2 and sta3 devices attached on the GHostAPd Status page
- Initialization Vector value in the Packet Details pane
- KEY FOUND in your aircrack-ng output
- Decrypted Hypertext Transfer Protocol data.

2. Any additional information as directed by the lab:

- None

SECTION 2

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

- “Username” and “Password” form items in the Packet Details pane
- CCMP Ext. Initialization Vector in the Packet Details pane
- Length of your yourname_Capture.txt wordlist in the JtR output
- Discovered passphrase in your aircrack output

2. Any additional information as directed by the lab:

- None

SECTION 3

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

- Recovered WPA2 passphrase in your aircrack-ng output
- Output from your john command used to generate rodRage_Final.lst

2. Any additional information as directed by the lab:

- None

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.

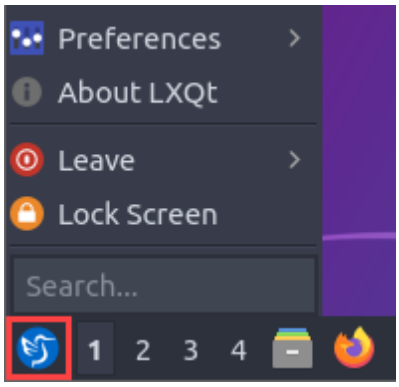
2. **Proceed** with **Part 1**.

Part 1: Capture Unencrypted Traffic with Wireshark

Note: You are seated beside your backpack in a booth at the local café, laptop propped open on the table beside your coffee and notebooks. You are probably using the free Wi-Fi to browse social media and stream some cat videos, before getting back to that assignment you have been putting off. And you are probably, for the most part, using encrypted connections, assuming you are connecting to most of these sites over HTTPS (which utilizes Transport Layer Security, a cryptographic protocol that runs encrypted tunnels to protect data). But you are also probably making DNS requests (“what’s the IP address of google.com”), maybe visiting some more obscure websites that do not run or enforce HTTPS connections, and generating other intelligible chatter that an eavesdropper might be interested in tuning into. When you are forced to use an open connection, it is best to limit your usage of insecure protocols – opting to only browse verified websites over HTTPS, for example – or to take a broader approach and encapsulate most of your application data in an encrypted tunnel through use of a Virtual Private Network (VPN) provider. Otherwise, be careful – you might be performing to an audience who does not have your best intentions in mind.

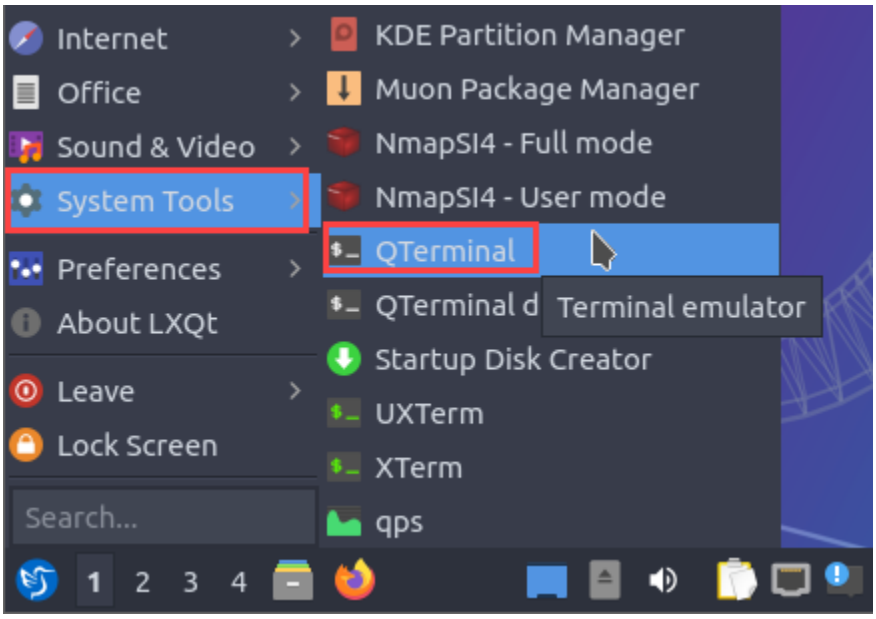
In the next steps, you will activate an emulated WLAN and open a console to one of its stations. From this station, you will create a monitoring interface for capturing packets in Wireshark, a popular protocol analysis tool. After capturing traffic in Wireshark, you will take a brief walk through some of the data to demonstrate the extent of intelligible information that can be gathered in the absence of encryption.

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



Lubuntu 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 open_Sec1.py` and **press Enter** to run the Mininet-WiFi 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 open_Sec1.py
[sudo] password for wifi:
*** Creating nodes
*** Configuring wifi nodes
*** Associating Stations
*** Starting network
*** 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

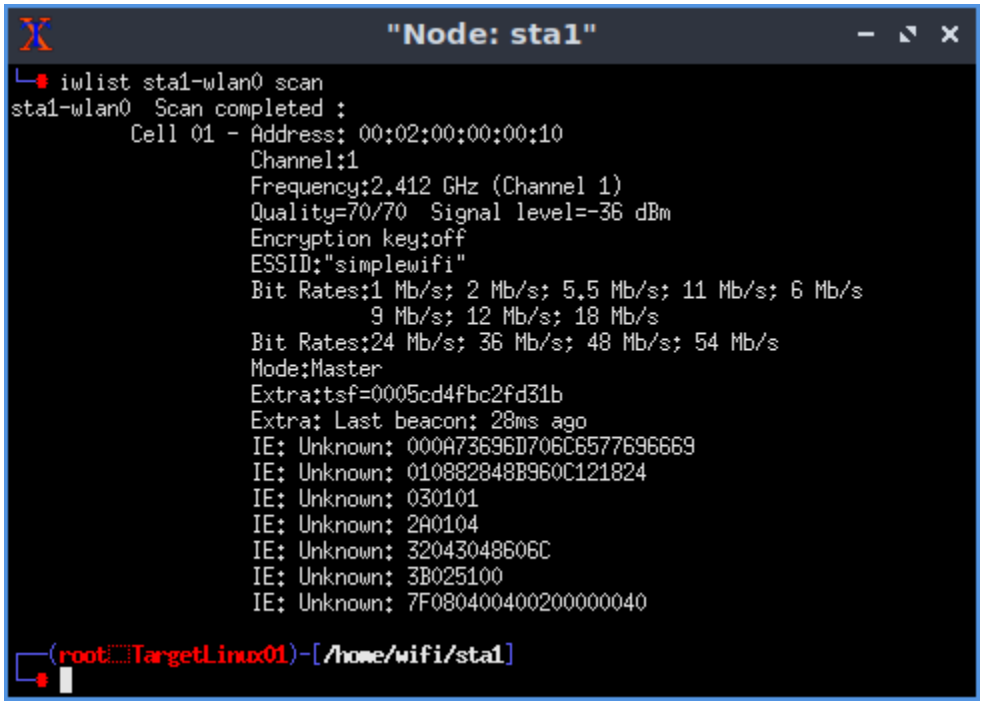
- **Sta1**: 10.0.0.1/24
- **Sta2**: 10.0.0.2/24
- **Sta3**: 10.0.0.3/24

Ap1 represents a wireless access point (WAP) with zero encryption – anyone can authenticate and establish a completely unencrypted connection with it. Sta2 and sta3 are both currently connected to ap1, while sta1 (the station you will work from) is in broadcast range, but not attached to the network.

Two terminals are launched after the network is successfully built: sta1 and sta2. The sta2 terminal is present only to generate traffic for inspection in this lab, and therefore can be minimized.

5. **Minimize** the **Node: sta2 terminal window**.

6. In the Node: sta1 terminal, at the command prompt, **type** `iwlist sta1-wlan0 scan` and **press Enter** to scan the airwaves for access point (AP) beacons using the wireless interface sta1-wlan0.



Scan for AP beacons

Note: Beacons are how access points announce (or “broadcast”) their presence to nearby clients looking to connect. Beacons often contain the Service Set IDentifer (SSID) – more commonly referred to as the friendly name, like “FBIvan” and “DeathStar” – although access points may also be configured to leave their SSID out of these beacons. This might be done to “cloak” the network and thereby improve its security posture, but note that simply turning off SSID broadcasting is not a robust security measure on its own. There are several ways of obtaining the SSID of a hidden network, and it would be a trivial obstacle for a hacker with any determination. It should instead be considered a layer in your multi-layered security solution.

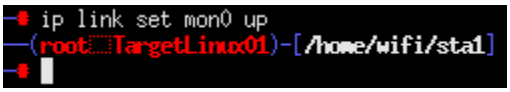
Looking at the output, you should find one wireless network broadcasting on channel 1, which is also broadcasting its SSID: *simplewifi*. Also, notice the encryption key parameter, which is set to *off* – this parameter is specific to WEP encryption and indicates it is not configured on the device.

In the next steps, you will create a monitoring interface for capturing and decoding packets on the network, which you will use to inspect the plain-text traffic being exchanged between the access point and its associated clients.

7. At the command prompt, **type** `iw sta1-wlan0 interface add mon0 type monitor` and **press Enter** to create a new virtual wireless interface called mon0 and place it in monitor mode.

Create wireless interface *mon0*

8. At the command prompt, **type** `ip link set mon0 up` and **press Enter** to bring your new monitoring interface online.

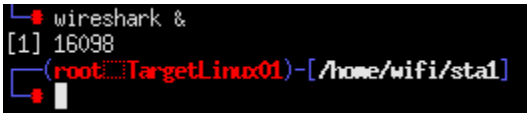


Bring mon0 online

Note: In the next steps, you will use Wireshark, a popular protocol analysis tool, to inspect the traffic being conveyed over-the-air (OTA). In preparation, you have created a new virtual interface on top of your existing wireless interface, which will be dedicated to your packet-capturing activities. This allows you to avoid tying up or altering the configuration of your primary wireless network interface card (wNIC or often just NIC to refer to interfaces broadly).

You have set this new interface to monitor mode, which allows it to inspect and capture packets from networks that you are not even associated with. This is important because typically a network interface card (NIC) only cares about traffic addressed for it. However, if your NIC supports *promiscuous* mode, you might enable it to begin inspecting all packets on your network, regardless of the source and destination addresses. Then, you might go a step further, intending to capture all airborne traffic, regardless of source, destination, or occupant network, in which case you would use the all-hearing *monitor* mode.

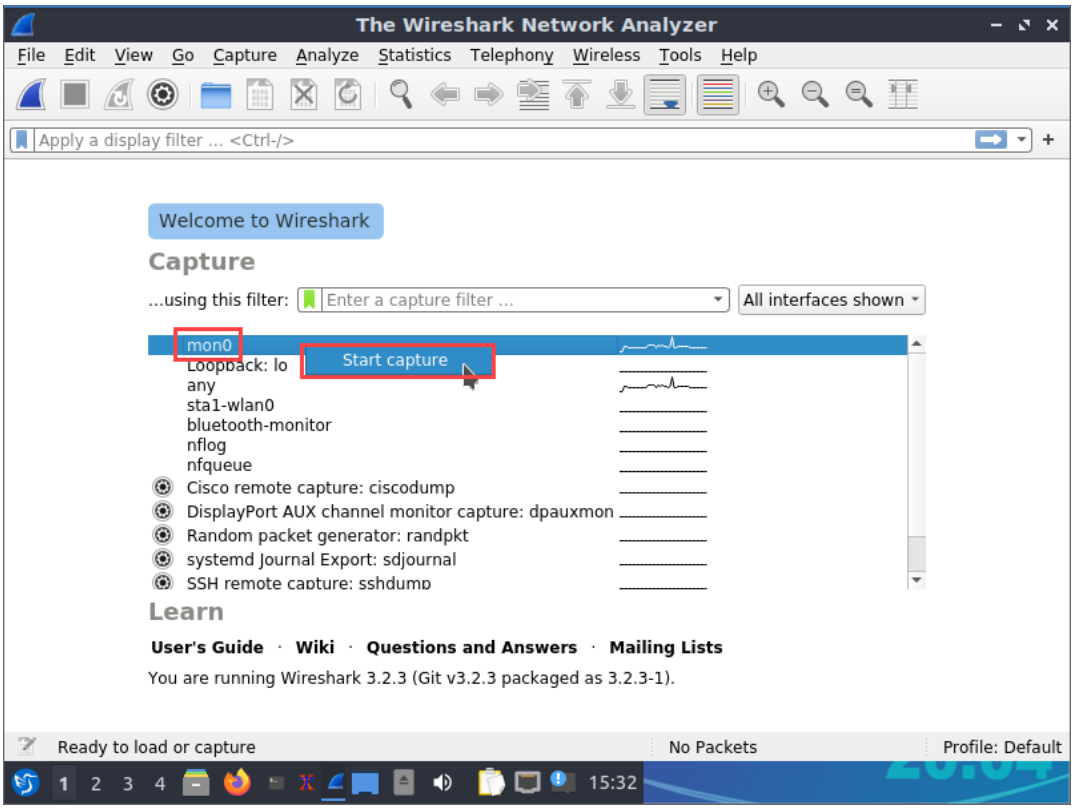
9. At the command prompt, **type** `wireshark &` and **press Enter** to launch Wireshark in the background (&) so that it does not tie up your console.



Launch Wireshark

Note: Welcome to Wireshark! You are provided a list of interfaces you can begin capturing packets on. You want your shiny new mon0, of course.

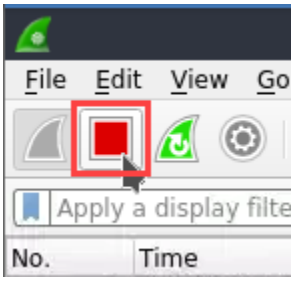
10. In the Capture section, **right-click** the **mon0 interface** and **select Start capture** from the context menu to initiate the capture.



Start capture on mon0

Note: You might see some interesting colors admidst the flurry of broadcast packets, but there is certainly not much time to appreciate them. Let this capture run for 30 to 40 seconds, and then move on to the next step to end your capture.

11. In Wireshark toolbar, **click** the **red square icon** to stop capturing packets.



Stop capture

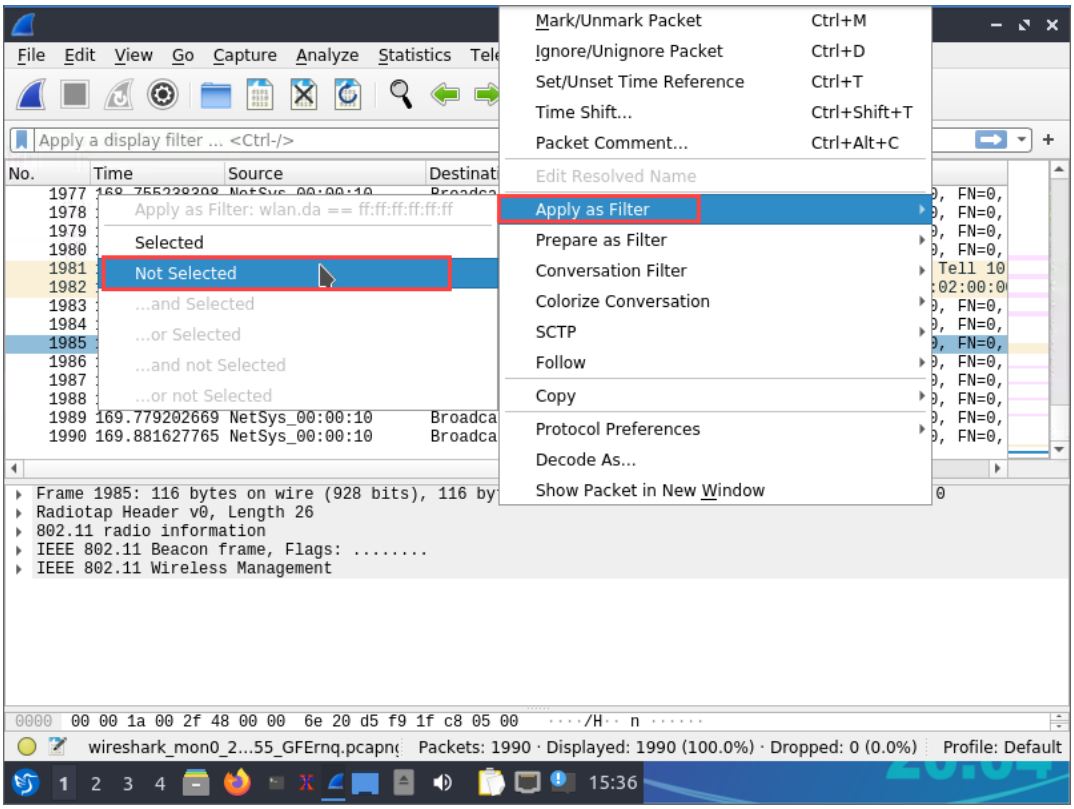
Note: Now is a good time to add a display filter that will hide the broadcast packets and allow you to zoom in on those colorful sequences you may have noticed scrolling by in Wireshark's Packet List pane. This pane, along with the one below it (called Packet Details) is where you will focus most of your attention, but for good measure, here is a brief overview of all three subdivisions in Wireshark's capture view.

Packet List pane (top): This view lists each received packet, with the most recently received packet at the bottom of the list (by default). For real-time captures, this view will change as it receives new packets.

Packet Details pane (middle): This view displays protocols and protocol fields or details of the packet that is currently selected in the Packet List pane. You can click the arrow on the left side of any detail line to see expanded details.

Packet Bytes pane (bottom): This view shows the raw data contained in the currently selected packet. Wireshark displays the packet contents in lines of 16 hexadecimal and the corresponding 16 ASCII characters. Values that do not have printable ASCII characters appear in the ASCII display as a period (.).

12. In the Packet List pane, **select any white broadcast packet**, then **right-click** on **Broadcast** (in the Destination column) and **select Apply as Filter > Not Selected** from the context menu to specify that all packets with a broadcast destination MAC address should be filtered out of the display.



Apply a display filter

Note: You should notice that the display filter bar, located immediately above the Packet List pane, now contains the following expression: **`!(wlan.da == ff:ff:ff:ff:ff:ff)`**. This expression was automatically added as a result of your selection, and uses the exclamation point to indicate NOT the following condition. The condition specifies any packet with a destination address of **`ff:ff:ff:ff:ff:ff`**. This destination address (which refers to the MAC, or physical address, of this interface) is significant, because it represents the Data Link layer (also referred to as Layer 2, the land of ethernet and 802.11) broadcast address. As with all MAC addresses, this address is expressed in hexadecimal, which you can review briefly to appreciate the significance of this address, as well as understand several other references to hexadecimal throughout this lab.

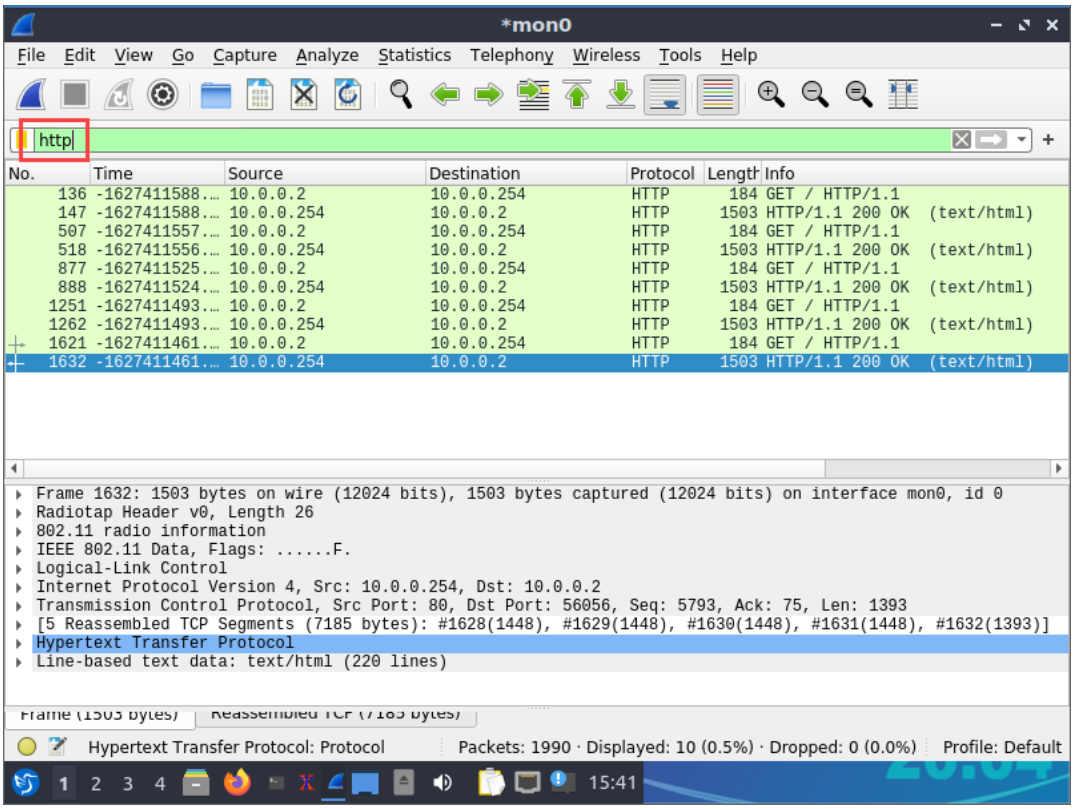
You will find hexadecimal digits used in many places in networking. A hexadecimal digit is a single digit that can store 16 different values (decimal values 0-15). For values 0 to 9, the hexadecimal looks just like a “regular” decimal digit. For values 10 through 15, you will see the letters A through F used. For example, the decimal value 12 would be the hexadecimal digit C, where the decimal value 15 converts to the hexadecimal value F.

MAC addresses, which are used to address traffic on a local network, are composed of six groups of hexadecimal pairs, where each pair can represent a decimal value from 0 to 255. Broadcast MAC addresses – that is, traffic addressed to the entire local network, such as beacons coming from a wireless AP to announce the network’s presence – use the last available address, which means the

highest number in each hexadecimal group: FF.

Here you are filtering out Layer 2 broadcast packets to make room for some of the more interesting traffic passing between connected clients on the network. That interesting traffic comes in the form of some Address Resolution Protocol (ARP) requests, wondering which MAC address a particular IP address is associated with; some ICMP requests and responses (pings), testing reachability to particular hosts over the network; and some HTTP requests and responses, requesting access to specific web resources. Most interesting, however, is not the content itself, but the fact that the content is intelligible at all, and by a station not even connected to the network. Without any data-in-transit protection provided by the AP, it is now the clients' responsibility to take their own security precautions. One might employ a granular approach, disabling all unnecessary multicast (one-to-many) or broadcast (one-to-all) services, and selecting only secure protocols (like HTTPS) for any outgoing connections. Or one might employ a more holistic approach and employ the services of a VPN provider, allowing safe data transport within the protections of an encrypted VPN tunnel (which, with a trusted VPN provider, is a great option if you are forced to associate with an open wireless access point).

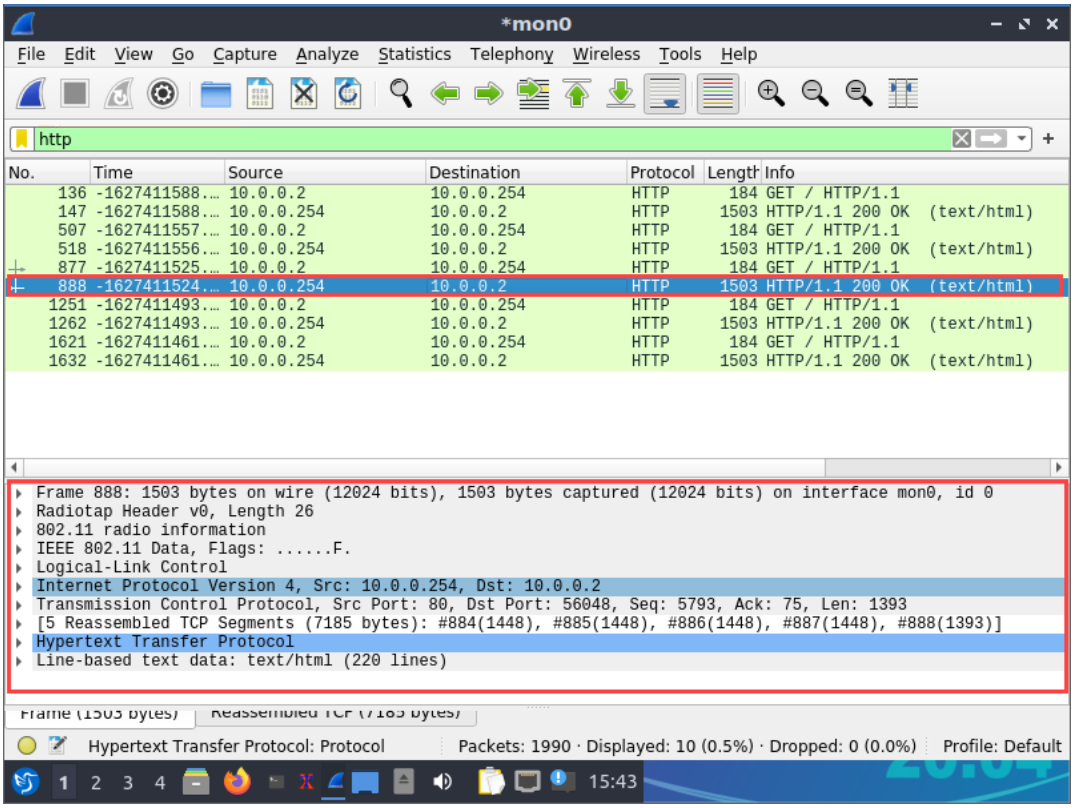
13. In Wireshark, **highlight the text** in the **display filter bar**, then **type** `http` and **press Enter** to replace the current filter with one specifying that only HTTP traffic should be displayed.



Update the display filter

Note: This traffic, generated by the sta2 (10.0.0.2) host in this wireless network, makes intermittent GET requests (requests for a webpage, like your browser makes each time you visit one) to the 10.0.0.254 host for its root web page (GET /), and the host responds back to sta1 with a 200 OK response, and, presumably, the web page. It is all in plain sight, as we will confirm in the remaining steps

14. In the Packet List pane, **select any one** of the **200 OK http response packets** to display its contents in the Packet Details pane.



Select a 200 OK http response packet

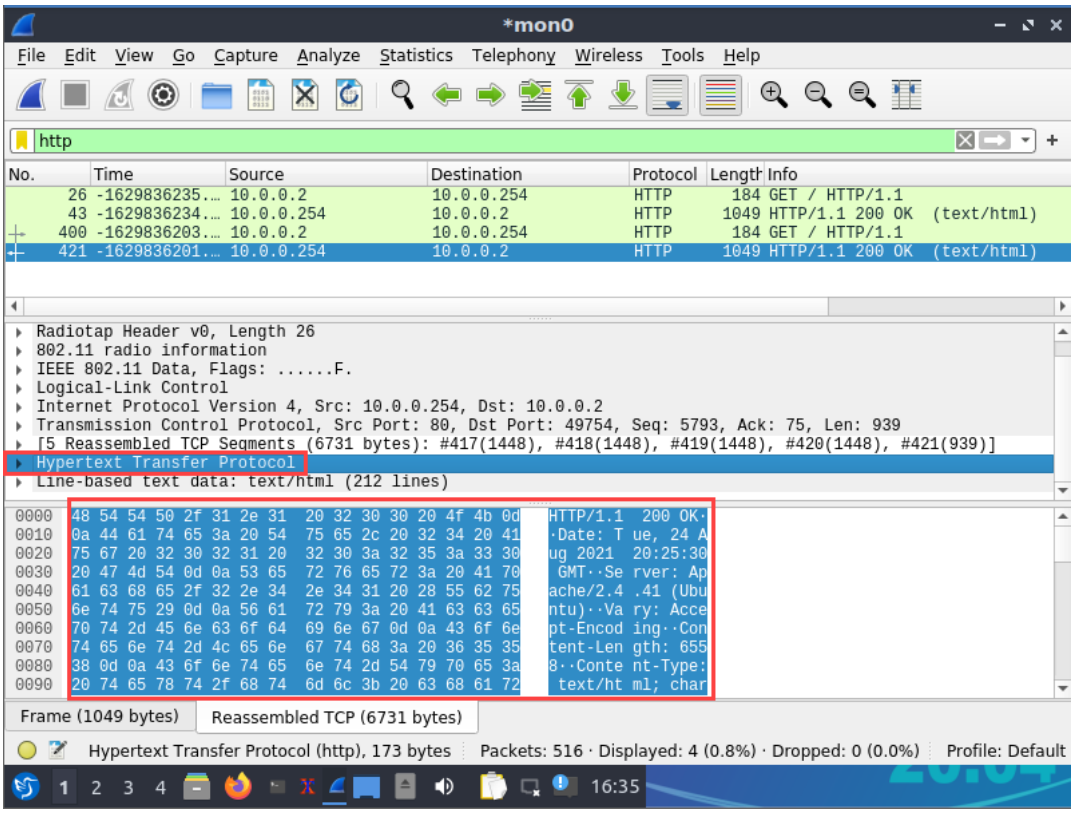
Note: Switch your attention to the Packet Details pane – that's the middle one – and take a second to appreciate the organization. These are protocol fields, which contain information on any of the protocols that were encapsulated in this packet, in the order they were encapsulated. The first field,

Frame, contains metadata on the packets such as response times and protocol usage, and is a Wireshark artifact. The remaining protocol fields are listed based on the order they appear in the network stack, with lower-layer (closer to the hardware) protocols, such as those used in 802.11 Wireless LAN standards, appearing near the top, while higher-layer protocols, such as HTTP (in the Application Layer, further abstracted from the hardware), appear at the bottom.

If wireless encryption were enabled on this network, this would be implemented early in the network stack, at the level of the 802.11 protocol suite. This would have the effect of sealing up all the higher-layer protocol data encapsulated within it (the lower-protocol fields in view), hiding it from inspection – or at least it would in theory, given that WEP has plenty of holes, but that is a matter for Part 3.

15. In the Packet Details pane, **select** the **Hypertext Transfer Protocol field** to display the HTTP protocol information in the Packet Bytes pane.

    You may need to adjust the borders of the Wireshark window to view the Packet Bytes pane. You can also change the resolution of the TargetLinux01 machine from the System > Change Resolution function on the Lab View toolbar to give yourself more screen space.



Display HTTP protocol information

Note: Take a look at the Packet Bytes View, the bottom pane with the hexadecimal rows on the left and the ASCII, interlaced with periods, on the right. You should be able to read the HTTP headers, which specify additional information about the HTTP response, such as the content-type, as well as the server specification (looks like Apache2.4 running Ubuntu!). The *Line-based text data* field immediately following also provides the entirety of the web page, reassembled from the TCP packets that actually transported the data.

Any further discovery here is beside the point, because that point is that none of this application data would be readable if the connections were encrypted. If WEP were enabled, for example, it would provide encryption at the 802.11 field, and provide confidentiality to all higher-layer protocol information (such as HTTP). Heck, nevermind digging around the html in some HTTP responses – if encrypted, you would not even be able to determine HTTP transactions are being made at all!

16. **Make a screen capture** showing the **HTTP headers in the Packet Bytes pane**.

17. **Minimize** the **Wireshark window**.

Part 2: Encrypt Wireless Traffic with WEP

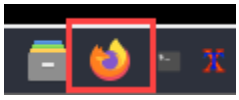
Note: In 1997, the Institute of Electrical and Electronics Engineers' (IEEE) first attempt to implement a security solution comparable to wired networks was ratified: Wired Equivalent Privacy (WEP). WEP selected the RC4 (Rivest Cipher 4) cipher for its implementation, a simple and fast stream cipher described by Ron Rivest (of RSA Security) in 1987. After seven years as an RSA trade secret, RC4 was leaked to the public in 1994, and quickly found widespread commercial use. It was especially popular due to its speed and simplicity – it could run pretty much anywhere with little to no effect on performance. In fact, RC4 appeared in the Secure Socket Layer (SSL) implementation two years earlier than WEP, in 1995, and then later in SSL's successor, Transport Layer Security (TLS), where it even spent a time as the preferred cipher suite for HTTPS in Google Chrome.

RC4 has since been demonstrated to be relatively weak compared to modern-day ciphers, with several vulnerabilities emerging in the years following its leak (although there have been various attempts to renew it, such as RC4+. At the time, it was considered a relatively good choice, when implemented properly – a task that WEP failed impressively by violating one of RC4's cardinal rules, but more on that in Part 3.

In this part of the lab, you will apply WEP encryption to the wireless network using a web-based graphical user interface. This GUI controls hostapd (host access point daemon), a software tool employed on Linux devices to provide access point and authentication services, which is being used to administer this emulated wireless network.

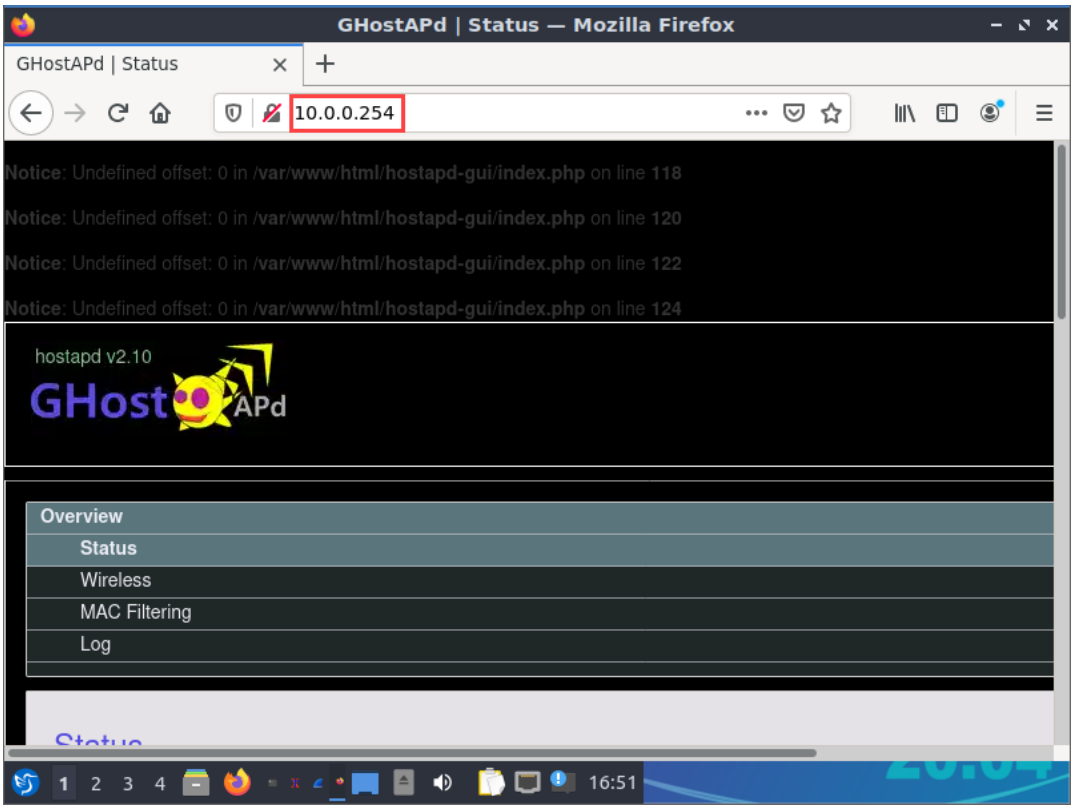
Ensure you have the emulated WLAN running, as instructed in Part 1, steps 1-4, before continuing.

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



Firefox icon

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



GHostAPd page

Note: GHostAPd is a custom front-end built for this lab and provides input to hostapd to make changes to the wireless network configuration. It provides some simple configuration options, such as configuring the security/encryption mode and disabling SSID broadcasts, as well as more advanced parameters like MAC filtering.

Presently, you will find yourself on the status page, which summarizes the settings currently active on the access point. Notice the security mode is currently set to None, indicating the network is wide open. The page also provides a table of the currently attached devices, within which you can see the two network residents, sta2 and sta3, whose traffic you captured in Part 1.

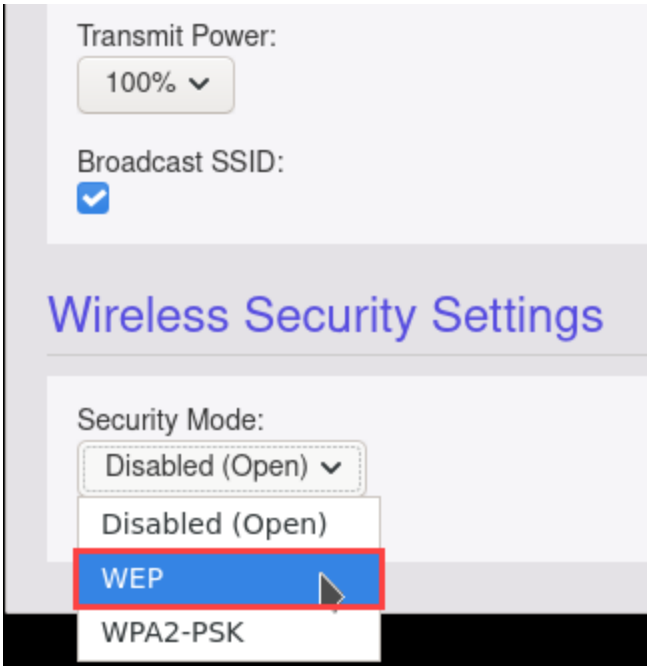
In the next steps, you will enable WEP decryption, which will kick stations 1 and 2 off of the network. You will then reconnect them over WEP, allowing them to resume their network activities, before moving on to the traffic-inspection phase.

3. From the GHostAPd navigation menu, **select Wireless** to navigate to the wireless configuration page.

Wireless configuration

Note: You will leave the other settings alone for now, but optionally you could change the SSID, assign the access point a different MAC address, change the selected 2.4GHz channel, or disable SSID broadcasting.

4. On the Wireless configuration page, **scroll down** to the **Wireless Security Settings section**, then **click** the **Security Mode menu** and **select WEP** from the security mode list.



Select WEP

Note: After selecting the WEP security mode, a passphrase input field will be revealed. Notice the information box below it, which provides you with guidance on configuring the passphrase, or pre-shared key.

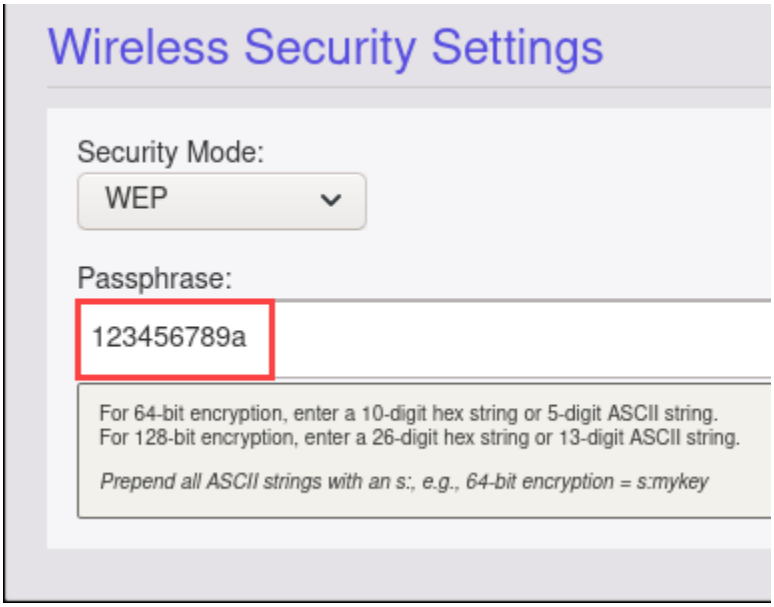
WEP supports two pre-shared key (PSK) lengths, 40 bits and 104 bits, which determine the length of the encryption key used. WEP-40 is implemented when a 10-hexadecimal digit pre-shared key (PSK) is provided, while WEP-104 is implemented when a 26-hexadecimal pre-shared key is provided. Alternatively, you can specify the PSK in ASCII by prefixing it with s: and entering an ASCII PSK of valid length. An ASCII character is 8 bits (1 byte), whereas a hexadecimal character is 4 bits (1 nibble), so you only need to provide half the number of ASCII characters for the same key length (10 hex characters = 5 ASCII characters).

These two modes are typically referred to as 64-bit WEP and 128-bit WEP, but you may also hear WEP-40 and WEP-104 to refer to them. These disparities are a matter of whether you include the initialization vector (IV) or not. Since WEP adds a 24-bit IV to complete the encryption key, then in one case you are referring to the PSK length (WEP-40), while in the other you are referring to the RC4 key length (64-bit WEP), and most importantly, in both, you are referring to the same WEP implementation.

There are also 152-bit and 256-bit WEP variants available from some vendors, but they provide little protection against most WEP-based exploits, because the weak 24-bit IV is central to most and remains unchanged in these key-lengthened variants.

In the next steps, you will implement a 64-bit WEP by specifying a 10-digit hexadecimal (40-bit) string and applying your changes to the access point.

5. In the Wireless Security Settings section, **type `123456789a`** in the Passphrase field to provide a 10-digit hexadecimal string as the PSK for this wireless network.

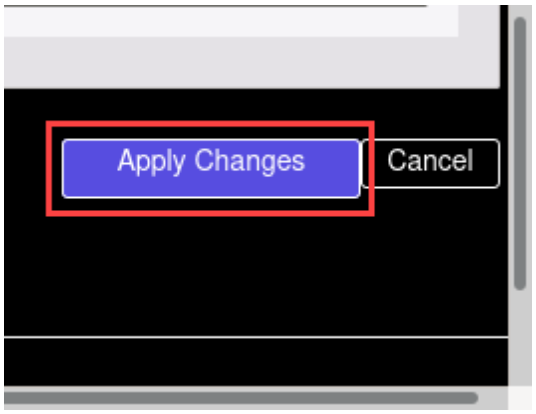


Assign a passphrase

Note: WEP concatenates the IV to the PSK and feeds the result directly into the RC4 algorithm to produce the keystream (the jumbled data that gets combined with your data to produce the ciphertext), which is then used to encrypt your data. This produces a unique key, so long as a novel IV is always used. However, if an IV is reused (for the same passphrase), an identical keystream is produced. This means that anyone with the key can decrypt all traffic on the network.

This contrasts with other methods, such as user-based authentication, which requires that each individual use separate credentials to gain access, or dynamic keying, as employed in WPA2-PSK – which, despite also using a PSK, derives separate session keys for each user (instead of using the PSK directly). In such cases, any key obtained would only give the holder sight into the session it was used for.

6. In the lower-right corner, **click Apply Changes** to reload the access point with your changes.



Apply Changes

Note: The access point may take up to 5 seconds to successfully restart. You will first be presented with an interim page that provides some of the output recorded after your change, allowing you to confirm that the AP was restarted successfully and is using the new parameter values specified.

An automatic redirect will place you back on the Status page, which should now be displaying “WEP” in the Security Mode field. You will also notice that the AP no longer has any attached devices, because authentication is now required to connect. In the next steps, you will re-attach sta2 and sta3 to the network so they can provide some encrypted traffic for you to observe.

7. **Make a screen capture** showing **WEP mode enabled on the GHostAPd Status page**.

8. **Minimize** the **Firefox window**.

9. **Restore** the **wifi@TargetLinux:~/Topos terminal window**.

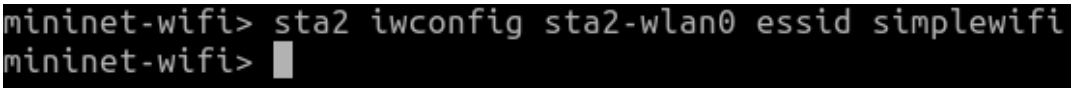
10. At the command prompt, **type** `sta2 iwconfig sta2-wlan0 key 123456789a` and **press Enter** to add the WEP key to the wireless interface on sta2.

```
mininet-wifi> sta2 iwconfig sta2-wlan0 key 123456789a
mininet-wifi>
```

Add the WEP key to wlan0 on sta2

Note: The **`sta2`** invocation is particular to Mininet-Wifi; it specifies that you would like the following command to be run on the sta2 host. The **`iwconfig`** command invokes the iwconfig utility, naturally, which is part of the wireless-tools package in Linux, and allows you to view and set the parameters of a wireless interface (it is the wireless equivalent of ifconfig, a utility you are more likely to be familiar with).

11. At the mininet-wifi prompt, **type** **`sta2 iwconfig sta2-wlan0 essid simplewifi`** and **press Enter** to connect to the network *simplewifi* via the sta2-wlan0 interface.



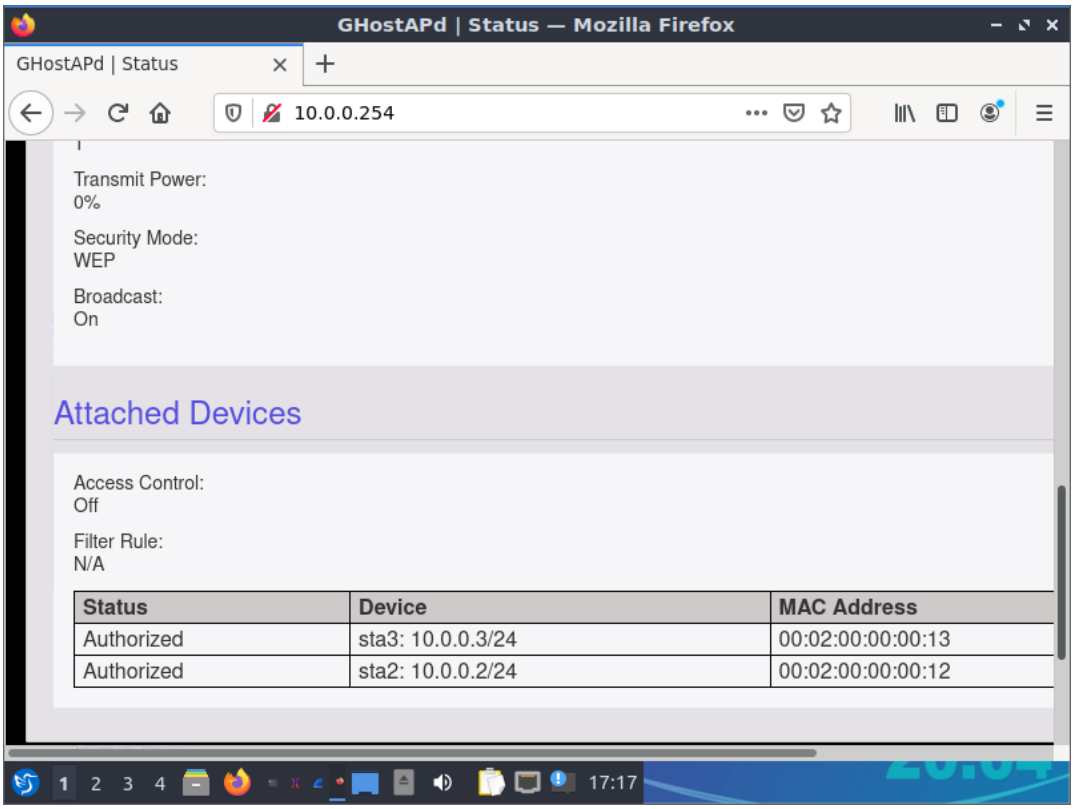
Connect to simplewifi

Note: The ESSID, or Extended Service Set IDentifier, can be safely interchanged with the term SSID. The “Extended” portion is merely to indicate that the SSID may be used to refer to a collection of access points within the same wireless network, as opposed to a single access point.

In any case, your command simply says to connect to the wireless network called *simplewifi* using the wireless network card called sta2-wlan0.

12. **Repeat steps 10-11**, this time replacing all **`sta2`** references with **`sta3`**.

13. **Restore** the **Firefox window**, then **refresh** the **Status page** to confirm the devices were successfully attached.



Attached Devices

Note: The stations may take a few seconds to complete authentication with the access point. If you do not see both sta2 and sta3 in the attached devices page, refresh the page to recheck their status. If either/both fail to appear, repeat steps 10-11 for each missing station.

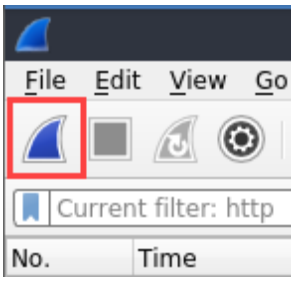
14. **Make a screen capture** showing **WEP mode enabled and both sta2 and sta3 devices attached on the GHostAPd Status page**.

Note: Now that both stations are back on the network, and that network has been fortified with a shiny new (to it) encryption implementation, it is time to pop back into Wireshark to admire the changes up close.

15. **Restore** the **Wireshark window**.

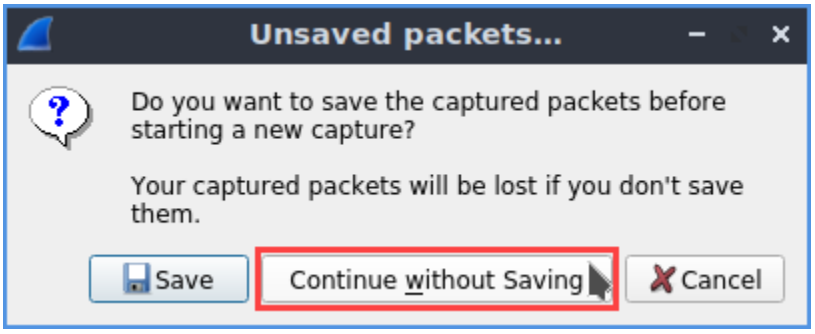
16. In the Wireshark display filter, **highlight http** and **press Backspace**, then **press Enter** to remove your filter from the previous part.

17. On the Wireshark toolbar, **click** the **blue shark icon** to restart your packet capture on mon0.



Start the packet capture

18. When prompted, **click Continue without Saving** to delete your original packet capture.



Continue without Saving

Note: As in the last capture, let mon0 suck up these packets for around 30 seconds, then proceed to the next step.

19. On the Wireshark toolbar, **click** the **red square icon** to stop capturing packets.

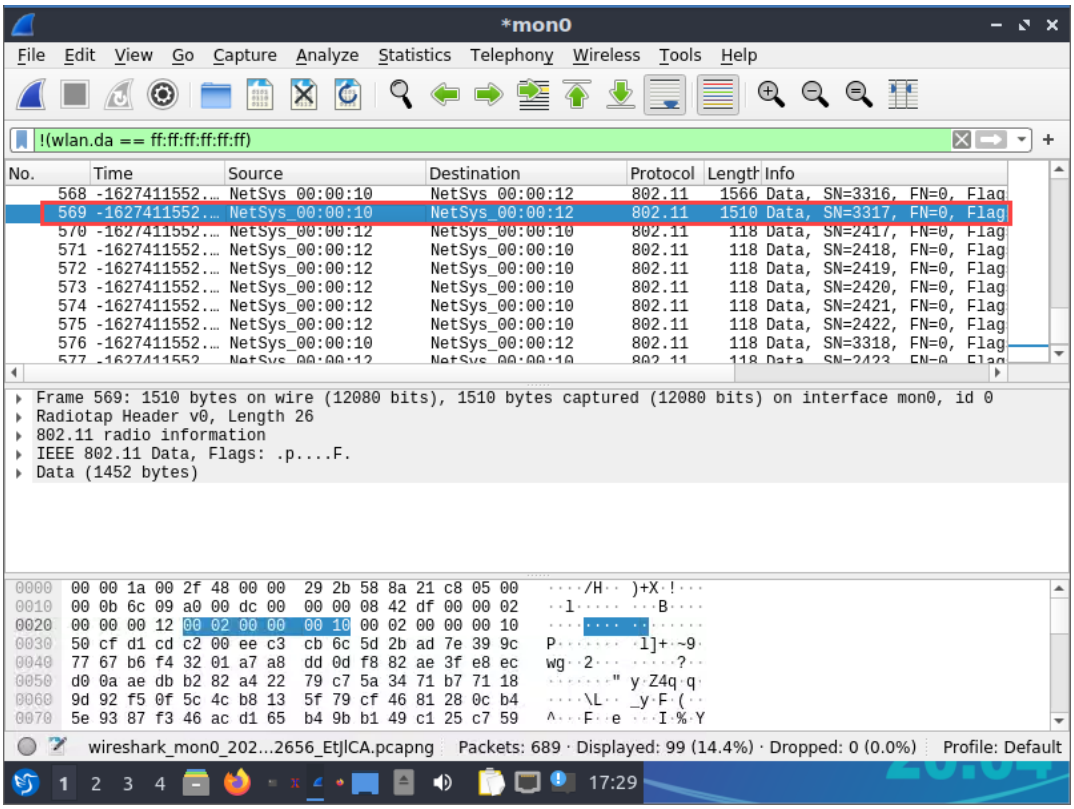
Note: The results are pretty dull this time – there's not very much color. Wireshark automatically color codes protocol information to highlight potentially interesting traffic, which explains your seeing splashes of green (HTTP) and yellow (ARP) in your earlier capture, but this capture does not seem to be displaying any protocol information beyond 802.11.

What about the fields in the Packet Bytes pane? Maybe they contain the additional protocol information, such as the HTTP request data you observed in the previous step. Time to take a closer look to see how far the fog reaches. But first, you will need to get rid of those broadcast packets.

20. **Select a broadcast packet**, then **right-click** on **Broadcast** (in the Destination column) and **select Apply as Filter > Not Selected** from the context menu to filter out all broadcast packets.

21. **Select any packet** with the following source and destination values (ap1 and sta2, respectively):

    Source: NetSys_00:00:10
    Destination: NetSys_00:00:12

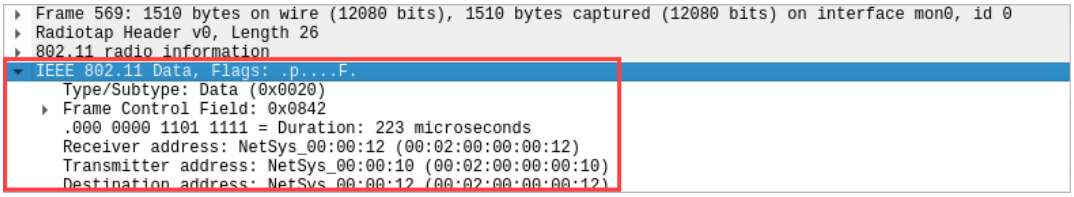


Select a packet from ap1 to sta2

Note: The packet you selected is likely the HTTP response from the AP (ap1) to station 2 (sta2), following the latter's HTTP GET request – much like the packet you inspected at the end of Part 1. See, sta2 runs a small script that generates the same traffic in a 20-30 second loop, and one of those operations is this simple HTTP transaction. Of course, without this knowledge, you could only surmise that there was an exchange between two hosts with the indicated MAC addresses.

If you switch your attention to the Packet Details pane, you will notice that there is no sign of any higher-level protocols: no HTTP headers or data – heck, there aren't even any IP addresses, so even the Network Layer protocols are hidden, which operate in the layer immediately above 802.11's (Layer 2). The only information beyond the 802.11 field is the Data field which, while containing all that other information you have been deprived of, is delivered in an unintelligible scramble due to the encryption, and so Wireshark is unable to unpack it. There is, however, some intelligible information of interest, prepended in cleartext to the front of the packet.

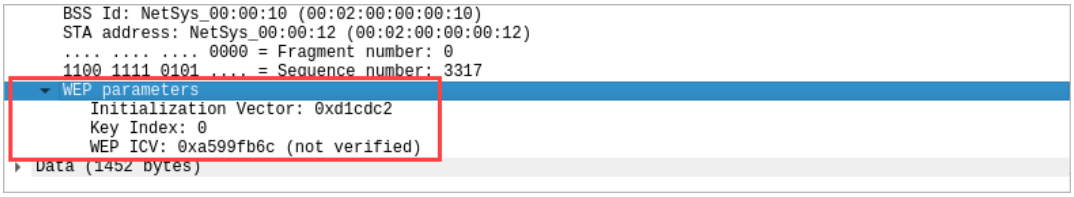
22. In the Packet Details pane, **double-click** the **IEEE 802.11 Data field** to expand it.



IEEE 802.11 Data field

23. **Double-click** the **WEP parameters field** to expand it.

    You may have to scroll down to see WEP parameters field. It is a sub-field within the IEEE 802.11 Data field.



WEP parameters field

Note: Behold the Initialization Vector, in all its hexadecimal glory. You can confirm that the complete 24-bit IV is there– the 0x just indicates the string is hexadecimal, and following it are six hex digits. Each hex digit runs 4 bits a piece, giving you 24 bits total – the complete IV.

You can also see the Key Index, which indicates which key is being used (WEP allows up to four, but only one can be in use at any time). A Key Index of 0 specifies the first key, whereas an index of 1 would specify the second key, and so on – in other words, the list is zero-indexed, as is the convention in computer science.

Last is the WEP Integrity Check Value, a checksum for verifying the data is complete and unaltered after transmission. The ICV is appended to the end of the plaintext data before both are combined with the keystream to produce the ciphertext.

24. **Make a screen capture** showing the **Initialization Vector value in the Packet Details pane**.

25. In the Wireshark display filter, **highlight** the **current filter** and **press Backspace**, then **press Enter** to remove your filter.

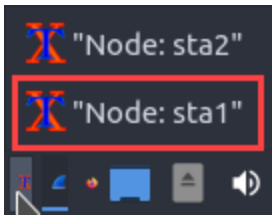
Part 3: Break WEP Encryption

Note: Although WEP's solution did strictly improve the security of 802.11 transmissions, it left much to be desired. WEP boasted many security flaws, the most pronounced of which was its short Initialization Vector (IV). An IV is used as a seed for the encryption algorithm (RC4, in the case of WEP), and is supposed to ensure that if the same plaintext data were encrypted twice, two different ciphertexts (encrypted data) would be produced. This would have been the case if the same IV were never reused, which the creators of the RC4 cipher specified to never do, as a cardinal rule. Alas, the IV WEP used was too short, and so was likely to repeat even within in a single business day on a busy network. Because these IVs were also sent in cleartext on the packet, an attacker could sit idly by, vacuuming up wireless client transmissions indiscriminately, and thereby generate an impressive collection of IVs. Once duplicate IVs begin getting fed into that collection, it would only be a matter of time (or 802.11 frames) before the attacker accumulated enough duplicates to determine the key, and then subsequently decrypt all traffic on the network – much like you are about to do.

In this part of the lab, the objective is to inject some color back into Wireshark. A decryption key would do that nicely, but first comes the actual exploit – time to get yourself some IVs.

Ensure you have the emulated WLAN running, as instructed in Part 1, steps 1-4, before continuing.

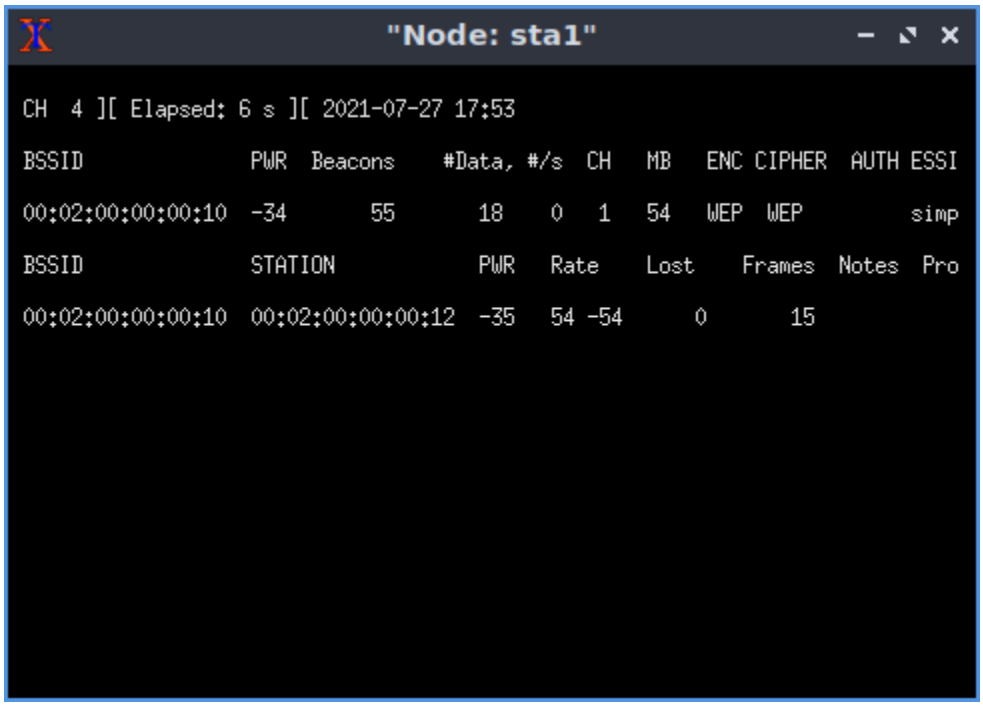
1. On TargetLinux01 taskbar, **hover over** the **XTerm group** to expand the list of open XTerminals, then **select Node: sta1** from the list to restore the sta1 terminal.



Restore the sta1 terminal

2. **Press Enter** to clear the command prompt.

3. At the command prompt, **type** `airodump-ng mon0` and **press Enter** to begin scanning the network for AP beacons.



Run airodump-ng

Note: Airodump-ng is a tool used to capture 802.11 frames and is provided as part of the Aircrack-ng suite, which contains a catalog of robust wireless security testing tools, several more of which you will use throughout this lab. Airodump-ng is particularly well-suited for gathering the IVs on WEP-encrypted 802.11 frames, like the one you inspected in the Part 2, which is exactly what you aim to do. First, a short aside on those IVs:

They are really short. Consider the IV you inspected in the previous part, which was just six hexadecimal characters long. That is what 24 bits buys you. That's three measly ASCII characters. Short. However, that is quite a casual way of appreciating it, since what truly matters are the bits. 24 bits (3 Bytes) gives you ~16 million permutations (24 binary digits = 2^{24} possible combinations). Although this format may be harder to appreciate, know that this number is relatively small for cryptographic purposes, and would be exhausted quite quickly on a busy network. Once it has, you

will start seeing instances of repeat IVs, events that are often referred to as collisions. Collisions are generally a no-no in cryptography, leading to things like the birthday attack, and are particularly problematic in WEP. This is also bolstered by the fact that WEP uses the RC4 stream cipher to encrypt data. Stream ciphers encrypt data one bit at a time, whereas their alternative, block ciphers, encrypt data in predetermined chunks, adding padding if the minimum is not met. Stream ciphers tend to be more efficient, since a simple bit-by-bit approach works well for frequent bursts of small data chunks of unknowable length (like a wireless connection). However, these bursty types of communications are likely to see repeat data in short periods of time, so key reuse becomes a major problem. The Initialization Vector in WEP is intended to prevent this, by seeding the keystream (the text that is combined with your plaintext to produce the ciphertext). However, a 24-bit IV, as has likely been made clear, is just too darn short to prevent key reuse on a busy network. And it is easy to make a busy network, as you will discover. For now, back to airodump-ng.

Airodump-ng will scan all channels in the 2.4GHz range (1-14) for beacon frames and report any wireless access points or stations it discovers. In this case, you have picked up the familiar *simplewifi* network, along with both connected devices. Airodump-ng reports some basic information about the AP, such as the channel it is operating on (1, in this case); the encryption scheme being used (WEP); its ESSID, or friendly name (simplewifi); and its Basic Service Set Identifier (BSSID), which is the MAC address of the AP's radio (or, the not-so-friendly name).

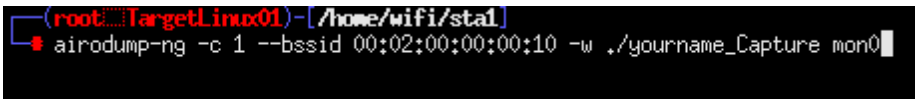
After identifying your target network, it is a good idea to invoke airodump-ng with additional information to narrow its scope. For example, if the target is on channel 1, there is not much point in having airodump-ng run through the other 13, and you will end up missing some of the relevant packets. Airodump-ng also allows you to send the results of your capture to a file, which you can then pass off to additional Aircrack-ng tools for further processing.

Alright, the target is well-defined; now comes the game-time capture, this time using some additional parameters to focus your 802.11 sniffer.

4. **Hold control**, then **press c** to terminate your airodump-ng session.

5. At the command prompt, **type** **`airodump-ng -c 1 --bssid 00:02:00:00:00:10 -w ./yourname_Capture mon0`** and **press Enter** to begin a new 802.11 capture on mon0.

   These additional parameters specify that airodump-ng should listen exclusively on channel (**`-c`**) 1 for traffic that belongs to the network with BSSID 00:02:00:00:00:10, and write (**`-w`**) the output to a file called *yourname*_Capture.



Run a narrower airodump-ng scan

Note: The next steps require patience and grit. You could gather data passively, as you are currently doing, and wait until you have gathered sufficient IVs (hopefully you are very patient). Or, you could get involved, and push things along by flooding the network with packets. The more traffic, the larger your working sample, and the faster you will be able to generate collisions and gather the necessary data for your subsequent cracking operations.

To inject data into the network, you must be associated with the access point. To associate with the access point (gain access to the network), you must first authenticate with it. As specified in 802.11b-1999, an early amendment to the 802.11 standard, these were the two steps that must be observed by any client in order to get access to a WLAN. The association stage is simply establishing a connection with the AP (gaining access to the network), while the authentication phase is where the actual gatekeeping is done. WEP supports the two forms of authentication specified in the 802.11b standard:

Open System Authentication (OSA): No verification of identity takes place (null authentication), so any station can join. All the client is required to present is the SSID and their MAC address.

Shared Key Authentication (SKA): The same static WEP key must be configured on both the client and the AP. This approach has the AP send a challenge text to the client, which the client encrypts with its configured WEP key, and sends back to the AP. The AP decrypts the text, and if it matches the original, the client is authenticated.

You might (reasonably) expect OSA to be the less secure option, but SKA actually presents the bigger threat. To understand why, first you need to know that: (1) just because a station has authenticated with the AP, does not mean it is able to establish an encrypted connection with it – this would still require the correct WEP key configured on the client; and (2) because the challenge-response sequence performed in SKA sends both the plaintext and encrypted versions of the challenge text, then instead of capturing thousands of packets, the attacker need only capture an authentication handshake to get one step closer to obtaining the key. And there are ways of forcing those handshakes to appear, one of which you will explore in Section 2.

It is likely this AP is using OSA, because SKA is pretty much a historical artifact. In fact, beyond WEP, this authentication phase is largely ignored, and so you are likely to find OSA wherever you look. Even OSA is only kept around for backwards compatibility with WEP, after being made a legacy feature in 802.11i-2004, the 802.11 amendment that introduced a new authentication phase. This new phase took place *after* the association phase, since the process was more involved and therefore required first establishing a connection (associating) with the AP.

You can determine which authentication method is being used by inspecting the airodump-ng output. You are looking for the value in that AUTH column on the right side of the screen (the only empty value in the AP row). Okay, small lie – you *could* determine the authentication method *if* you had

captured an authentication sequence. And if you had, you would see either OPN or SKA displayed here.

Instead of waiting around for that information to be harvested, you are going to assume its OSA, and attempt a Fake Authentication with it. This is both because OSA is much more likely, but also because a rejected authentication would mean it is SKA by default, at which point you could just pivot to handshake captures.

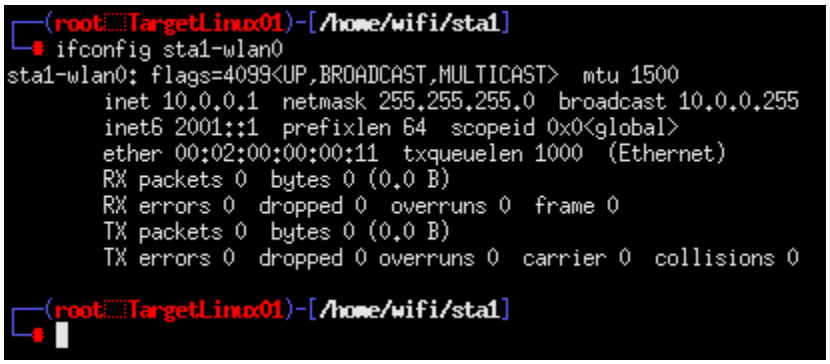
Alright, next, find out what your MAC address is, then perform a Fake Authentication. You will need to open a new terminal, since this one is tied up with airodump-ng.

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

7. At the mininet-wifi> prompt, **type** `xterm sta1` and **press Enter** to launch another terminal for sta1.

Launch another terminal

8. In your new Node: sta1 terminal, at the command prompt, **type** `ifconfig sta1-wlan0` and **press Enter** to view the configuration of your sta1-wlan0 interface.

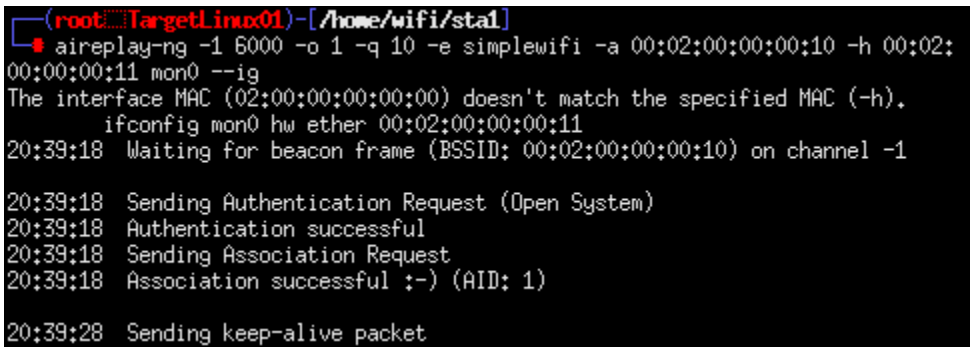


View configuration

Note: The MAC address is four lines down, labelled *ether* (ethernet): 00:02:00:00:00:11.

You already have the MAC address for the AP, as well as its SSID, so you are ready to arrange them all nicely in a command string, add a few more parameters for good measure, and proceed to your ARP injection activities.

9. At the command prompt, **type** **`aireplay-ng -1 6000 -o 1 -q 10 -e simplewifi -a 00:02:00:00:00:10 -h 00:02:00:00:00:11 mon0 --ig`** and **press Enter** to initiate the fake authentication attack.



Run the fake authentication attack

Note: This command specifies aireplay-ng's fake authentication attack (**-1**), which will reauthenticate

with the AP every 6000 seconds, send one set of packets sent at a time (**-o**) to avoid confusing the AP, and some keep-alive packets (**-q**) every 10 seconds. The AP is specified by both its ESSID (**-e**) and its MAC address (**-a**), and you have specified your own MAC address (**-h**) to authenticate. The **--ig** option just ignores a warning generated about the monitoring interface, which is particular to the environment and can be safely ignored.

You should see a sent authentication request (using Open System) followed by an authentication successful claim – unsurprisingly, it is OSA. Time to move ahead with the injection plan.

For this to work, you will require the wireless interface you are using for injection to be associated with the AP (check!) and in monitor mode (also check!). You are now locked, loaded, and ready for an ARP request replay attack. You will listen for any ARP requests on the network and retransmit (replay) any you find back to the AP, which will then repeat it with a new IV. Your airodump-ng process will continue running in the background, harvesting all the IVs you are sowing, and at that point the job is all but done.

First, you are going to need another terminal.

10. **Repeat steps 6-7** to open another terminal for sta1.

11. In your new terminal, at the command prompt, **type** **`aireplay-ng -3 -b 00:02:00:00:00:10 -h 00:02:00:00:00:11 mon0 --ig`** and **press Enter** to begin an ARP request replay attack (**-3**) on mon0, directed at access point with a BSSID (**-b**) of 00:02:00:00:00:10, and your own host's MAC address (**-h**) at 00:02:00:00:00:11.

```
┌──(root㉿TargetLinux01)-[/home/wifi/sta1]
└─# aireplay-ng -3 -b 00:02:00:00:00:10 -h 00:02:00:00:00:11 mon0 --ig
The interface MAC (02:00:00:00:00:00) doesn't match the specified MAC (-h).
        ifconfig mon0 hw ether 00:02:00:00:00:11
20:44:45  Waiting for beacon frame (BSSID: 00:02:00:00:00:10) on channel -1
Saving ARP requests in replay_arp-0727-204445.cap
You should also start airodump-ng to capture replies.
Read 1009 packets (got 504 ARP requests and 0 ACKs), sent 497 packets...(500 pps
Read 1110 packets (got 554 ARP requests and 0 ACKs), sent 547 packets...(500 pps
Read 1211 packets (got 605 ARP requests and 0 ACKs), sent 597 packets...(499 pps
Read 1312 packets (got 655 ARP requests and 0 ACKs), sent 647 packets...(499 pps
Read 1413 packets (got 705 ARP requests and 0 ACKs), sent 697 packets...(499 pps
Read 1515 packets (got 757 ARP requests and 0 ACKs), sent 748 packets...(500 pps
```

Run the ARP request replay attack

Note: It may take up to 30 seconds to pick up your first ARP request ("got # ARP requests" in the first parenthetical of your output), but after you find it, you will begin rapidly reeling them in. Once you have ~30,000 ARP requests, continue to the next step.

12. **Hold ctrl** and **press c** to terminate your injection command.

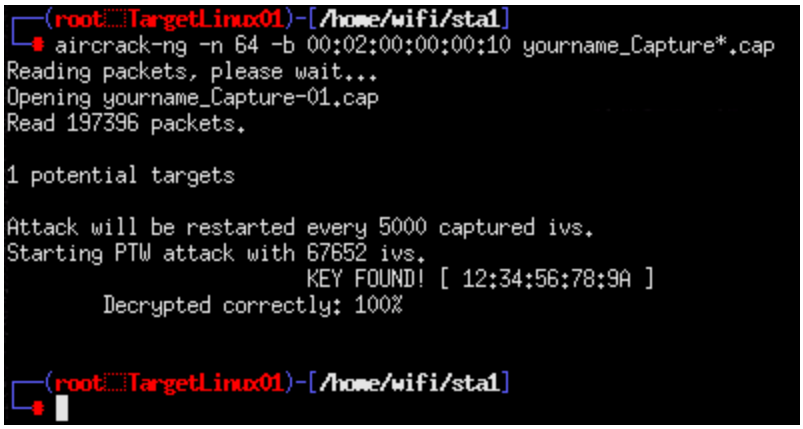
Note: One command stands between you and the WEP key. Here you will train Aircrack-ng suite's eponymous tool on the capture files created by airodump-ng. Aircrack-ng has two primary approaches to WEP key cracking: the FMS/KoreK attack and the PTW attack.

FMS/KoreK is the older and slower choice, and relies on statistical techniques to get close to the correct answer, before finishing off with brute force. This method is predicated on several statistical revelations in WEP, such as an increase in the likelihood of your key-byte-value guess when a particular IV is used (in other words, there are several particularly weak, or leaky, IVs that can be used to infer information about the key). While slower and requiring more packets than the PTW approach, it is handy for when PTW fails.

PTW is a newer technique, and is able to crack keys much faster than the former. It makes use of, and improves upon, several of the techniques used in the FMS/KoreK approach, in conjunction with a key-ranking system that focuses the tool on the likeliest keys first. These were based on additional correlations discovered between the IV and the WEP key. However, this technique only works on ARP packets (good thing you took the ARP injection route to generate IVs), whereas the others can make use of the other packets captured by airodump-ng.

Aircrack-ng also has a plain dictionary attack (using a long list of words to brute force entry), which happens to be the only viable approach to cracking WPA2 (which you will explore in Section 2).

13. At the command prompt, **type** `aircrack-ng -n 64 -b 00:02:00:00:00:10 yourname_Capture*.cap` (replacing *yourname* with your own name) and **press Enter** to attempt cracking a key with a length of 64 bits, using the capture file you created, and looking only at packets with the access points BSSID (**-b**) or MAC address.



Successful key crack

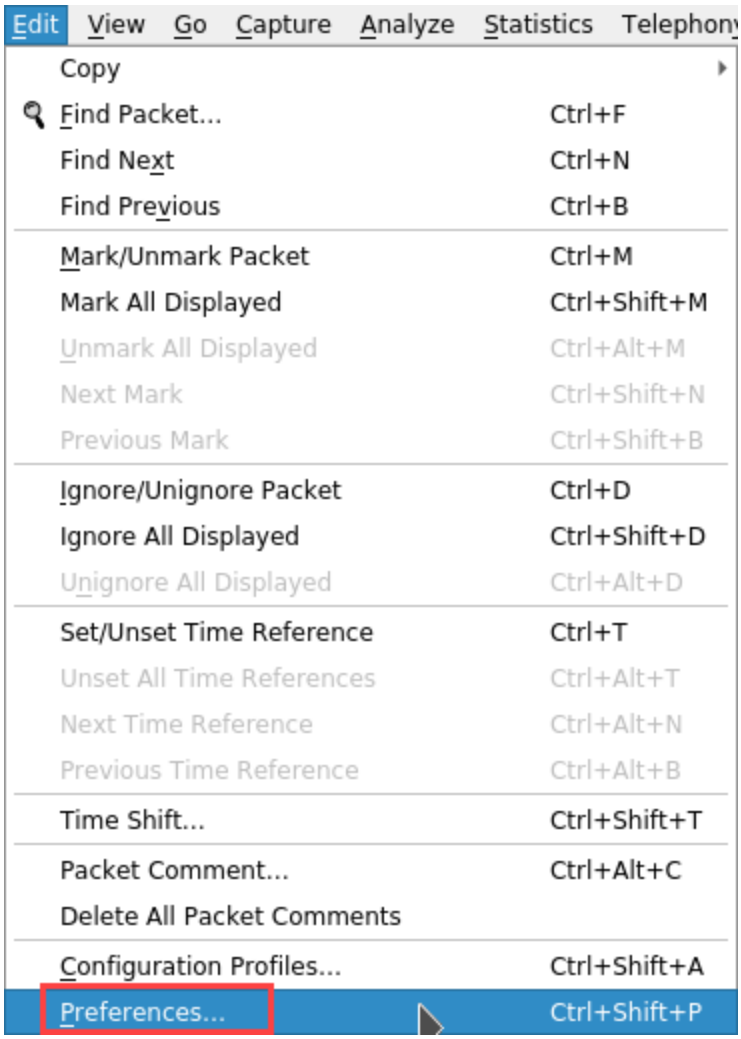
Note: If aircrack-ng fails to find the WEP key, you simply need to obtain more IVs. Reissue the command in step 8 to generate a few thousand more arp requests and then attempt the above command again. Once you have determined the key, proceed to the next step.

14. **Make a screen capture** showing **KEY FOUND in your aircrack-ng output**.

Note: Found the key! No sweat. Now you can obtain an encrypted connection to the AP, just like the other clients, using that same static key. But that isn't what you are interested in right now. Instead, your objective is to perform some decryption on your packet capture and restore those colors back to your Packet List pane, as you will do in the following steps.

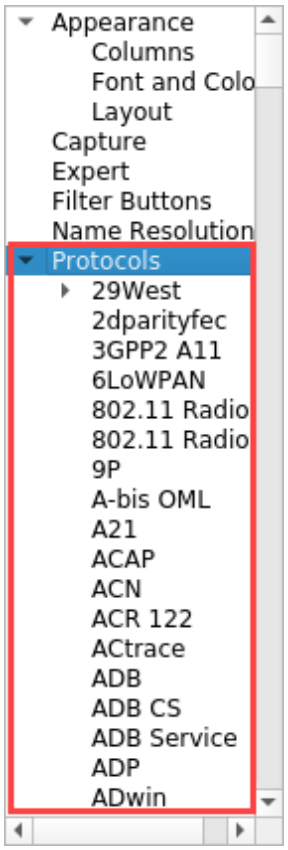
15. **Restore** the **Wireshark window**.

16. From the Wireshark menu, **select Edit > Preferences** to open the Preferences window.



Edit menu

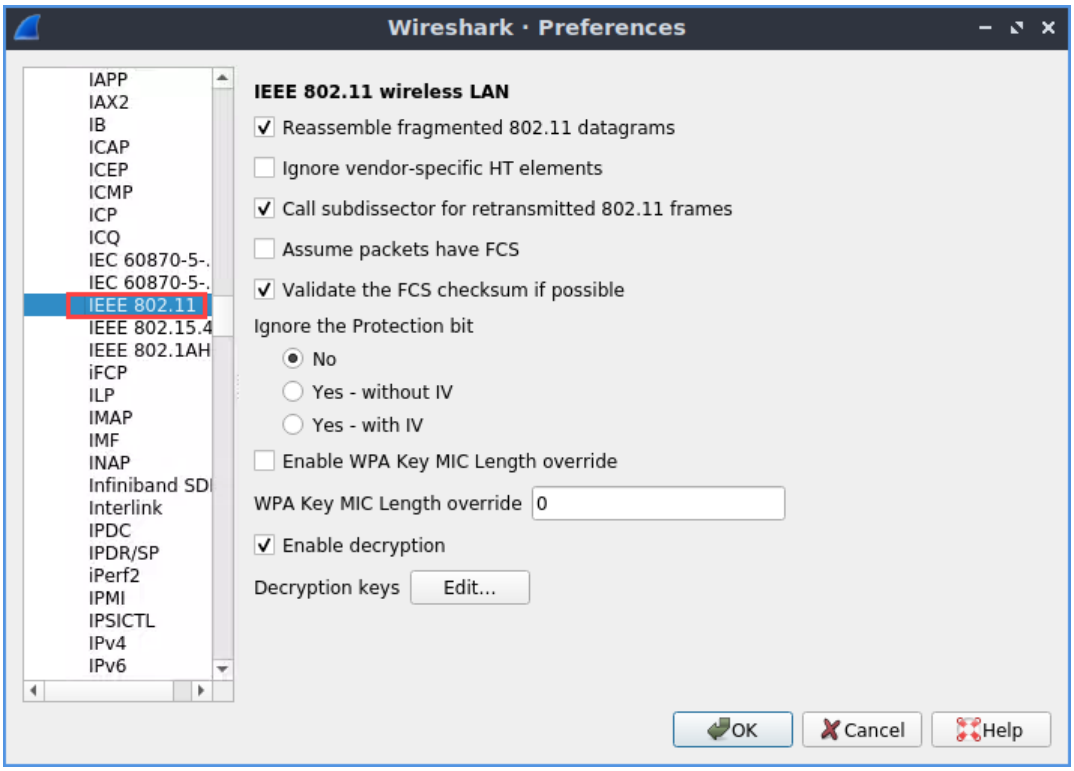
17. In the left pane of the Preferences window, **double-click Protocols** to expand the protocol list.



Protocols

Note: You are looking for IEEE 802.11, which is hard to forget, given that it is the only flavor currently being offered in your colorless Packet List pane.

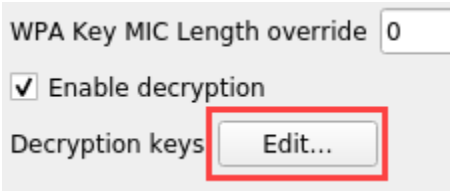
18. Under Protocols, **select IEEE 802.11** to display Wireshark's IEEE 802.11 wireless LAN preferences.



IEEE 802.11 preferences

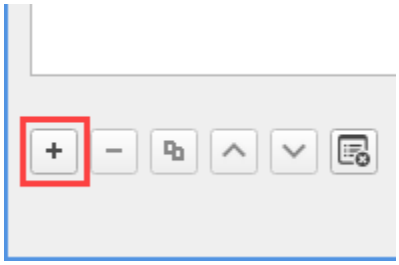
Note: Wireshark is able to decrypt both WEP and WPA/WPA2 traffic that uses a pre-shared key (often referred to as “personal” mode). Decryption is enabled by default, but you will need to add the key obtained by aircrack-ng.

19. To the right of the Decryption keys label, **click** the **Edit button** to open the WEP and WPA Decryption Keys window.



Edit button

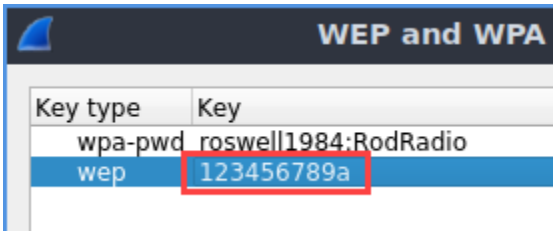
20. In the WEP and WPA Decryption Keys window, **click** the **+ sign** to begin adding a new key.



Add a new key

Note: The Key type in the top left will default to WEP, so no changes are needed there. You just need to add the key in its hexadecimal format, with or without the colons (:).

21. **Click** the **blank row** below the Key column, then **type** **123456789a** and **press Enter** to complete your input.



Enter the key value

22. **Click OK** to complete your WEP key addition and close the WEP and WPA Decryption Keys window.

23. **Click OK** to close the Wireshark Preferences window.

Note: Back in the Packet List view, you should be seeing a little more color, now that Wireshark is able to decrypt the captured traffic. In these final steps, you will filter for HTTP traffic in Wireshark and take one last screen capture to verify your successful breakage.

24. In the Wireshark display filter, **type** `http` and **press Enter** to display only http packets.

25. In the Packet List pane, **select** any **http packet with a Source value of 10.0.0.254**.

26. In the Packet Details pane, **expand** the **Hypertext Transfer Protocol header** to display the unencrypted data.

Note: You should see that all the protocol information is present in the Packet Details pane again, including the Hypertext Transfer Protocol data you observed in Part 1.

27. **Make a screen capture** showing the **decrypted Hypertext Transfer Protocol data**.

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 apply Wi-Fi Protected Access II (WPA2) to a wireless network and examine the network traffic in Wireshark. You will then break into a WPA2-encrypted network by collecting data during wireless client authentication, perform a dictionary attack to crack the passphrase using the collected data, and then use the passphrase to decrypt the network traffic in Wireshark.

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: Capture Unencrypted Traffic with Wireshark

Note: In 2003, the Wi-Fi Protected Access (WPA) standard arrived, conceived as an intermediate solution to the problems with WEP. WPA attempted to improve the WEP implementation by adding several additional security measures, the most notable of which was the Temporal Key Integrity Protocol (TKIP). TKIP ensured a new key would be generated for each packet, thereby protecting the network from many of the weaknesses that plagued WEP. For this reason, it may also be referred to as the TKIP standard.

A year later, in 2004, the cavalry had arrived: 802.11i was ratified, WEP was deprecated, and WPA2 was now in the building. For those with adequate hardware, WPA2 would solve most of its predecessor’s problems. For those stuck on legacy hardware that could not support WPA2, some shade could at least be found underneath the WPA tree.

WPA2/802.11i offered numerous improvements, such as an extension of the IV length (to 48-bits) and the addition of some reuse/replay prevention mechanisms. It also replaced RC4 and TKIP with

Advanced Encryption Standard (AES) and Counter Mode with Cipher Block Chaining Message Authentication Code Protocol (CCMP, I know), and added another authentication stage that came after association. This new authentication stage supported IEEE 802.1X, also known as port-based network access control (PNAC), through WPA2-Enterprise mode, which provided additional methods such as user- and certificate-based authentication. It also provided authentication via a PSK, the old favorite, using this new authentication framework. This resulted in the seemingly redundant connection sequence seen in the standard of: authentication (WEP/legacy) > association > authentication (WPA2/802.11i). As discussed in the previous section, this initial authentication stage would almost always be left to open, or null, authentication (OSA), making it inconsequential in terms of gatekeeping. However, this stage was still required for WEP, and therefore it was kept to continue support for legacy devices. Despite some vulnerabilities creeping up throughout the years, WPA2 was and is considered a fairly secure solution for most users, with its strength ultimately coming down to the length of the key.

In this part of the lab, you will explore a network with the same three stations and one wireless access point as in Section 1, but with some slightly different traffic. You will briefly review the traffic on this open network in Wireshark. You will begin by starting the emulated wireless topology and creating a monitoring interface for your capturing efforts.

1. From the TargetLinux01 application menu, **launch** the **QTerminal application**.

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

3. At the command prompt, **execute** `sudo python open_Sec2.py` to build the wireless network topology containing sta1, sta2, sta3, connected to ap1 over 802.11.

Note: Once the emulated wireless network has started successfully, you will be presented with a console for sta1 and sta2. As in Section 1, sta2 is generating ICMP, ARP, and HTTP traffic in a loop. You will minimize sta2 so that it can continue generating traffic in the background, and then create your monitoring interface, which you will immediately bring online.

4. **Minimize** the **Node: sta2 terminal window**.

5. In the sta1 terminal, **execute** `iw sta1-wlan0 interface add mon0 type monitor` to create a monitoring interface called mon0.

6. **Execute** `ip link set mon0 up` to bring your new monitoring interface online.

Note: Now comes Wireshark. Your next move is to launch Wireshark from sta1 and begin a capture on the mon0 interface. Sta2 should be busy chattering in the background and will run for around 30 seconds before repeating itself.

7. **Execute** `wireshark &` to launch Wireshark and send the process to the background to avoid tying up your console.

   Wait for the Wireshark GUI to load, then proceed to the next step.

8. In Wireshark, **start a packet capture** on the **mon0 interface**.

Note: Colorless 802.11 beacons will be the main feature as you watch Wireshark pull additional packets into the display. As is typical, this access point is set to emit a beacon once every 100 Time Units (TU), where 1 TU is equal to 1.024ms. Basically, one is sent every 102.4 ms, which ends up being around 10 per second.

Let this capture run for 30 seconds so that you grab all the good stuff in sta2's traffic loop, then move on to the next step.

9. **End your packet capture** on mon0.

Note: Next, you are going to filter for HTTP packets in the Packet List view so that you can dig into the insecure web traffic that sta2 is generating.

10. In the Wireshark display filter, **apply a display filter** to show only HTTP packets in the Packet List pane.

Note: Unlike the web traffic you analyzed in Section 1, here you can see a POST request, which could be an exciting discovery. While the GET method is generally used to request a resource, POST is typically used to submit data for processing, which means this POST request could contain sensitive data, such as a username and password submitted to a login system.

Take a look at the information column for these HTTP packets, and notice the HTTP responses to the POST requests. There are several HTTP 200 OK responses followed by an HTTP 302 Found response. A 200 response code indicates the requested page was returned without issue, while a 302 code indicates a temporary redirect. There are a number of possible explanations for that sequence, most which would require additional information to be put forth with any kind of certainty, but a strong working hypothesis might sound like the following:

The HTTP 200 OK responses are failed authentication attempts, the result of the login page being simply reloaded every time the incorrect credentials are given. The HTTP 302 response, then, would be a response to the correct credentials, given that it now redirects to a different page, probably some kind of user dashboard guarded behind the login. It should be easy enough to drill down and test some of this.

11. In the Packet List pane, **select a packet** containing the **HTTP 302 Found response**.

12. In the Packet Details pane, **expand** the **Hypertext Transfer Protocol field**.

Note: Take a look at the location header, which specifies the URL to redirect to. It looks like it goes to index.php, which is typically the web page loaded when you navigate to the top directory of a web site (http://example.com/ vs. http://example/specificPage.php). This would support your hypothesis that the page is a dashboard guarded behind the login. If that is the case (it is), the POST request immediately before this 302 Found packet probably contains the correct login credentials.

13. In the Packet List pane, **select** the **HTTP POST packet** preceding the HTTP 302 Found response packet.

14. In the Packet Details pane, **expand** the **HTML Form URL Encoded field**.

Note: How convenient, itemized and everything – sure enough, this user was submitting credentials in an HTML form, and now you have them!

15. **Make a screen capture** showing the **“Username” and “Password” form items in the Packet Details pane**.

16. **Remove** the **HTTP display filter**.

Part 2: Encrypt Wireless Traffic with WPA2

Note: In this part of the lab, you will configure WPA2 encryption on the ap1 access point using WPA2-PSK, and then perform a final capture and review of the same traffic, now encrypted.

Use of a pre-shared key for authentication is available in all WPA versions, but is most commonly used in small office/home office environments (SOHOs). It is often referred to as WPA-Personal. The other option is WPA-Enterprise mode, which requires a RADIUS (remote authentication dial-in user service) server for authentication and provides a number of alternative authentication strategies through 802.1X, or PNAC. This can include basic credentials-based authentication (username and password, as opposed to just a single PSK), as well as mutual client/server authentication using certificates. You may also hear it referred to as WPA2-802.1X, or just WPA2, to differentiate it from its SOHO sibling.

WPA2-Enterprise is reserved for organizations, and requires more work to get running, but offers a number of protections that increase your network security posture. For that reason, WPA2-PSK/WPA-Personal is most often seen in the wild, as it is easy to set up and offers fairly strong security for the average user. It is also the only implementation you might actually have a chance of cracking, but the security of any WPA2 network usually comes down to the strength of its key: a sufficiently long random string for a key makes the network virtually unbreakable. Your mother's maiden name, on the other hand? That's one express ticket to pwnville.

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

2. In Firefox, **navigate** to **`http://10.0.0.254`** to access the GHostAPd web interface.

3. From the GHostAPd menu, **navigate** to the **Wireless configuration page**.

4. Under Wireless Security Settings, **select** the **WPA2-PSK option** from the Security Mode menu.

Note: You should see a box appear with password length specifications. WPA2-PSK encrypts traffic using a 128-bit encryption key that is derived from a 256-bit shared key. This shared key is the titular PSK, and can be set as either a 64-digit string of hexadecimal characters, or as an 8 to 63-character ASCII string (which WPA2 converts to a PSK). The dialog seems to suggest ASCII is expected, so you will go ahead with an ASCII string.

Some access points will also allow you to specify the encryption method, usually offering

WPA2-CCMP (or WPA2-AES, depending on the AP vendor, where both refer to the same thing) and WPA2-TKIP. The latter is provided for backwards compatibility with WPA and is by far the weaker choice. The GHostAPd GUI does not provide this option, instead defaulting to WPA2-CCMP, although it is configurable directly in Hostapd, the command-line access point and authentication software that GHostAPd provides input to.

5. In the Passphrase input field, **type `r3@Lse<ur3badminton`**, then **apply your changes** to reload the access point with your new encryption parameters.

Note: Congratulations, you have created a virtually unbreakable network (discounting that large vulnerability seated between your computer chair and keyboard). Sure, you have a dictionary word in there, but even if it were somehow known, there are still 800 trillion possible combinations to guess from, which would reliably keep out all but the most determined and resourceful threats – a nation state, for example, or large organized cybercrime group.

Okay, so that is not quite true, and you should aim for a better password at home. The barrier to cracking this password is going to be largely determined by its susceptibility to various techniques available, which may leverage statistics like word frequency (for example, the number of appearances in password dumps from previous breaches), and word substitution (not “real”? Hmm, let’s try “r3@l”, that’s a common substitution), so the length and number of possible combinations are not the full story. Your best bet would be to use a random string of characters, with no words or word substitutions – which is made easier with a password manager – but in the event you opt to have such statistical targets in there, you can use your awareness of these techniques to grab some big security gains with some small changes. For example, one password checker claimed that the password you used above would take around 4 months to crack, given current techniques. If you replace the "a" in badminton with "@", the time-to-crack doubles. That's one easily remembered letter substitution and you’ve bought yourself another 4 months. But wait, "@" is probably a pretty common substitution for ‘a’, so what if you tried using an asterisk (*) instead? Now you've got 3000 years!

In the next part, you will get to take a shot at a WPA2 network using an attack that will be predicated on brute-forcing a weak password. You will have to hope the administrator of that network heeded none of what was said above, or this assignment might be coming in late. For now, get sta2 and sta3 back on your WPA2-PSK WLAN, then hop back into Wireshark and explore the effect of your changes.

6. **Make a screen capture** showing the **GHostAPd Status page with WPA2 enabled as the Security Mode**.

7. **Minimize** the **Firefox window.**

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

9. At the mininet-wifi> prompt, **execute** `xterm sta2` to open a new terminal for sta2.

Note: iw and iwconfig only support WEP configuration. When connecting to a WPA2-enabled network from a Linux shell, you must use wpa_supplicant, a Linux command-line WPA client and 802.11i supplicant – basically, a program for making login requests to a wireless 802.11i-supported network. Wireless supplicants will actively maintain your connection in the background, and can automatically reauthenticate when disconnected.

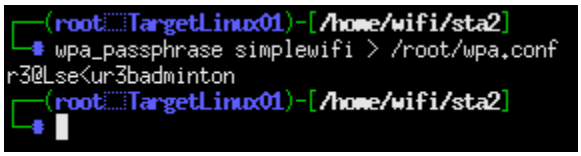
Much like with iwconfig and WEP, access to the WPA network through wpa_supplicant is a two-step process that you will follow in the next steps: (1) add the wpa passphrase, and (2) connect to the network.

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

Note: Wpa_passphrase takes your console input as an ASCII key and converts it to a PSK. The PSK and SSID entries are then written into the wpa.conf file in the appropriate format.

11. **Type** `r3@Lse<ur3badminton` and **press Enter** to complete the command.

    Your passphrase is configured. Now you just need to connect.



Configure a wpa passphrase

12. At the command prompt, **execute** `wpa_supplicant -B -D nl80211 -i sta2-wlan0 -c /root/wpa.conf` to connect to the network with interface sta2-wlan0, using the configuration (**-c**) described in the /root/wpa.conf file (the SSID and WPA passphrase you configured in the previous steps) on sta2.

Connect to the network

Note: The process is also placed into the background (**-B**) so that it does not tie up your terminal. This is common, as wpa_supplicant was designed to run as a daemon, managing your wireless connection from the background. The **-D** option specifies the wireless drivers to use, which are interface dependent. In this case, you are using Linux softMAC drivers, and have therefore selected nl80211.

In the next steps, you will repeat the same process for sta3. This is one more opportunity to practice your skills with wpa_supplicant. Instead of opening a terminal for sta3, you will just direct your commands to it from the mininet-wifi prompt by adding “sta3” to the beginning of each command.

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

14. **Repeat steps 9-12** for sta3.

    When the Node: sta3 window opens, you will need to **press ctrl+c** to restore the command prompt.

Note: Now that both devices are re-connected to the network, you will return to Wireshark and capture the newly encrypted traffic.

15. **Restore** the **Wireshark window**.

16. In Wireshark, **restart** your **capture on mon0**.

    When prompted, continue without saving your previous capture. Leave this capture running for 30-40 seconds, then continue to the next step.

17. In Wireshark, **stop** your **capture on mon0**.

Note: Hopefully you are unsurprised to find Wireshark is painting you another colorless tableau. No sign of those HTTP packets you checked out earlier, but there is stuff happening in there, so why not first clear out those broadcast packets so you can focus on the 802.11 traffic sta2 is generating.

18. **Apply a display filter** to hide all broadcast packets.

19. **Select any 802.11 Protocol packet** exchanged between **NetSys_00:00:12** and **NetSys_00:00:13.**

    These stations represent sta2 and sta3, respectively.

20. In the Packet Details pane, **expand** the **IEEE 802.11 Data > CCMP parameters**.

Note: The introduction of AES-CCMP into WPA2/802.11i represented a fundamental switch from stream ciphers to block ciphers in the IEEE 802.11 standards. Remember, a block cipher, such as AES, encrypts data in data chunks of preset lengths, adding padding to shorter blocks so that they meet the minimum. However, block ciphers operate only on single blocks, and therefore require a mode of operation (or chaining mode) to describe how that cipher is to be repeatedly applied to successive blocks to “chain” them together. For AES in WPA2, these details are taken care of by CCMP (CCM is technically the mode of operation, where CCMP is the WPA2 protocol that utilizes it), which provides one chaining mode for encryption (Counter Mode, or CTR), and another for authentication (Cipher Block Chaining Message Authentication Code, or CBC-MAC). Conventionally, the name of an implementation will indicate the block cipher algorithm utilized, followed by its mode of operation (as in AES-CCMP).

Looking over the CCMP parameters, you may be surprised to see the IV freely available in cleartext again. However, the IV does not typically need to be secret. The more important requirement that you will find in most prescriptions is that the IV never be reused -- in other words, it must be a nonce, or a number used once (although some definitions of a nonce include the requirement it be random/pseudo-random/unpredictable, ruling out non-repeating sequential sequences). After all, the whole point of an IV is to ensure the same plaintext is not encrypted using the same key, so doing so would be

antithetical to its design. Stream ciphers are particularly vulnerable to IV-related attacks, and since the RC4 keystream in WEP was dependent only on the IV and the PSK, reusing this pair would produce the same keystream and make traffic decryption trivial.

Block ciphers are highly dependent on their selected mode of operation, both in terms of IV requirements and consequences for IV reuse. For example, CBC (Counter Block Chaining), requires the IV to be both unique and unpredictable (not necessarily random), and IV reuse will reveal information about the first plaintext block. Others like CTR effectively turn the block cipher into a stream cipher, and so IV-key pair reuse may produce an identical keystream, quickly trivializing any encryption on the network. CTR is, in fact, the mode used by CCMP for encryption. However, the IV used is twice the size as in WEP, and derived partly from a replay counter that resets each session to prevent IV reuse.

While this is all very interesting, you are not exactly interested in this traffic, nor the broadcast packets. See, WPA2-Personal encrypts each client session with a different session key, which is separately derived from the PSK. This means that to discover the PSK, you must go after the frames exchanged during the WPA2 authentication handshake, as you will do in the next part. This next part will also require a new wireless topology, so you will need to reset the lab.

21. **Make a screen capture** showing the **CCMP Ext. Initialization Vector in the Packet Details pane**.

22. From the Lab View toolbar, **select Options > Reset Lab** to reset TargetLinux01 in preparation for Part 3.

Part 3: Break WPA2 Encryption

Note: 802.11i-2004 brought with it a new approach to a new authentication called the four-way handshake. This handshake entails an exchange of 4 messages used to prove to each party that the other knows the secret code, without ever revealing the code. In WPA2-Personal, that code *is* the PSK, but you may also hear it referred to as the pairwise-master-key (PMK). The PMK is ultimately used to generate the pairwise-transient keys (PTK) keys that are used to encrypt the session (remember that AES-CCMP uses 128-bit block sizes\encryption keys? These are those).

Despite the 802.11i authentication phase serving as a major upgrade to the original 802.11b pre-association authentication used by WEP, vulnerabilities still emerged in the implementation, as they tend to. One such instance is the KRACK vulnerability, which took advantage of the fact that at stage 3 in the four-way handshake, the AP provides an encryption key to the client, and if the client does not respond, it will send it again. Over and over. An attacker can then replay and analyze those messages until they have discovered the code.

Firmware updates were later released to address KRACK, but the standard approach to cracking WPA2-PSK remained: the offline brute-force/dictionary attack. This approach takes advantage of the

message integrity code (MIC) calculated during the handshake. The MIC serves a similar role to the ICV, or integrity check value, you saw in the 802.11 WEP parameters in Section 1 – that is, verification that the data is from the stated sender and was unaltered in transit. While the PSK or PMK is not sent during authentication, it is used to calculate the MICs, which are. Once a handshake is captured, the attacker can simply make infinite guesses about the PSK by calculating their own MIC values and comparing them to those in the captured exchange until a match is found. However, this attack vector goes away with a sufficiently long password – a 63-digit string of random ASCII characters is not going to be brute-forced in this lifetime.

The wireless topology in this part consists of 3 stations and 1 access point, just like the others explored earlier in the lab. However, in this case, the access point has a different SSID and is already configured for encryption. In addition, your sta1 machine has been swapped for a Kali Linux Docker container. Kali is a Linux operating system that was built with penetration testing and digital forensics in mind. Produced by Offensive Security, Kali is not the only robust operating system for security testing, but it is perhaps the most popular (especially after being featured in the television series Mr. Robot). You will leverage this Kali machine to craft a targeted dictionary and crack the WPA2 network.

In this part of the lab, imagine that you are a gray-hat hacker who has been hired to breach a target's network. According to the person that hired you, the target is a friend of his, to whom he would like to teach a friendly lesson about the importance of cybersecurity. First, you will need to start your Mininet-WiFi topology.

1. From the Lubuntu application menu, **launch** the **QTerminal application**.

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

3. At the command prompt, **execute** `sudo python wpa2_Sec2.py` to build and start a preconfigured WPA2 wireless topology.

   When prompted for the sudo password, **use** `wifi`.

Note: Sta2 is back making some noise in this topology, so you will again get a console for it. In addition, you are going to see a blank one for sta3. These are both involved in the communications taking place on this wireless network, so let them continue their conversation, but in the privacy of your taskbar.

4. **Minimize** the **Node: sta2** and **Node: sta3 XTerminals**.

5. In the QTerminal window, **double-click the grey area** to the right of the **wifi@TargetLinux01:~/Topos tab** to open a new terminal in this window.

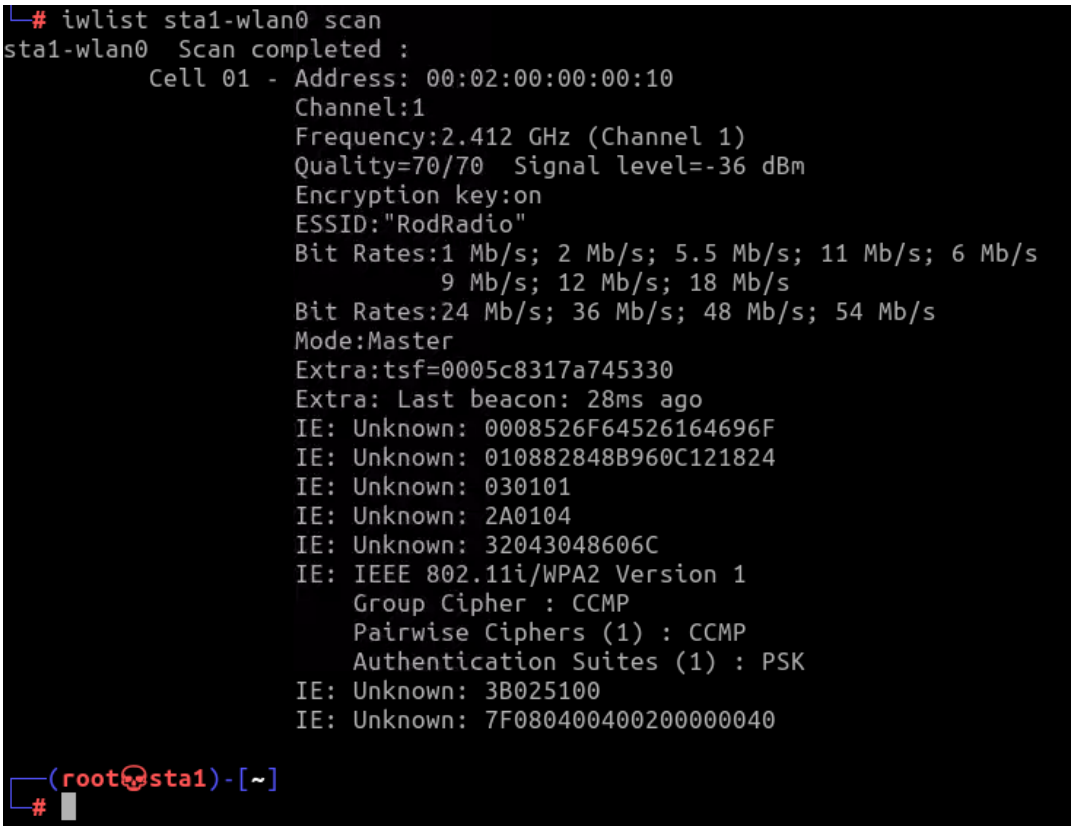
6. At the command prompt in your new terminal, **execute** `sudo docker attach mn.sta1` to attach to your fancy new Kali container.

   When prompted for the sudo password, **use** `wifi`.

Note: You are now seated behind the control panel of the almighty Kali. In the next steps, you will use the familiar airodump-ng and aircrack-ng tools for your capture and cracking efforts, but you will also engage with some new tools to assemble a targeted dictionary file based on information you have gathered on your target.

But first comes discovery. Nothing fancy, just a scan to determine some information about the AP.

7. At the command prompt, **execute** `iwlist sta1-wlan0 scan` to use your sta1-wlan0 wireless interface to scan for AP beacons.



iwlist scan results

Note: Iwlist is an extension of the iwconfig command-line utility, which displays configuration information about the interface. It includes a scan utility that will report on all access points in range. There are a couple things worth noting from the output.

First, the address (or BSSID): 00:02:00:00:00:10. You will need this address to train airodump-ng exclusively on this network. Second, this network also has a friendly name, or ESSID: RodsRadio. Third, and the one that should perhaps be checked first, to find out if you even have a chance: the wireless encryption mode. Look for the IEEE 802.11i/WPA2 Version 1 specification about halfway down the page. It looks like you are dealing with a WPA2 network using CCMP (AES-CCMP) as its cipher mode and most importantly, it performs authentication with a PSK, which means there's a chance!

In the next steps, you will run airodump-ng in a tmux session. Tmux is a terminal multiplexer, which is tech-jargon for a program that lets you run multiple terminal sessions in a single terminal display (using what are called pseudo terminals). Tmux is popular in the hacking community, where it is common to have several programs running simultaneously (as you may have discovered in the previous Section), and window management can quickly become a problem. Tmux allows you to list all running sessions, and easily attach to or detach from these sessions (or pseudo terminals) from a single window. And these sessions are user-based, so even if you close the terminal window, you can

open another and access your sessions from it.

8. At the command prompt, **execute** `tmux new -s dump` to create a new interactive tmux session (`-s`) called dump.

Note: The bottom of your screen will turn green, and your session name will be displayed in the lower left corner. This is useful for keeping track of which session you are currently in at any moment.

9. At the command prompt in your dump session, **execute** `airodump-ng -c 1 --bssid 00:02:00:00:00:10 -w ./yourname_Capture sta1-wlan0` to start an airodump-ng capture on sta1-wlan0 using the information you obtained from your run of iwlist scan.

Note: You are looking for EAPoL packets, which will show up in the Notes column for the respective wireless client shown in the bottom half of your airodump-ng output. These EAPOL packets indicate a 4-way handshake is taking place, which consists of a four-message exchange between the Supplicant (the wireless client seeking to authenticate) and the Authenticator/Authentication server (the AP is both in this case). This is the data you need, which contains the bits necessary for PSK discovery, and you have two options for obtaining it.

- **Wait**. Just wait until someone connects and you are able to capture some EAPoL data. It is bound to happen eventually. How patient are you? When is this due?

- **Deauthenticate**. Here you are actively trying to disrupt a user's connection, hoping to trigger a reconnection. The client's supplicant software is likely running in the background (where wpa_supplicant is usually placed) and may automatically reconnect to (and so perform a 4-way handshake with) familiar access points.

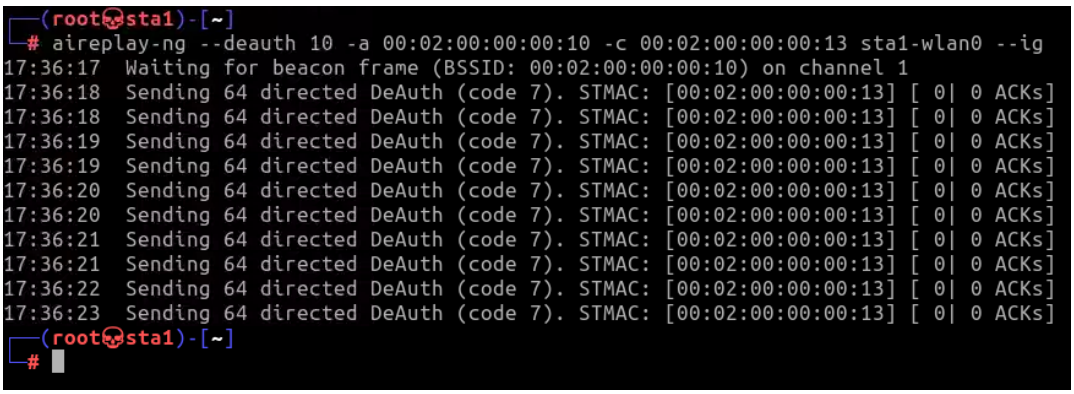
The IEEE 802.11 standards have a provision for deauthentication frames, which allow an access point to disconnect specific clients from the network. By issuing the target client a deauthentication frame, you can terminate their connection with the access point to force a reconnection, and then capture the handshake packets in Wireshark for offline study.

Option B? Proceed to take out sta3.

10. **Hold ctrl**, then **press b**. Release both, then **press d** to detach your session.

Note: When you spend a little time in Tmux you get used to pressing Ctrl-b, which is the typical way to say “the next key press should be interpreted as a command.” The “d” key binding that follows allows you to easily detach from a session, and perhaps pop back into another session to review your progress there. By detaching, you return to your main console and place your dump session into the background where it will continue running

11. At the command prompt, **execute** `aireplay-ng --deauth 10 -a 00:02:00:00:00:10 -c 00:02:00:00:00:13 sta1-wlan0 --ig` to direct a set of 10 sets of (64) deauth packets at sta3 from the sta1-wlan0 wireless interface.



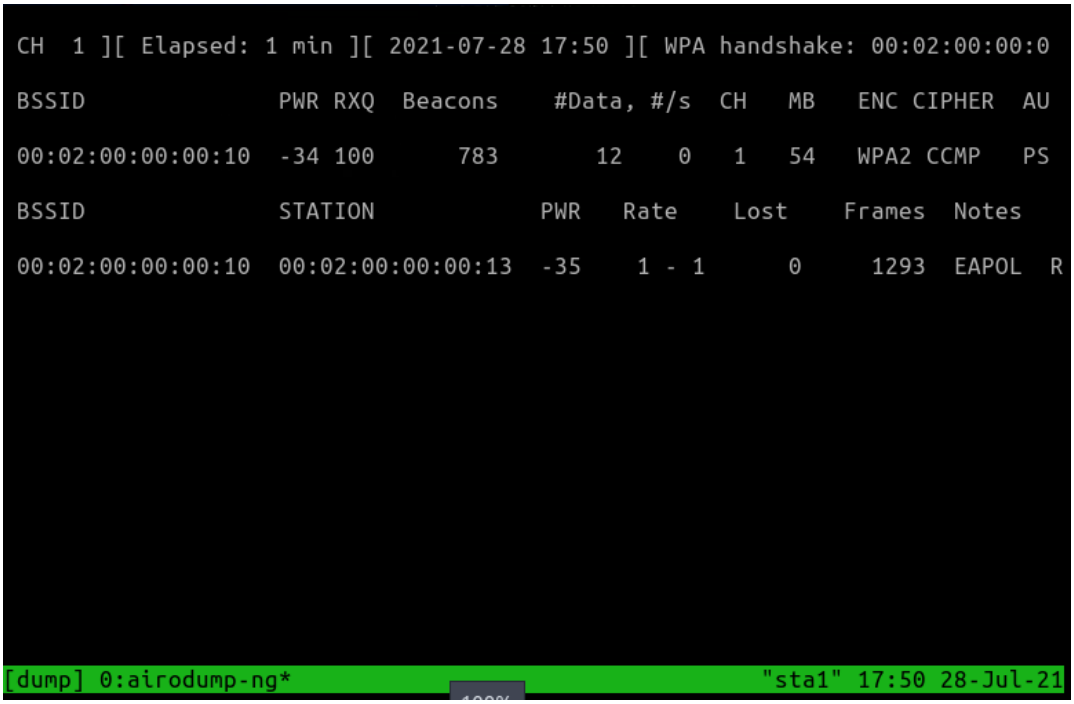
Run aireplay-ng

Note: Deauthentication can be accomplished by sending frames to the AP or the client. Aireplay’s deauth attack performs both, resulting in a total of 128 deauthentication frames sent out (64 to the AP, 64 to the client).

Wait for the command to complete, then continue to the next step.

12. At the command prompt, **execute** `tmux a` to attach (**a**) to your only running tmux session.

    If you had several sessions running, you could specify your attachment target by name using **`tmux a -t sessionnamehere`**



Airodump tmux session

Note: Notice the Notes column beside 00:02:00:00:00:13 (sta3) – EAPOL! The live part is over, now you can get into the workshop and wordsmith your way to victory. You will begin by using a Linux command-line utility called CUPP, or Common User Password Profiler. Using its interactive mode, you will answer several prompts about the target to generate an appropriate wordlist of possible passwords. At this point, you would have performed some Open Source Intelligence Gathering on the target (OSINT), combing popular information repositories such as social media platforms and e-commerce sites for posts or reviews, and accumulated some personal information you can feed to your list generators. In this case, that work has already been done for you – here is your briefing on the target:

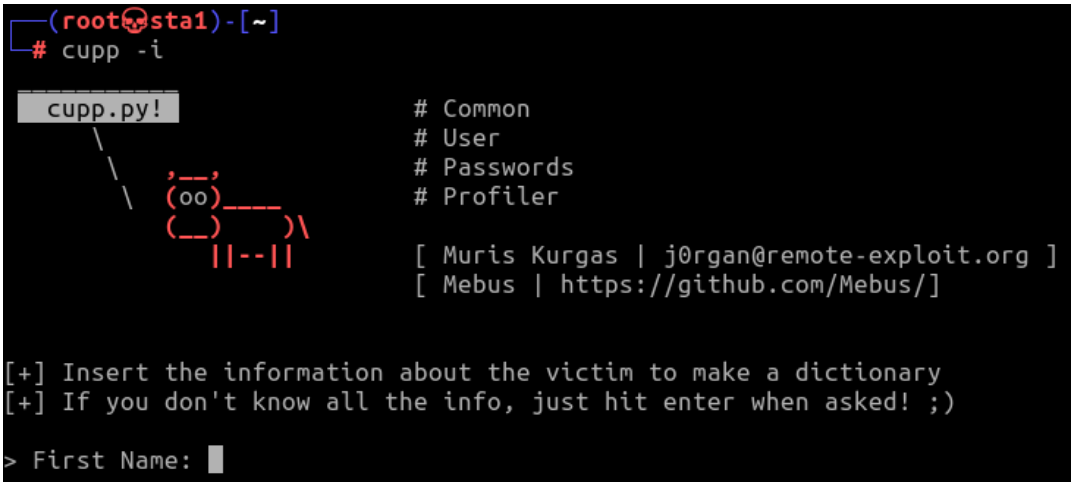
RodRadio, the target ESSID/SSID, belongs to Rodney Walton, a 30-40 something who is frequently posting about aliens and area51 on Facebook. He has a dog, Roz, short for Rozweiller; a son, Marshawn, who he affectionately refers to as Marsbar; and he runs an Ebay seller account as MotherShipper.

Good enough for your purposes. In the next steps, you will end the capture session and start profiling.

13. **Hold ctrl** and **press c** to terminate the airodump-ng process.

14. At the command prompt, **execute** `exit` to kill the dump session and return to the main prompt.

15. At the command prompt, **execute** `cupp -i` to start the Common User Password Profiler in interactive mode.



CUPP interface

16. **Complete** the survey using the following information, presented in order. If no value is specified, **press Enter** to proceed past it.

    First Name: **Rodney**
    Surname: **Walton**
    Nickname: **Rod**
    Birthdate:
    Partner's name:
    Partner's Nickname:

Partner's Birthdate:
Child's name: **Marshawn**
Child's nickname: **Marsbar**
Child's birthdate:
Pet's name: **Roz**
Company name:
Do you want to add some key words about the victim? **Y**
Please enter the words, separated by comma: **rozweiller, ebay, mothershipper**
Do you want to add special chars at the end of words?
Do you want to add some random numbers at the end of words?
Leet mode?

Note: Your dictionary is complete – well, almost. Aliens and area51 were featured in his OSINT profile, so it would have made sense to include these in the additional key words section. However, you are going to use a separate program to create an additional dictionary file, which you will then combine with this CUPP list (saved as rodney.txt, as shown in the output) to produce your initial dictionary of seed words.

17. At the command prompt, **execute** **`timelimit -t 40 -s 2 cewl -d 2 -m 7 -w area51.txt https://en.wikipedia.org/wiki/Area_51`** to use the Custom Wordlist Generator to scrape the Area 51 web page and return a list of words, saved as area51.txt.

Note: The **`timelimit`** command at the beginning specifies that this command should run for (**`-t`**) 40 seconds, after which the SIGINT signal (**`-s 2`**) should be sent, which is the same signal you send to a process when you use the ctrl-c hotkey combination to stop it (SIGnal INTerrupt). Timelimit is mostly used for the purposes of this lab, to ensure your results remain consistent with the lab guide, but it is a handy tool nonetheless.

The **`Cewl`** command then specifies a depth (**`-d 2`**) to spider (move from one web page to another to gather data), and to only grab words with a minimum length (**`-m`**) of 7. While the minimum passphrase length for WPA2 is 8, you will be adding additional data to these words. 7 just seems like a good number for catching any juicy proper nouns, doesn't it?

Once the command finishes and the prompt returns, continue to the next step.

18. At the command prompt, **execute** **`cat rodney.txt area51.txt | sort --unique > rodRadio.txt`** to combine both of your wordlists, remove any duplicates, and save the remaining entries in a single file called rodRadio.txt.

Note: **`Cat`** normally outputs information to the console (well, technically STDOUT), but the pipe (|) indicates the output (the wordlist content) should be passed to another program. In this case, that program is **`sort`**, which is invoked with **`-unique`** to remove any duplicates in the list before writing the sorted output to a file (**`>`**) called rodRadio.txt.

19. At the command prompt, **execute** **`wc –l rodRadio.txt`** to invoke the Linux word count utility to find the number of words in your custom dictionary file.

Note: Your dictionary should have around 10,000 words. It is short, and simple, with no fancy word combinations. The password would have to be a single word for this dictionary to be effective, and it's hard to believe Rodney would have been that careless. And he wasn't, but you are not out of luck yet. An old picture surfaced in Rod's Facebook memories that will provide a significant upgrade to your dictionary. Here's the scoop:

This picture showed the dashboard of Rod's new AP/router/firewall install, during a period when he was really into cyberpunk and hacking games. He managed to install some custom firmware on a consumer-grade router/firewall he purchased and was showing off his new firmware to get that sweet social credit. The dashboard displayed his configuration screen with a slice of the WPA2 passphrase, but blurred out. However, what looked like the third and fourth to last digits were poorly obscured, and looked like they could be a 1 and a 9.... Hmm. Could it be a birthday? Rod is probably in his 30s or 40s, so adding the years 1970-2000 to the end of each word in that dictionary might cast a wide enough net, while still keeping your wordlist relatively small.

Hopefully he hasn't changed that password since the old post – back to the writing room.

20. At the command prompt, **execute** **`john --wordlist=/root/rodRadio.txt --rules=years --stdout > `*`yourname`*`_Wordlist.txt`**, where *yourname* is your own name, to construct your end-game dictionary.

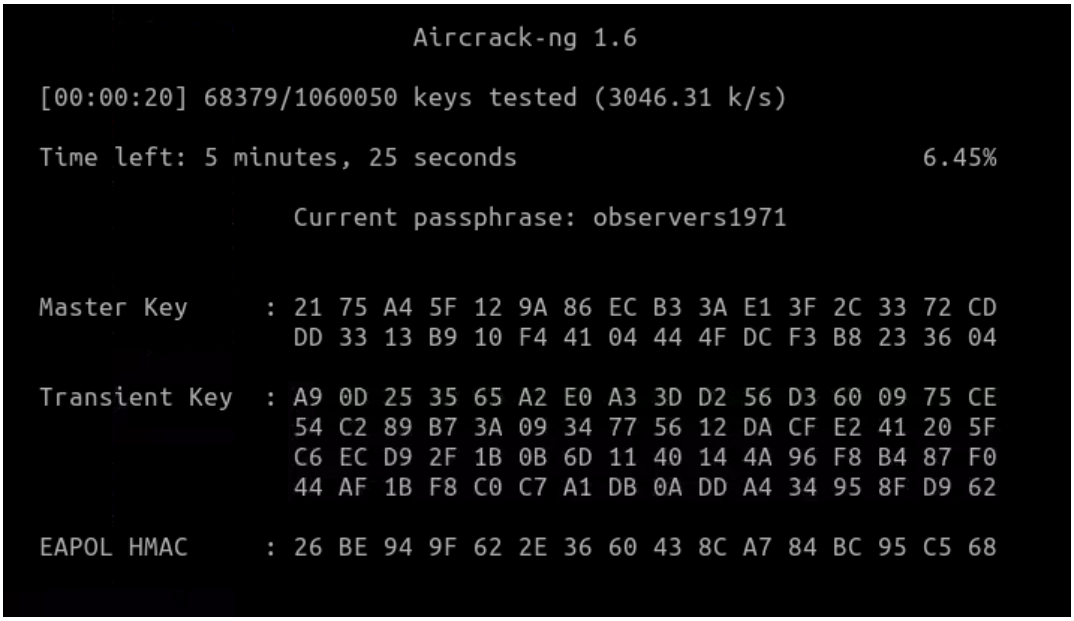
Note: John the Ripper is a password recovery and security auditing tool that is a staple in password cracking efforts. It also provides options for mangling wordlists through rules specified in the configuration file. The rule being used in this case is a simple one: each word should be combined with the years 1970-2000, effectively multiplying your list by 30. You can confirm that without the help of *wc* this time, since JtR provides the word (or password) count in the bottom left of its output (for example, 409p to indicate 409 passwords).

You are likely looking at a number in the 300K-400K range – about 30x larger than your previous list.

This is still a small wordlist, but it was founded on some pretty juicy information, and will only take a couple minutes to run.

21. **Make a screen capture** showing the **length of your new *yourname*_Capture.txt wordlist in the JtR output**.

22. At the command prompt, **type** `aircrack-ng -b 00:02:00:00:00:10 -w /root/yourname_Wordlist.txt /root/*.cap` to run aircrack-ng and your wordlist (`-w`) on all the capture files (`*.cap`) you gathered in airmon-ng.



Aircrack-ng

Note: Bingo, you have the goods. Turns out your research paid off, as this dictionary contained Rod's password. In the remaining steps, you will return to Wireshark to decrypt the capture you used to get that PSK.

23. **Make a screen capture** showing the **discovered passphrase in your aircrack output**.

24. In your QTerminal window, **click** the **wifi@TargetLinux:~/Topos tab** to return to your mininet-wifi> prompt.

25. At the mininet-wifi> prompt, **execute** `sh wireshark&` to start Wireshark from the TargetLinux01 console.

26. From the Wireshark menu, **select File > Open**, then **navigate** to **/home/wifi/sta1** and **open** the ***yourname*_Capture-01.cap file** to load the first capture file created from your airodump-ng into Wireshark.

27. From the Wireshark toolbar, **select Edit > Preferences**, then **navigate** to **Protocols > IEEE 802.11** and **edit** the **Decryption keys**.

28. **Add a new key**, using the **wpa-pwd option**, overriding the default wep selection.

Note: WPA2-PSK accepts an 8 to 63-character ASCII string (wpa-pwd, for password) or a 64-character hexadecimal string (wpa-psk), but really it only accepts a PSK. If you configure the AP with an ASCII string, as you have, the AP will just convert this value to a PSK on its end using a combination of that string, the SSID of the network, and some key stretching functions (key stretching is performing additional computations on a key to make it harder to guess/crack). Wireshark will also do this conversion for you, and so provides the option to use either the PWD or PSK.

Because that PSK is dependent, in part, on the SSID, it is common to provide the passphase in the following format:

```
Passphrase:networkssid
```

If you don’t, Wireshark is smart enough to search your capture for the SSID and add it for you automatically. However, it is good practice to always specify, especially when there are packets from numerous APs contained in the capture.

29. In the Key field, **type** `roswell1984:RodRadio` to supply Wireshark with the passphrase and the SSID of the WPA2-enabled AP, then **click OK** to complete your key addition.

30. **Click OK** to close the Preferences window and load your changes.

Note: Wireshark will take a few moments to load while it opens all these presents – ahem, packets — but once the blue loading bar at the bottom of the window disappears, your collection of packets should be decrypted – at least, some of them. You are probably staring at a screen of encrypted frames (which you could confirm by looking at the CCMP parameters) thinking it didn't work. And it might not have, if you did the steps wrong, but this isn't indicative of either yet. The thing is, these packets were captured *before* the handshake (which was triggered when you deauthenticated sta3, which then immediately started reauthenticating). Because the keys are derived from the handshake, you are unable to decrypt packets previous to the handshake you witnessed. You can confirm this by scrolling through the Packet List pane until you see Deauthentication in the Info column. An association request can be found somewhere in that flurry of deauth packets, and shortly thereafter you will start to see some color in the list.

In this case, you will be saved the trouble of searching for interesting traffic – you will instead apply a filter for ftp traffic to jump straight to the pay-off.

31. In the Wireshark display filter, **execute** `ftp` to only show information for the File Transfer Protocol in the Packet List pane.

Note: Hello! There are definitely filenames in there (check out the Info column), so what is Rod up to? The 10.0.0.2 host performed several STOR operations on host 10.0.0.3, which indicates FTP uploads. And take a look at the filenames: Blues Brothers and Desperate Housewives multimedia files? Rod is probably behind the 10.0.0.2 host and uploading to some media center he has dialed in there. He is probably using FTP (no encryption) for simplicity, figuring that his wireless connection is already encrypted, so what's the point of the additional complexity for something secure like SFTP (someone should tell Rod about the importance of a layered security approach!).

In any case, go ahead and grab the real gift. Locate the credentials used to log in to this FTP session. Looks like the username is starman, but what is the password? With both the username and password in your possession, Rod is basically pwned, so just document that secret string and log out – thanks for playing.

32. **Record** the **password discovered for the FTP user in your Wireshark packet capture**.

33. **Close** any **open windows**.

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.

Part 1: Mangle a Wordlist with John the Ripper

That mysterious gentleman, Rodney's so-called friend who originally hired you to break into Rod's network, is back with another request. Looks like Rodney got wise and decided to change his password, so the friend is asking you to take another crack at the network. You briefly consider the ethical qualms of breaking into a network without explicit and comprehensive written-and-signed permission from the target, but then you remember you are just a fictional character in a contrived scenario, whose author provides you the permission to assault the network of another of their creations – otherwise, you would never, ever do it, and this consoles you. So, you accept.

Your employer has also informed you that Rod has been removing most of his online presence, so he is unable to provide much in the way of personal information. What he does say is that he thinks Rod has been cooped up in his house for a while, alternating between that new spaceflight simulator game and that sweet MMORPG everyone plays. Hmm.

You could use the Wayback machine to try scooping up any additional data that Rod has since deleted, but you figure it might be easier than that. In fact, you figure Rodney probably didn't change his password all that much. When forced to change a password, most people (who don't use a password manager/generator) just add or remove a few characters, maybe rearrange several others, or make some simple character substitutions in (like '@' for 'a') to "munge" their password. Why not just mangle the original password a whole bunch of times and find out if Rod had the same idea?

Rod's profile was not very thick, but he had been interested in hacking for a period, and is apparently an avid gamer…. What are the chances he just created some leetspeak version? You know, that internet dialect that replaces characters with similar glyphs, used to indicate that you are of 1337 status in a particular game or domain of computer science? Worth a shot – time to cook up some el33t permutations. Well, after grabbing yourself a Kali Linux container, and creating a starting list with one word: roswell1984:

1. From the TargetLinux01 desktop, **open a QTerminal console.**

2. **Execute** `sudo ./Topos/Section3/kaliStart.sh` to launch your Kali Linux container (this script contains a single Docker command that starts the same Kali container as in seen in Section 2 Part 2)**.**

   Use `wifi` when prompted for the sudo password.

3. **Execute** `sudo docker attach mn.sta1` to attach to your Kali container.

4. **Execute** `echo "roswell1984" > rodRage_0.lst` to write Rod's previous password to a new file, serving as your initial wordlist.

Okay, it's show time, and you know just the tool for the job: John the Ripper. You also know that John the Ripper provides a mangling rule called L33t that could work nicely here. **Execute the john command** with the following specifications:

- Wordlist: **/root/rodRage_0.lst**
- Rules: **L33t**
- Write output to file: **rodRage_1.lst**

Great, but only ~400 words long? Not a very big list, this would take about a second to get through. You should probably at least make it longer to process than the time it takes to type the command…

JtR's "single" rule should do nicely. This rule contains an exhaustive set of rules within it that transform each password in many and varied ways. It will produce a few thousand iterations for each password, so with a short list of 400, you can easily justify that computational overhead.

One more time now: **Execute the john command** with the following specifications:

- Wordlist: **/root/rodRage_1.lst**
- Rules: **single**
- Write output to file: **rodRage_Final.lst**

Done, around 320k – still small, but arguably worth your time. You may also find a ton of duplicates here, but you can tune things up if this initial shotgun attempt doesn't get you in.

Make a screen capture showing the **output from your john command used to generate rodRage_Final.lst**.

Part 2: Perform a Dictionary Attack using a WPA2 Network Capture

After assembling your revised wordlist, you told your employer that you weren't going back to capture traffic on Rod's network in person, and he should just do it himself. And he did do it, which is great news. You made sure to instruct him that it was the EAPoL packets you were after, and to call it quits once he captured 4 of those. You didn't bother explaining deauthentication, but don't feel too bad about making him wait around for a handshake – he doesn't seem like the most savory character.

Okay, you have uploaded the new captures to your Kali machine at /root/S3/ and taken a peek inside to confirm the AP on Rod's network still has the same BSSID – confirmed. Now, get that passphrase by executing the aircrack-ng command with the following specifications:

- BSSID: **00:02:00:00:00:10**
- Wordlist: **/root/rodRage_Final.lst**
- Capture files: **/root/S3/*.cap**

Make a screen capture showing the **recovered WPA2 passphrase in your aircrack-ng output**.

Note: This concludes Section 3 of the lab.