Introduction

Device fingerprinting refers to the practice of uniquely identifying a device according to a combination of attributes, such as operating system, IP address, web browser, and more. It is commonly performed by network administrators seeking to document devices that are currently connected on a LAN or as part of network access control (NAC). However, it can also be used by advertisers and threat agents to track and monitor a person's online activities. Regardless of its positive or negative implications, it is essential to understand the process and the device properties that are examined during device fingerprinting.

While many of the same principles apply to fingerprinting both traditional PC's and mobile devices, mobile devices are accompanied by a unique set of challenges and opportunities. The value of a fingerprint lies in its ability to remain diverse and stable. The fingerprint must have diversity, meaning no two device fingerprints can be identical. Mobile devices — especially Apple's iPhone — often contain built-in configuration settings and apps that can hinder that diversity. Stability means that the fingerprint must be unique over a length of time. Since users have the ability to change certain device configurations or uninstall and reinstall apps, stability also can be challenging. Additionally, due to increased concern among mobile users about privacy, certain unique device characteristics are not exposed to third-party applications or web sites. Nonetheless, enough unique information can be obtained about a device during typical network communications to develop a fingerprint that is sufficiently diverse and stable.

There are two distinct types of device fingerprinting: proximity fingerprinting and online fingerprinting. Proximity fingerprinting is defined as identifying network devices that are located on the same network, typically conducted by administrators who wish to identify wireless devices on a network or to scan for unauthorized, rogue access points. Online fingerprinting is performed over the Internet, usually by e-commerce web sites, to give users a more contextualized experience. However, online fingerprinting is also used by marketing firms who collect data about large sets of users and sell this information to advertisers.

While the previously mentioned types focus more on the "where" in terms of the device being fingerprinted, there are also two different methods of fingerprinting – passive fingerprinting and active fingerprinting. Passive fingerprinting involves collecting data about a device without performing any intrusive queries. This can be done by analyzing TCP/IP headers, which contain data that is often unique to a particular device, operating system, or wireless environment. Active fingerprinting, on the other hand, uses more aggressive tactics to directly query the device. Protocols such as SNMP or scripting languages like JavaScript can be used to interrogate a device to learn about a wide array of device and system properties to create a fingerprint that is unique and stable.

In this lab, you will explore each method and type of device fingerprinting. You will use Wireshark to perform passive proximity fingerprinting by collecting data within TCP/IP and HTTP headers. Next, you will explore Nmap to see how active proximity scanning is more powerful for collecting unique device characteristics. In Section 2 of the lab, you will conduct passive proximity fingerprinting using p0f and work with JavaScript to conduct active online fingerprinting. Finally, you will see how disabling browser cookies can prevent certain types of online fingerprinting, and learn how resources like Cover Your Tracks can provide insight into what information a browser might reveal about a user when surfing the web.

Lab Overview

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

1. In the first part of the lab, you will use Wireshark perform passive fingerprinting of an Android mobile device.

2. In the second part of the lab, you will use Nmap to perform active fingerprinting of an Android device.

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 use p0f and ClientJS to perform advanced analysis of data collected from a mobile device.

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 proximity fingerprinting techniques to identify a mobile device on a network.

2. Use online fingerprinting to identify a mobile device over the Internet.

3. Use passive fingerprinting to identify a mobile device without querying the device itself.

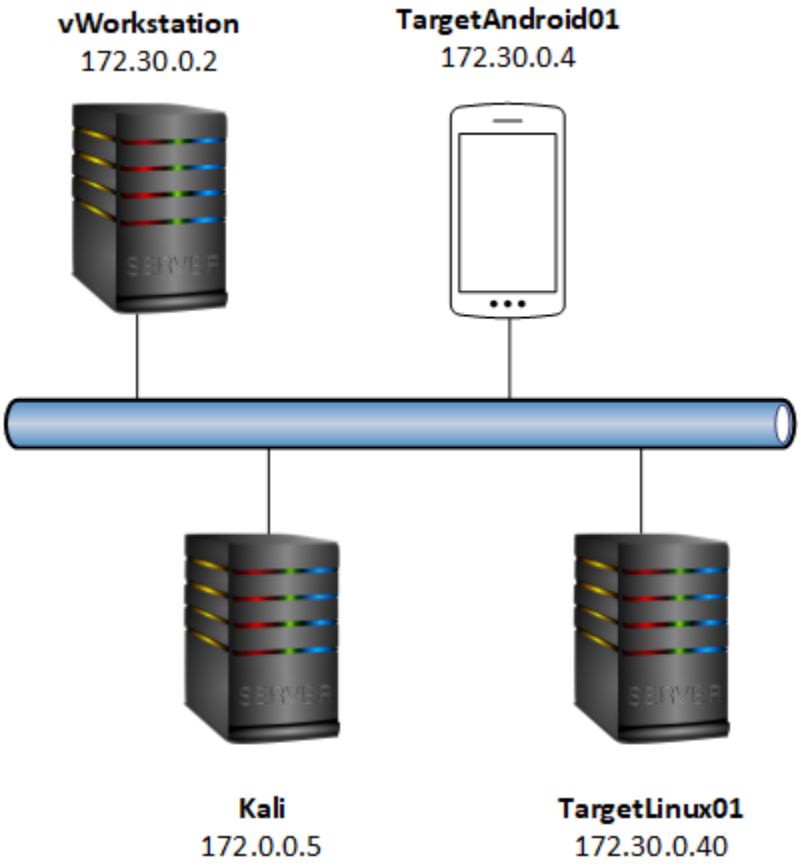
4. Use active fingerprinting to identify a mobile device via direct interactions.

5. Identify and mitigate vulnerabilities that allow aggressive, online fingerprinting.

Topology

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

- vWorkstation (Windows: Server 2019)
- TargetAndroid01 (Android)
- Kali Linux (Linux: Kali)
- TargetLinux01 (Linux: Ubuntu)



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.

- Wireshark
- Nmap
- p0f
- ClientJS

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:

- Time to live field
- User-Agent string for the Chrome browser session
- User-Agent string for the Firefox browser session
- User-Agent string for the Edge browser session from the vWorkstation
- Results of the default Nmap scan
- Results of the -O -v Nmap scan
- Product name in the Nmap output

2. Any additional information as directed by the lab:

- Compare the three User-Agent strings. How do they differ? How are they similar?

SECTION 2

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

- p0f result identifying 172.30.0.4 as Linux (Android)
- p0f result identifying 172.30.0.4 as Android 9.x
- Traffic generated by Nmap in Wireshark
- Chrome fingerprint on TargetAndroid01
- Firefox fingerprint on TargetAndroid01
- Updated Firefox fingerprint on TargetAndroid01
- Chrome fingerprint on the vWorkstation

2. Any additional information as directed by the lab:

- Compare the Wireshark results from the p0f scan to the nmap scan.
- Identify the differences between the Datapoints values generated for Chrome and the Datapoints values generated for Firefox.
- Identify the differences between the Datapoints values for Firefox before and after the update.
- Identify the differences between the Datapoints values on the vWorkstation and TargetAndroid01.

SECTION 3

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

- Cookie settings in Chrome on TargetAndroid01
- Cover Your Tracks assessment results

2. Any additional information as directed by the lab:

- What would happen if you switched to a different option for allowing versus blocking cookies, and how would it hinder attempts to track your activities online? Would there be any

drawbacks?

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: Perform Passive Fingerprinting with Wireshark

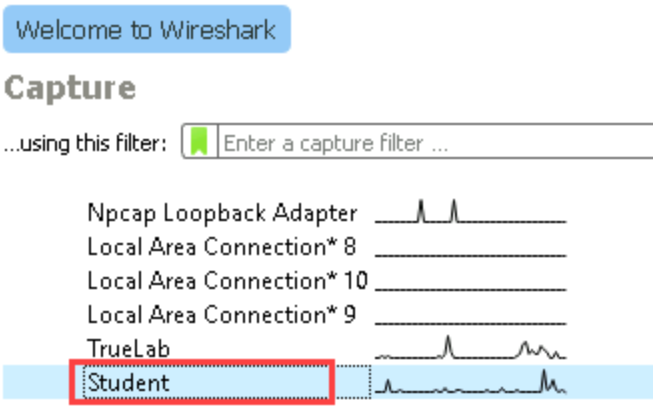
Note: When a client initiates communication with a server, it produces specific values within TCP/IP and HTTP headers that can be highly valuable when fingerprinting a device. One of the simplest ways to capture this information is with Wireshark, an open-source network utility that is widely recognized as one of the most versatile traffic analysis tools in the industry. Wireshark can run on both wired and wireless networks, although additional configuration is required in order to capture traffic in wireless environments. This lab activity will focus on the information that can be captured at the OSI Network Layer and higher, so there is no need for analysis of Data Link Layer wireless frames or Physical Layer radio signals.

In the next steps, you will use Wireshark to capture and analyze network traffic between an Android mobile device and a web server. You will inspect the traffic that Wireshark captures and analyze specific fields, including the IP protocol's TTL value and the HTTP protocol's User-Agent string. For comparison, you will also use a Windows device and different browsers to connect to the web server and review the information generated by each device.

1. On the vWorkstation desktop, **double-click** the **Wireshark icon** to open the Wireshark application.

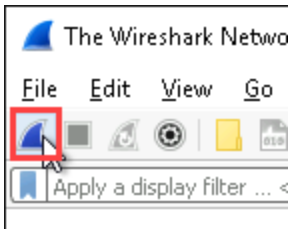
Wireshark icon

2. In the Capture section of the Wireshark window, **click** the **Student interface** to select it as your capture interface.



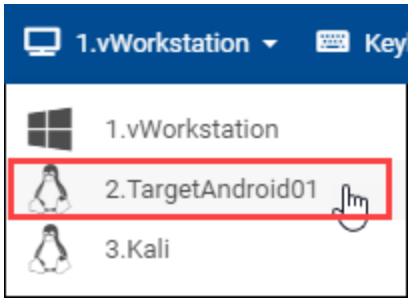
Student interface

3. On the Wireshark toolbar, **click** the **Start Capture button (the blue shark fin)** to begin capturing packets on the Student interface.



Start Capture

4. On the Lab View toolbar, **select TargetAndroid01** from the Virtual Machine menu to connect to the Android virtual machine.

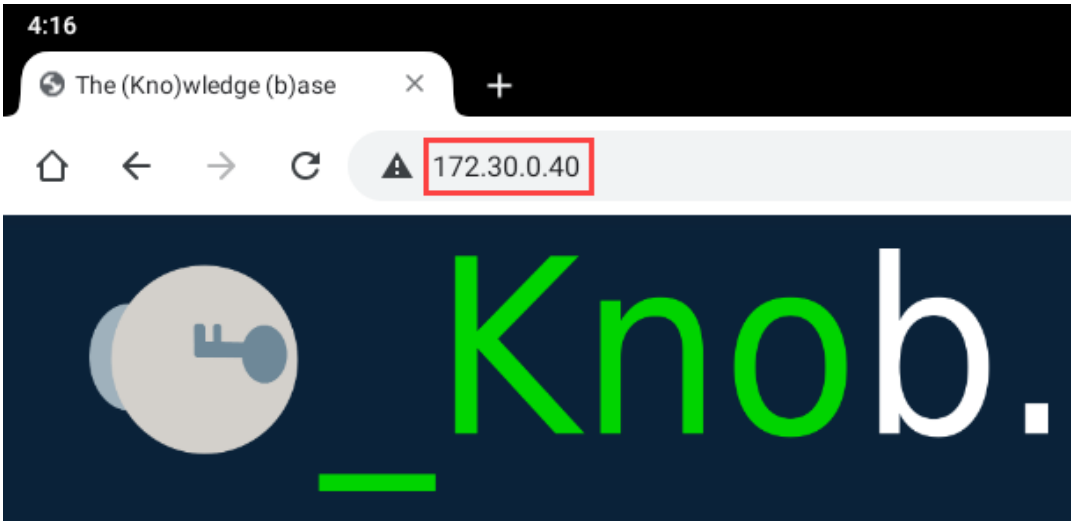


Virtual Machine menu

5. At the Apps screen, **click** the **Chrome icon** to open the Chrome app.

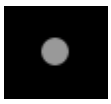
Chrome icon

6. In the browser navigation bar, **type** `172.30.0.40` and **press Enter** to connect to the web server on TargetLinux01.



Chrome navigation bar

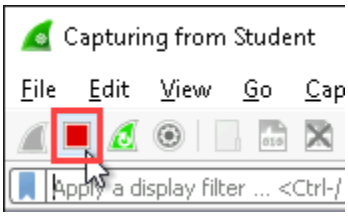
7. At the bottom of the screen, **click** the **Circle icon** to return to the home screen.



Circle icon

8. On the Lab View toolbar, **select vWorkstation** from the Virtual Machine menu to return to the vWorkstation.

9. On the Wireshark toolbar, **click** the **Stop Capture button** (the red stop sign) to stop capturing packets on the Student interface.



Stop Capture

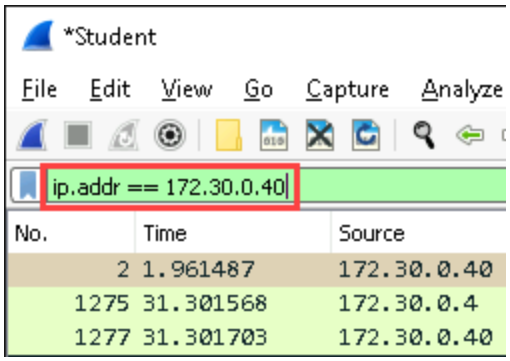
Note: The Wireshark capture interface contains three sections:

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

- Packet Bytes View (bottom): this view shows the raw data in contained 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 (.).

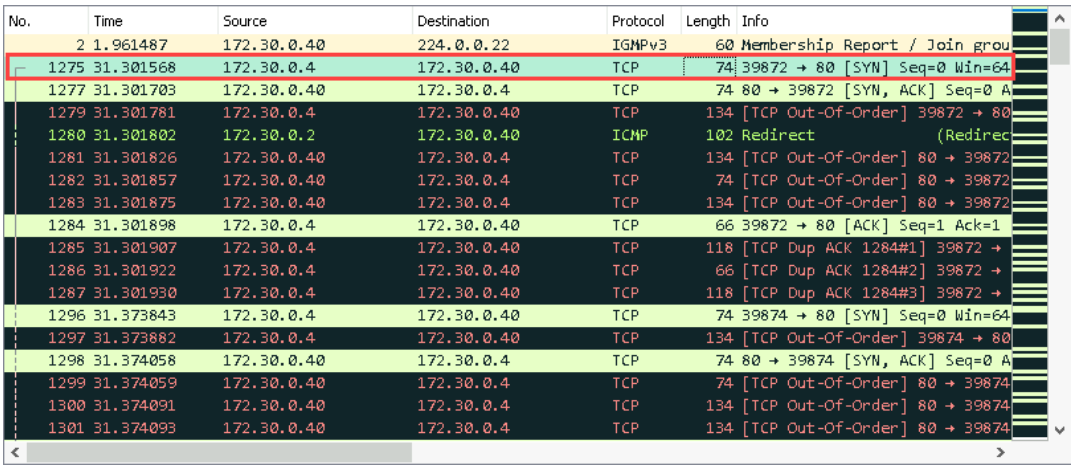
10. On the Wireshark toolbar, **type** `ip.addr == 172.30.0.40` in the Display Filter field and **press Enter** to display only traffic to and from the web server you connected to earlier.



Display Filter field

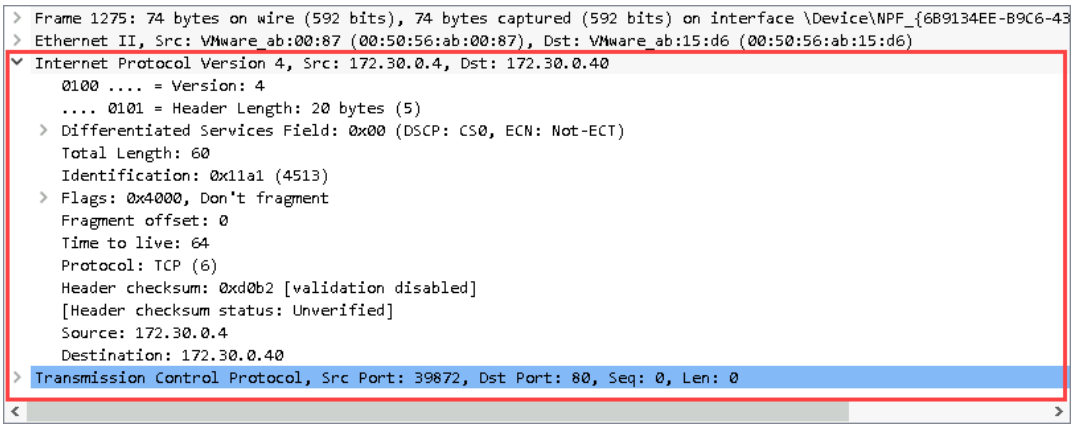
Note: One of the first packets that appears with the filter applied should be a TCP SYN request from the Android mobile device (source IP address 172.30.0.4) that viewed the web page. TCP SYN is the first part of the three-way handshake that occurs before data is transmitted between a client and server.

11. In the Packet List View, **select** the **packet** with a **Source IP address of 172.30.0.4** and a **SYN flag** in the Info column.



SYN packet

12. In the Packet Details View, **expand** the **Internet Protocol Version 4 header** to display the IPv4 packet details for the currently selected packet.



Internet Protocol Version 4 header

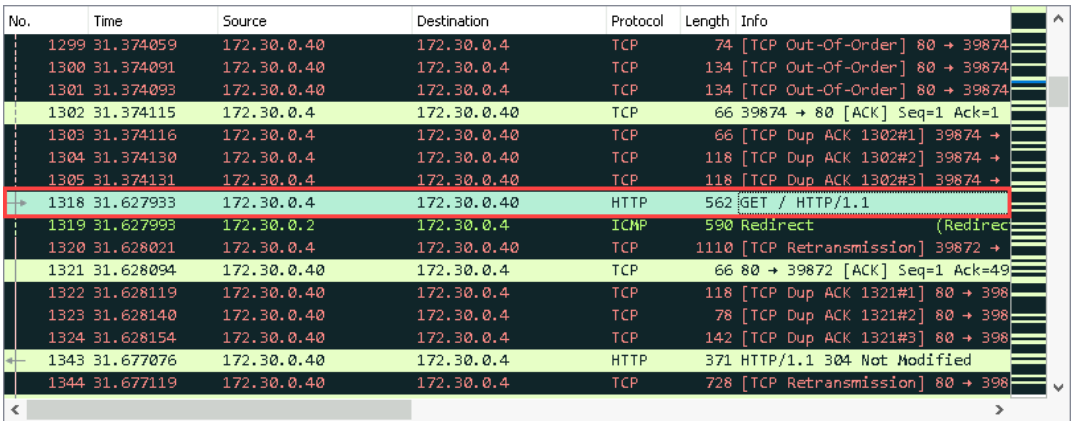
13. Within the IPv4 packet details, **identify** the **Time to live field**.

Note: Time to live, or TTL, is a value used to limit a packet's lifetime within an IPv4 network. Each operating system has its own default TTL value. The default TTL value for Windows systems is 128, whereas the default for Linux (and Android) is 64. While this value can be changed, it is used here to show how a comparison and determination can be made when developing a device fingerprint.

14. **Make a screen capture** showing the **Time to live field**.

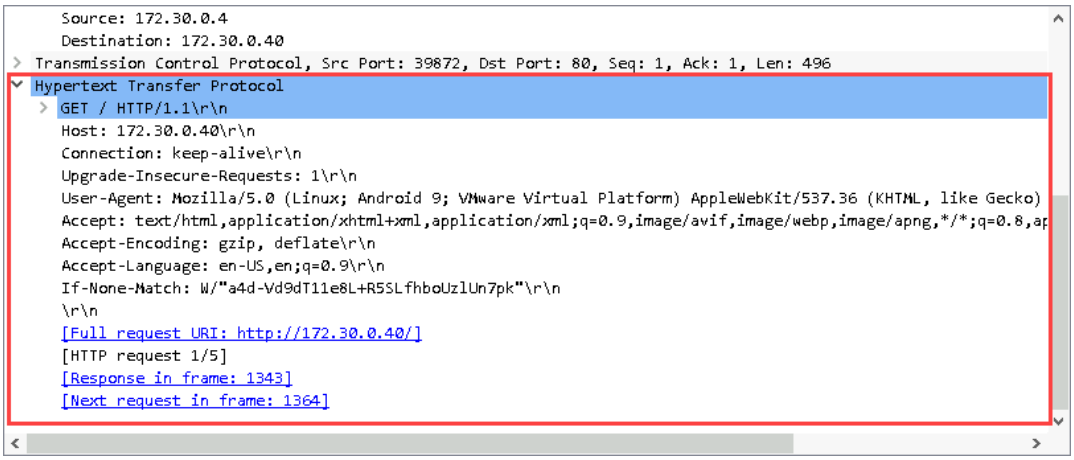
Note: In the next steps, you will examine specific data within the HTTP protocol that is used in developing a device fingerprint. HTTP uses a series of messages between a client and server to request and deliver content. The first message sent from the client is known as HTTP GET. This message contains a field known as the User-Agent string, which contains information about the client operating system. You will use Wireshark to examine this data and develop a more unique device fingerprint.

15. In the Packet List View, **select** the **packet** with a **Source IP address of 172.30.0.4** and a **GET flag** in the Info column.



GET packet

16. In the Packet Details View, **expand** the **Hypertext Transfer Protocol header** to display the HTTP packet details for the currently selected packet.



Hypertext Transfer Protocol header

17. Within the HTTP packet details, **identify** the **User-Agent field**.

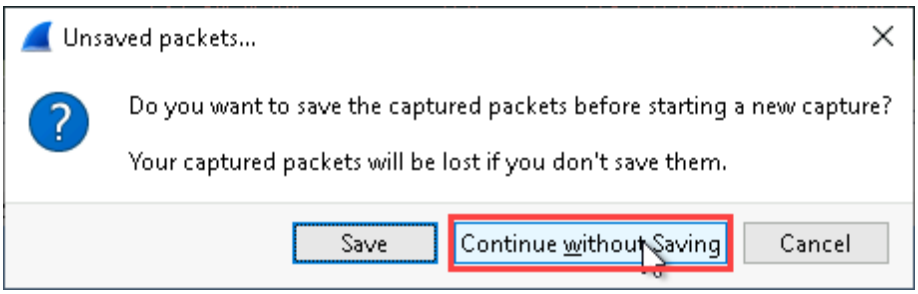
Note: The User-Agent string contains several fields that can be useful in developing a device fingerprint. It was originally used so web site developers could deliver content to devices based on whether it is a desktop, laptop, phone, etc. However, recent industry trends have moved away from this method and the User-Agent string has become deprecated. Another drawback to the User-Agent string is the fact that some of the properties can be changed or spoofed. The information developed for a device fingerprint might be considered more of an “educated guess.”
Certain elements of the User-Agent string are common among many devices. References to Mozilla and Gecko are listed when the client device is using the Firefox web browser. Google Chrome and Microsoft Edge will generate different data within these fields. However, depending on the web browser and device that is used, some information can reveal more distinct characteristics about the device, including the hardware platform and if it is, in fact, a mobile device.

In the next steps, you will test how different devices and different browsers generate unique signatures within the User-Agent string.

18. **Make a screen capture** showing the **User-Agent string for the Chrome browser session**.

19. On the Wireshark toolbar, **click** the **Start Capture button** to start a new capture session.

20. When prompted, **click Continue without saving** to discard your first packet capture.



Unsaved packets dialog box

21. **Repeat steps 4-9** using the Firefox app.

To access the Firefox app, **position your cursor** in the middle of the TargetAndroid01 home screen, then **click and drag up** to simulate the 'swipe up' action and open the apps screen.

22. **Repeat steps 15-17** to identify the User-Agent for the Firefox browser session.

23. **Make a screen capture** showing the **User-Agent string for the Firefox browser session**.

24. **Repeat steps 19-20** to start a new packet capture session.

25. On the vWorkstation taskbar, **click** the **Edge icon** to open a new browser window.

Edge icon

26. In the browser navigation bar, **type** `172.30.0.40` and **press Enter** to connect to the web server on TargetLinux01.

27. **Close** the **Edge window**.

28. **Repeat steps 15-17** to identify the User-Agent string for the Edge browser session on the vWorkstation (IP address 172.30.0.2).

29. **Make a screen capture** showing the **User-Agent string for the Edge browser session from the vWorkstation**.

30. **Compare** the three User-Agent strings. How do they differ? How are they similar?

31. **Close** the **Wireshark window**.

Part 2: Perform Active Fingerprinting with Nmap

Note: In the previous activity, you learned how network analysis tools like Wireshark can be used to perform passive device fingerprinting. While passive fingerprinting is useful for collecting data that is part of a normal client-server communication, additional information is sometimes necessary to develop a fingerprint that is both unique and stable. Active fingerprinting, a process that is more aggressive, allows administrators and others to send specific types of network communications designed to generate responses from a device that contain tell-tale signatures regarding its operating system, version, network service configurations, and more.

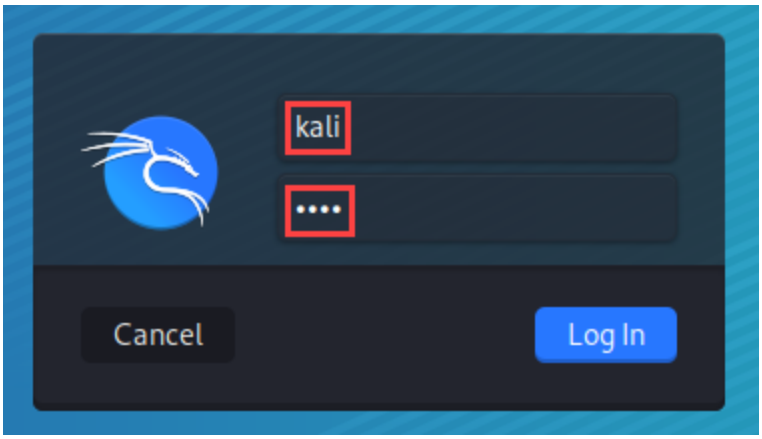
In the next steps, you will use Nmap, another popular open-source network utility, to aggressively scan and query a device. While Nmap is a very useful tool, you should always use extreme caution when running it on a production network. Nmap is a versatile tool with many features that could be considered a violation of a company's acceptable use policy. Be sure to always obtain express, written permission from any organization where you plan on running Nmap prior to conducting any scans.

1. On the Lab View toolbar, **select Kali** from the Virtual Machine menu to connect to the Kali virtual machine.

Note: Kali is a popular Linux distribution used by administrators, security professionals and, yes, even hackers, to perform a wide variety of network tasks. As with Nmap, you should only use the tools in Kali Linux on a production network when you have obtained written permission to do so.

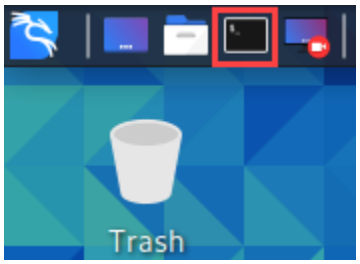
2. At the Kali log-in screen, **type** the following credentials and **press Enter** to log in.

   Username: **`kali`**
   Password: **`kali`**



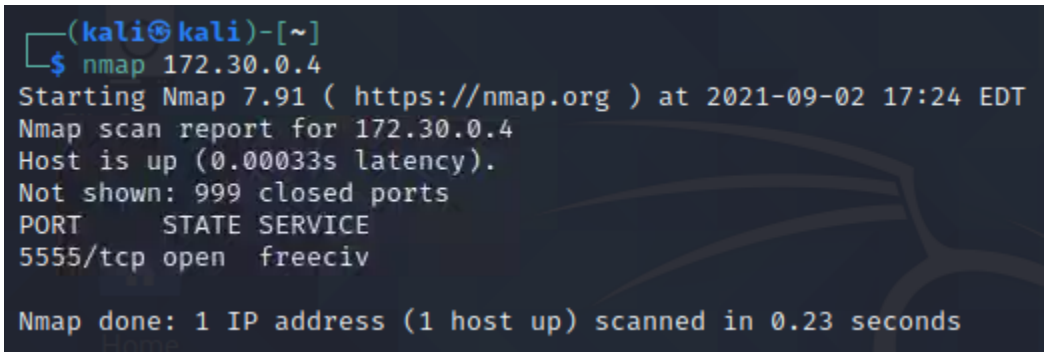
Kali log-in screen

3. On the Kali menu bar, **click** the **Terminal icon** to open a new Terminal window.



Terminal icon

4. At the command prompt, **type** `nmap 172.30.0.4` and **press Enter** to run a default Nmap scan of the TargetAndroid01 device.



Default Nmap scan

Note: The central component of the scan that Nmap conducted is called a port scan. A default Nmap scan will probe the device for the top 1000 TCP ports and attempt a DNS resolution. Based on the results, an Nmap user can potentially draw conclusions about the device and its purpose. For example, a dedicated server running Microsoft SQL Server requires the use of TCP ports 1433, 4022, 135, and/or 1434. Nmap also contains a database of common port uses, and will attempt to identify the service associated with the port.

In this case, the results of this scan indicate that TCP port 5555 is open, which Nmap associates with the freeciv service. This is actually an error in the Nmap database. Freeciv is an open-source clone of the Civilization game series that uses a default listening TCP port of 5556. In reality, TCP port 5555 is associated with the Android Debug Bridge (ADB) service, which is a command-line developer tool used to communicate with Android devices. This very simple scan has already provided useful information for identifying the device.

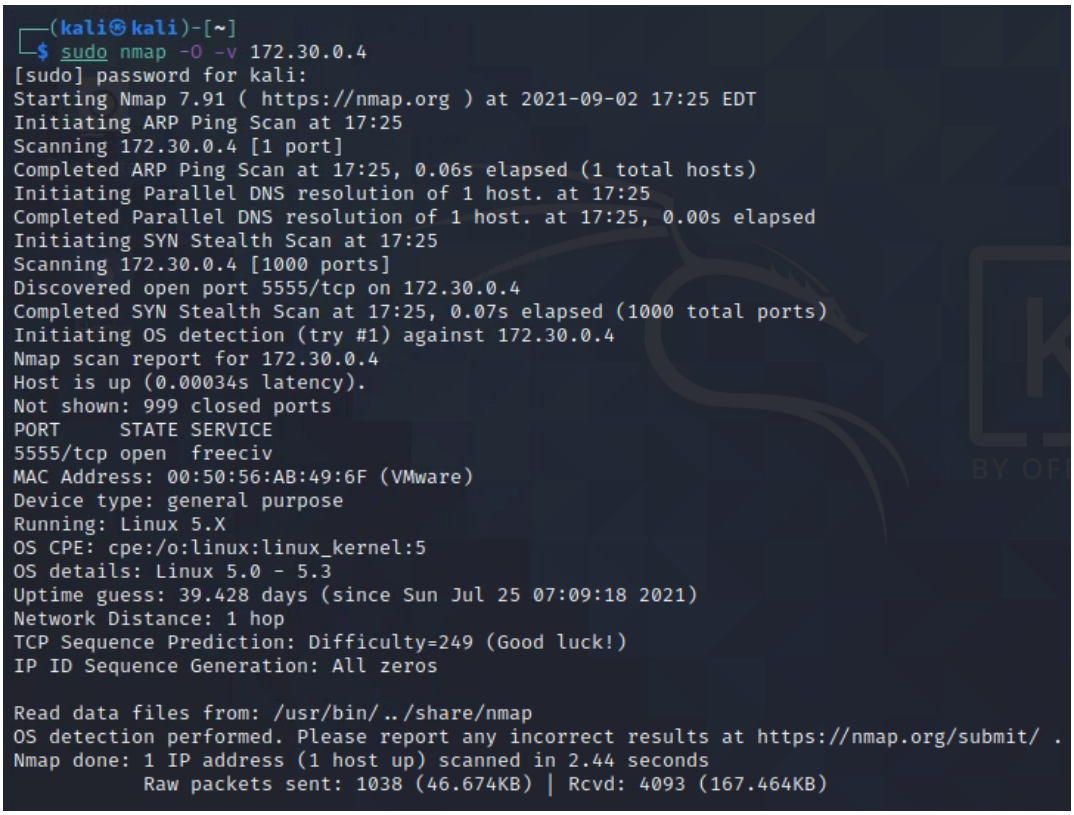
Of course, what you currently see on-screen is just a fraction of Nmap's scanning and output capabilities. To retrieve additional information about the target device, you can add the **`-v`** switch to the previous command to display “verbose” output, which you will do in your next scan.

5. **Make a screen capture** showing the **results of the default Nmap scan**.

Note: There are numerous command switches that can be used with Nmap to probe a device. They can be useful in discovering information about the operating system, any running services, open TCP and UDP ports, and more. In the next steps, you will work with some of these switches to obtain more information about the device. Some Nmap commands require elevated privileges. To do this, you will use the Linux **sudo** command.

6. At the command prompt, **type** `sudo nmap -O -v 172.30.0.4` and **press Enter** to run another Nmap scan of the Android device.

   When prompted, **type** `kali` to authenticate the privilege escalation.



-O -v Nmap scan

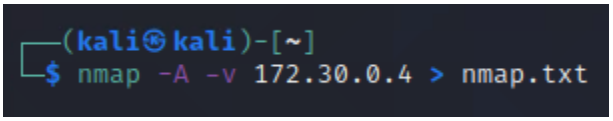
Note: The **`-O`** switch instructs Nmap to attempt to enumerate the device operating system. Note the additional information on the lines for “Running,” “OS CPE” and “OS details.” All three indicate the remote device is using Linux. CPE, which stands for Common Platform Enumeration, is used to identify systems and software within an IT environment.

7. **Make a screen capture** showing the **results of the -O -v Nmap scan**.

Note: In the next steps, you will use the **`-A`** switch, which attempts to enumerate operating system as well as version information. You will also use the **`cat`** command to output the results to a file.

8. At the command prompt, **type** `nmap -A -v 172.30.0.4 > nmap.txt` and **press Enter** to run another Nmap scan of the Android device and save the output to a text file.

   This scan will take 1-2 minutes to complete.

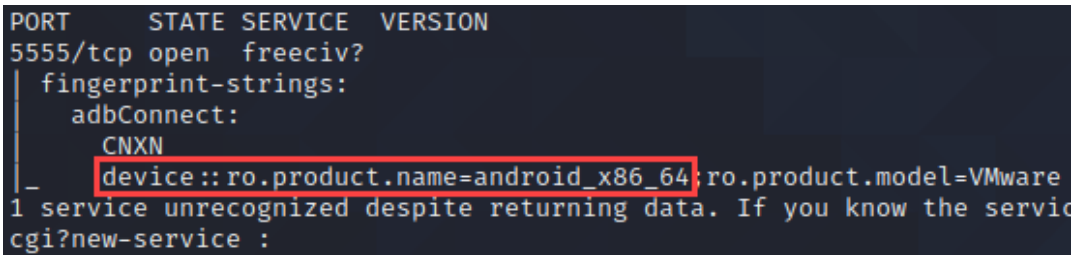


-A -v Nmap scan

9. At the command prompt, **type** `cat nmap.txt | less` and **press Enter** to open the text file you created in the previous step.

Display the nmap output

10. If necessary, **press Enter** to step through the output one line at a time until you locate the *device* row.



Device name

Note: As previously mentioned, Nmap scans the top 1,000 TCP ports to see if they are open (unless instructed otherwise). Once again, the result of this scan shows that port 5555 is open, but this time it is followed by a fingerprint-string of adbConnect, which is associated with the Android Debug Bridge (ADB). Within the same field, you should also see a row titled *device* that contains the product.name value "android_x86_64." This additional information confirms that the device is not just running Linux, but running Android.

11. **Make a screen capture** showing the **product name in the Nmap output**.

Note: Due to limitations of virtualizing Android, certain attributes that can be used to identify a mobile device are not available for this lab. These include key attributes at the Physical Layer, such as radio signal frequencies, and Data Link Layer information available through 802.11 framing characteristics. While these elements can prove crucial in developing a mobile device fingerprint, the steps completed in this lab still provide the foundations for how this data can be collected and examined in the real world.

12. **Press q** to return to the command prompt.

13. **Close** the **Terminal window**.

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 use p0f and ClientJS to perform advanced analysis of data collected from a mobile device.

Part 1: Perform Passive Fingerprinting with p0f

Note: In the previous section, you used Wireshark and Nmap to conduct passive and active fingerprinting of a mobile device. While Wireshark and Nmap are both widely recognized for their power and versatility, there are occasions when more specialized tools are necessary. For example, while Wireshark is an excellent general-purpose tool for packet capture and analysis, it is not explicitly designed for fingerprinting and requires a non-trivial degree of effort to surface the relevant data points for uniquely identifying a specific device. Meanwhile, as an active fingerprinting tool, Nmap generates spoofed TCP messages (SYN-ACK, FIN, XMAS scan, etc.) that may be blocked by a firewall or prompt alerts in an Intrusion Detection System (IDS).

In situations where fingerprinting must be performed efficiently, but passively, you might consider using p0f. P0f (short for Passive Operating System Fingerprinting) is an open-source tool that is included with Kali Linux. As its full name implies, p0f passively intercepts TCP messages that occur during typical, day-to-day network communications, then analyzes the intercepted traffic to attempt to identify the operating system of each host involved in the communications. In practice, p0f's output resembles the information you gathered using active fingerprinting techniques with Nmap, but developed using the same unsolicited inputs as Wireshark.

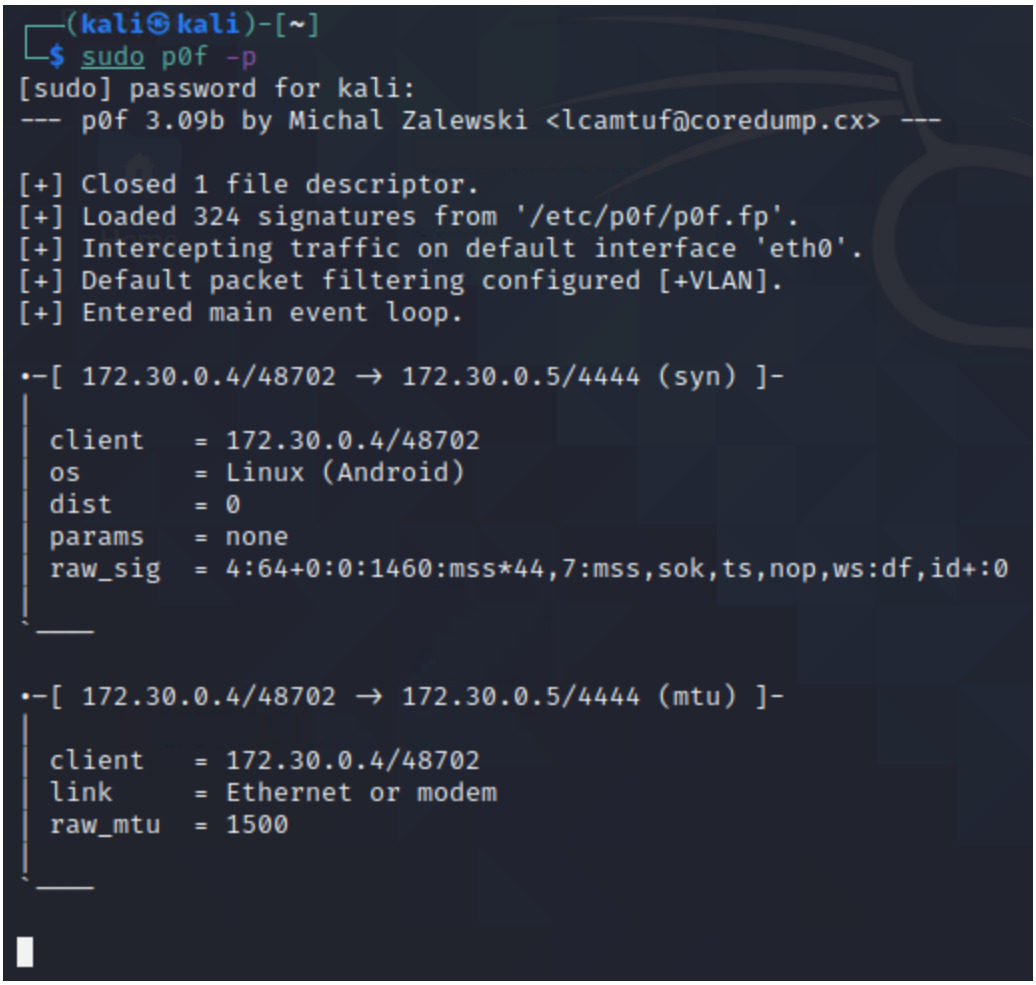
In the next steps, you will activate p0f on the Kali system, then generate traffic for it to analyze.

1. Connect to the **Kali** virtual machine and **log in** with the username `kali` and password `kali`.

2. From the Kali menu bar, **open** a new **Terminal window**.

3. At the command prompt, **execute** `sudo p0f -p` to launch the p0f utility.

   When prompted, **enter** `kali` to authenticate the privilege escalation.



p0f scan

Note: The p0f utility is now running. P0f will examine all intercepted network transmissions and display them in the terminal. Each message captured contains the source and destination IP addresses, port numbers, and other specific data contained in network headers (IP, TCP, HTTP, MTU, etc.), which p0f will use to attempt to identify the device model, connection type, operating system, and more.

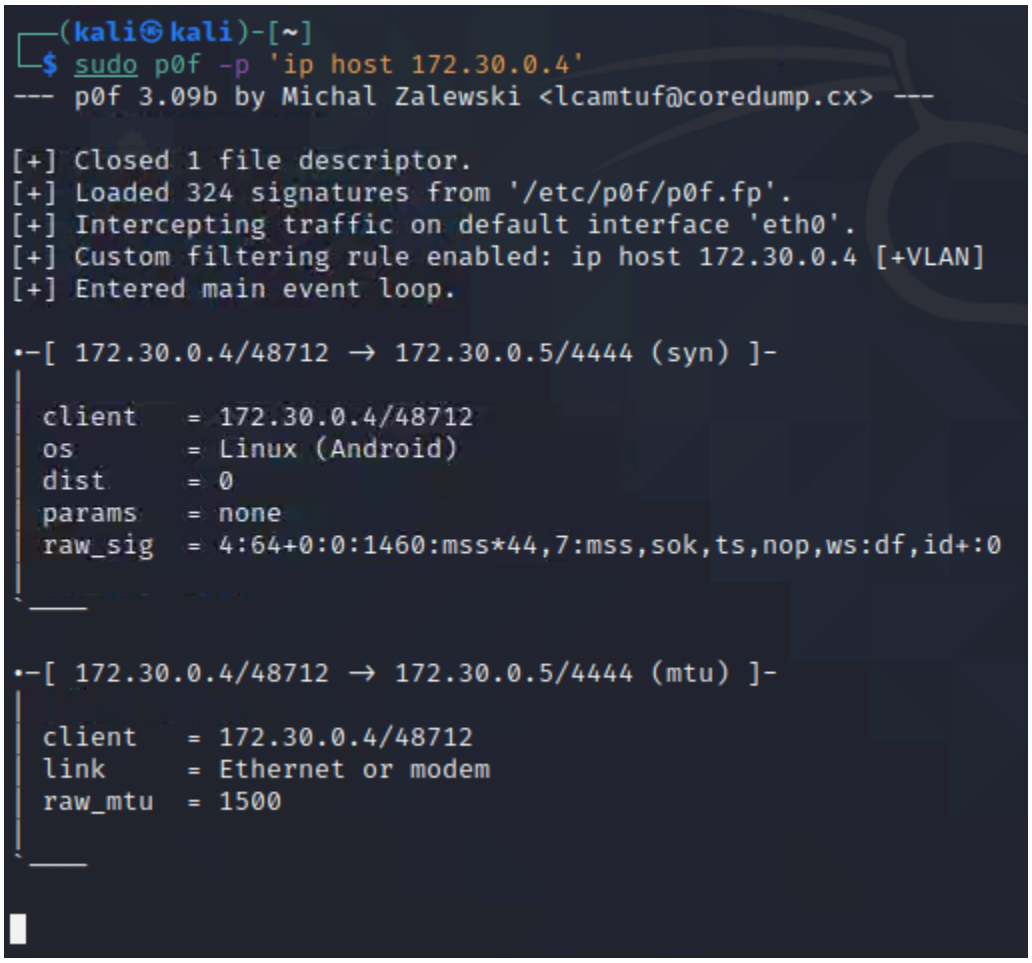
The **-p** switch instructs p0f to run in promiscuous mode. In a typical wired LAN environment that uses a switch, a machine running a packet sniffer like p0f in promiscuous mode will be able to inspect all broadcast and multicast traffic on the local network segment, in addition to the traffic directed to or from the machine itself. In a wired LAN environment that uses a hub, a machine running a packet sniffer in promiscuous mode will be able to inspect all traffic. Similarly, in a wireless LAN environment, a machine running a packet sniffer in promiscuous mode will be able to inspect all network traffic. In this lab environment, p0f will capture all traffic generated within the four-host network segment.

In the next steps, you will refine your p0f scan to collect only traffic related to the TargetAndroid01 device.

4. **Press Ctrl+c** to end the p0f session.

5. At the command prompt, **execute** `sudo p0f -p 'ip host 172.30.0.4'` to launch the p0f utility in promiscuous mode, limiting the capture to traffic to and from the TargetAndroid01 device.

   If you do not see any results after two minutes, reset the lab (Options > Reset Lab) and try again.



Targeted p0f scan

Note: Among the results, you should see a SYN message from source IP (client) 172.30.0.4 indicating

the OS is Linux (Android). The raw_sig field contains the values that were used by p0f to make this 'educated guess' about the device type. These include Maximum Transmission Unit (MTU), Maximum Segment Size (MSS), Time To Live (TTL), and more. Different operating systems use different values for each of these, allowing p0f to make its determination.

6. **Press Ctrl+c** to end the p0f session.

7. **Make a screen capture** showing the **p0f result identifying 172.30.0.4 as Linux (Android)**.

Note: In the next steps, you will generate HTTP traffic between the TargetAndroid01 device and the TargetLinux01 web server, then examine any additional information that p0f captures to develop a fingerprint.

8. At the command prompt, **execute the command** to start a new p0f capture in promiscuous mode.

9. From the Lab View toolbar, **connect** to the **TargetAndroid01 device**.

10. On the TargetAndroid01 device, **open Chrome** and **navigate** to **`172.30.0.40`**.

11. **Return** to the **TargetAndroid01 home screen**.

12. From the Lab View toolbar, **connect** to the **Kali system**.

13. **Press Ctrl+c** to end the p0f capture.

Note: Among the results, you should see a http request message from source IP address 172.30.0.4. Fields within the HTTP Request header have been used to develop a more distinct fingerprint of the device. It now indicates in the "app" field that it is an Android 9.x device. You should also note the raw_sig field again, which contains the values that were used in developing the signature, such as User-Agent string, browser properties, and more.

14. **Make a screen capture** showing the **p0f result identifying 172.30.0.4 as Android 9.x**.

Note: In the next steps, you will compare the differences between active fingerprinting tools and passive fingerprinting tools by running a Wireshark capture while simultaneously running Nmap and p0f scans.

15. At the command prompt, **execute** `sudo wireshark &` to open the Wireshark application with superuser privileges.

16. **Start a capture session** on the eth0 interface.

17. At the command prompt, **execute the command** to run an Nmap scan of the Android device and enumerate its operating system and version.

18. In the Wireshark window, **stop** the **capture session**, then **apply** a **display filter** to show only traffic related to the TargetAndroid01 device.

Note: If you scroll to the bottom of the Packet List View, you will see the total number of captured packets in the Packet Display View exceeds 2000. As in the previous section, Nmap was able to successfully fingerprint the device, but as you can see from the Wireshark capture, an aggressive scan like this generates a great deal of traffic that is likely to be blocked by a company firewall or even raise an alarm on an IDPS.

19. **Make a screen capture** showing the **traffic generated by Nmap in Wireshark**.

Note: In the next steps, you will run another Wireshark capture while running p0f for comparison. Because p0f does not generate traffic itself, you will once again generate an HTTP request on the TargetAndroid01 device.

20. **Start** a new **Wireshark capture**, discarding the first packet capture.

21. At the command prompt, **execute the command** to launch the p0f utility in promiscuous mode, limiting the capture to traffic to and from the TargetAndroid01 device.

22. **Repeat steps 9-12** to generate HTTP traffic from the TargetAndroid01 device.

23. On the Kali system, **stop** both the **p0f scan** and the **Wireshark capture**.

24. **Compare** the Wireshark results from the p0f scan to the Nmap scan.

Part 2: Perform Active Fingerprinting with ClientJS

Note: In Section 1 of the lab, you used Nmap to perform active fingerprinting on the local network. While highly effective for this purpose, Nmap is typically used for proximity fingerprinting, where the target device and fingerprinting device are both connected to the same network. In this part of the lab, you will perform a different type of fingerprinting - online fingerprinting.

To perform fingerprinting over the Internet, code such as JavaScript can be embedded in a web page to gather information from devices that establish a connection with the server. There are several uses for online fingerprinting, not all of which are popular, given concerns over privacy of user data and Internet browsing habits. One positive application is for a website to create a more convenient experience for users who are shopping online. For example, knowing the individual allows the site to recommend products or services based on previous shopping history. Another less popular example would be an online marketer who wishes to use a visitor’s fingerprint to target them with third-party advertisements that may not be desired. Users may attempt to thwart this form of aggressive targeting by switching between different devices, using different browsers, deleting browser cookies, or even reinstalling an app that was used previously. These efforts can diminish the fingerprint’s stability, minimizing its overall value to the website that is attempting to identify and track users.

In the next steps, you will see how one specific tool – ClientJS – can be used to develop an online fingerprint. ClientJS not only generates a unique, 32-bit hash for each device, it also provides detailed output of each data point that was used to develop the fingerprint. To demonstrate this, you will use the Android device to visit a web server with ClientJS installed and review the fingerprint generated for your device. You will then use different browser configurations on the same device to see how they change the 32-bit value that is generated. Finally, you will use a different device to visit the same web page and observe how the fingerprint changes.

1. From the Lab View toolbar, **connect** to the **TargetAndroid01 device**.

2. On the TargetAndroid01 device, **open Chrome** and **navigate** to **`172.30.0.40`** to open the Knob website, then **click** the **Login link**.

Note: On the Login page, you will see a 10-character fingerprint value. This is the unique ID generated by ClientJS for the purpose of tracking your device. Ordinarily, this information is not outwardly displayed to the user in this manner, but in this case it has been incorporated into the webpage for demonstration purposes.

3. On the Login page, **expand** the **Datapoints header** to display the data that was used to generate the fingerprint.

Expanded Datapoints header

Note: You should notice some similarities in the data collected here by ClientJS and what was used

by both Nmap and p0f. For example, you will see that the User-Agent string data is displayed and that it lists Android version 9, along with browser configurations, fonts, and more.

4. **Make a screen capture** showing the **Chrome fingerprint on TargetAndroid01**.

Note: It is also worth mentioning that some of this data is not entirely accurate, due to the restrictions of running a mobile device in a virtualized environment. A device running 'in the wild' would have generated many more data points to develop a fingerprint that is diverse and stable.

5. **Return** to the **TargetAndroid01 home screen**.

6. At the TargetAndroid01 home screen, **position your cursor** in the middle of the display's background, then **click and drag up** to simulate the 'swipe up' action and open the apps screen.

7. At the Apps screen, **click** the **Firefox icon** to open the Firefox app.

8. **Repeat steps 2-3** in Firefox.

Note: You should quickly notice that the fingerprint value is different from the one you observed on the same device using Chrome. Even minor changes to the values documented under the Datapoints header will result in a completely different unique identifier.

9. **Make a screen capture** showing the **Firefox fingerprint on TargetAndroid01**.

10. **Identify** the differences between the Datapoints values generated for Chrome and the Datapoints values generated for Firefox.

Note: In the next steps, you will update the Firefox app to the latest version and observe how this update impacts the fingerprint value that you just documented.

11. **Return** to the **TargetAndroid01 home screen**.

12. On the TargetAndroid01 home screen, **click** the **Google Play Store icon** to open the Google Play Store app.

13. In the Google Play Store app, **search** for the **Firefox** app, then **click** the **Update button** to update Firefox to the latest version.

14. **Return** to the **TargetAndroid01 home screen**.

15. **Repeat steps 6-8** using the updated version of Firefox.

16. **Make a screen capture** showing the **updated Firefox fingerprint on TargetAndroid01**.

17. **Identify** the differences between the Datapoints values for Firefox before and after the update.

Note: In the final steps, you will use Chrome on the vWorkstation to visit the same website and observe the differences in the Datapoints values.

18. From the Lab View toolbar, **connect** to the **vWorkstation system**.

19. **Repeat steps 2-3** on the vWorkstation.

20. **Make a screen capture** showing the **Chrome fingerprint on the vWorkstation**.

21. **Identify** the differences between the Datapoints values on the vWorkstation and TargetAndroid01.

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: Disable Third-Party Cookies

In this scenario, you will assume the role of yourself. In this lab, you examined various methods for conducting both active and passive device fingerprinting and learned that there are many legitimate reasons for fingerprinting devices. Administrators can learn about devices that are connecting to a LAN. Web developers can customize content delivery for specific platforms. And yet, you may find yourself worrying about your privacy when browsing the web, knowing that your data is constantly being harvested for tracking purposes. In the following exercises, you will examine some methods for protecting your device against malicious forms of fingerprinting.

One of the most widely used techniques for fingerprinting and tracking users online is the cookie. A cookie is a small block of data that is generated by a website when a user connects to it and then stored on the user's device, typically within the browser. While cookies serve many essential functions for the modern web, privacy-minded users may consider disabling them or otherwise limiting their use.

Using the Internet, research how to manage the cookie settings in Chrome for Android. Next, open the cookie settings in Chrome on TargetAndroid01 and observe the current selection, as well as the other options.

Make a screen capture showing the **cookie settings in Chrome on TargetAndroid01**.

What would happen if you switched to a different option for allowing versus blocking cookies, and how would it hinder attempts to track your activities online? Would there be any drawbacks?

Part 2: Test Your Browser’s Tracking Protection

In addition to limiting cookie usage, you can also access web-based tools that will automatically assess your browser's privacy settings. One such is tool is Cover Your Tracks, developed and managed by the Electronic Frontier Foundation. Cover Your Tracks will assess how visible your browser is to trackers, then provide a detailed report of how the results were measured.

From the TargetAndroid01 device, open Chrome and navigate to https://coveryourtracks.eff.org/learn, then click the Test Your Browser button to run the assessment. Once the assessment is completed, review the results.

Make a screen capture showing the **Cover Your Tracks assessment results**.

Naturally, you are encouraged to run the same assessment directly from your own local browser to reap the real-world benefits of this tool.

Note: This concludes Section 3 of the lab.