Cyber Deception: Tools and Techniques

Authors: Nandan Desai, Tariq Ahmed, Maddie Tasker-Fernandes, Kishan Kumar

Fall 2022

There are a variety of defense strategies when it comes to defending against cyber attacks. And these strategies can be roughly classified into two types: preventing an attack and detecting a breach when the prevention fails.

As the saying goes, "no system is 100% secure". Security engineers need to assume that the prevention strategies are bound to fail no matter how perfect the strategies may seem. At the end of the day, it all boils down to how quickly the engineers can detect a breach of the systems and how they will eliminate the threat.

One of the traditional ways for detecting a breach has been the use of Intrusion Detection Systems (IDSes). IDSes work by either looking for signatures of known attacks or deviations from normal activity. Attackers always keep finding creative ways to appear "normal" in front of these IDSes. So maybe, we, as defenders, can be creative as well!

That's where the concept of Cyber Deception comes in! Cyber Deception is all about laying traps inside the network so that we get alerted as soon as the attackers step on them. You might have heard of 'honeypots'. Well, that's one of the deception strategies!

The main goal of deceiving an attacker is: to buy more time for the Incident Responders and, to collect a significant amount of Threat Intelligence (like, what exactly did the attackers intend to do, what commands did they run, etc.).

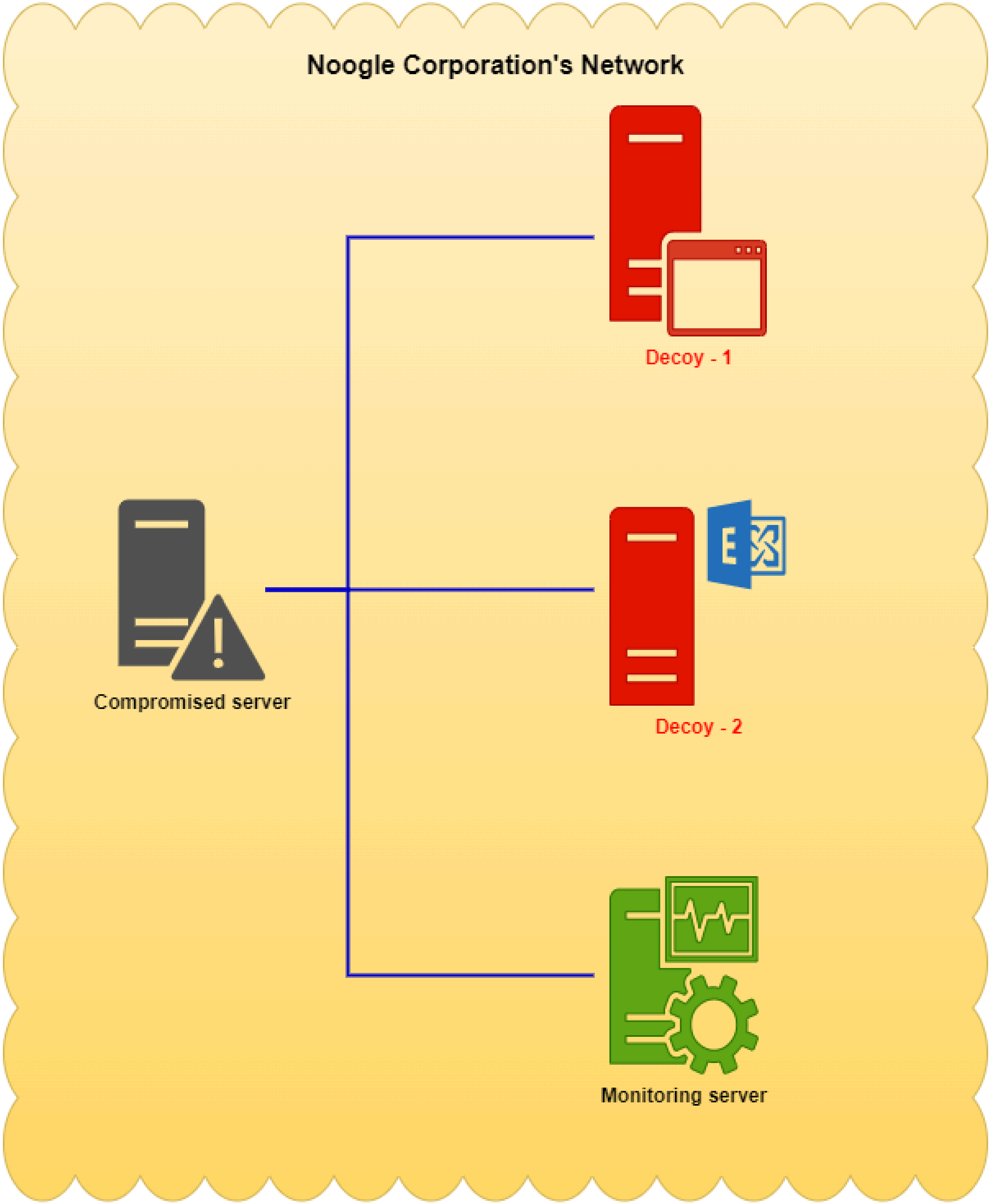
In this lab, we'll teach you how you can find creative ways to deceive an attacker once they have bypassed our primary cyber-defense systems.

Learning Objectives

In this lab you will learn:

- How to deceive an attacker when they try to run a port scanning tool (like nmap) or a path traversal tool (like gobuster).
- Different ways a honeypot can be placed in a network.
- Lastly, how to collect some actionable threat intelligence to learn more about the attacker's intentions.

Network Diagram

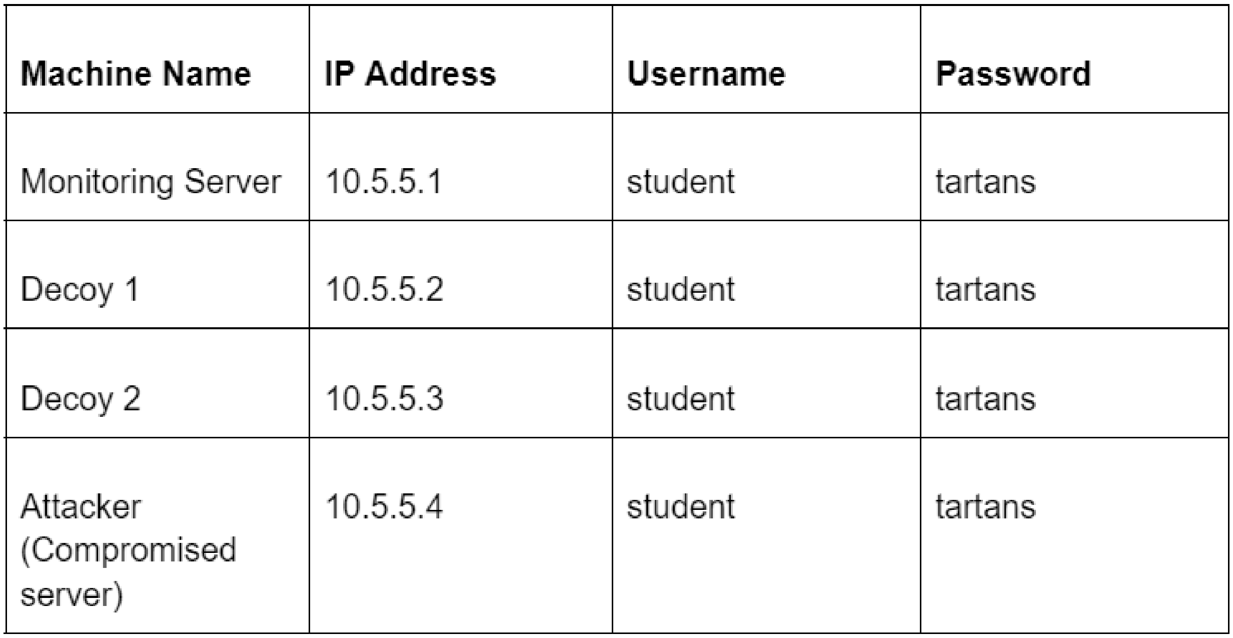


Scenario

The scenario set up in the lab activity is designed to show you how cyber deception works from both the attacker and defender perspective. You will be taking on the role of both an attacker and defender/analyst as we walk you through some different deception methods. In each stage of the lab we will be walking through a different Cyber Deception technique.

The tools and techniques used in this lab are designed to deceive and detect an attacker who is inside the network. This attacker could be an insider threat, or an outside attacker who has already breached the external firewalls.

Lab Machines



Machine Name	IP Address	Username	Password
Monitoring Server	10.5.5.1	student	tartans
Decoy 1	10.5.5.2	student	tartans
Decoy 2	10.5.5.3	student	tartans
Attacker (Compromised server)	10.5.5.4	student	tartans

Step 0: Getting Started

Please wait at least 5 minutes after starting the lab before opening any of the machines. This is so some of the services have time to start properly before starting the lab.

Before we get started with our deception techniques, let's setup our monitoring tools! For this lab, Filebeat is used to send the logs from the decoy machines to the monitoring server which is running ELK.

Run the following commands on Decoy-1 and Decoy-2 machines:

Unset

```
sudo service filebeat start
```

To view and monitor the logs in this lab, we will be using Kibana (part of ELK stack). To access the Kibana dashboard, open a browser on the Monitoring-Server and go to

Unset

```
                        http://localhost:5601/app/logs/stream
```

You will not see any logs yet, but we will come back to this later.

Login to the Compromised-Server-(Attacker) machine and run the first grading script using the following command in a new terminal:

Unset

```
python3 /home/student/Desktop/GradingScripts/quiz.py
```

Keep this script running in a separate terminal as you go through the lab, since you will be answering the questions as you go.

Let's first take a look at the network information, without any cyber deception tools running.

Run the following command, from a new terminal, on the Compromised-Server-(Attacker) machine:

Unset

```
nmap 10.5.5.0/24
```

What output do you see? Make note of the IPs, ports, and services shown in the output. You should now be able to answer question 1 of the quiz.

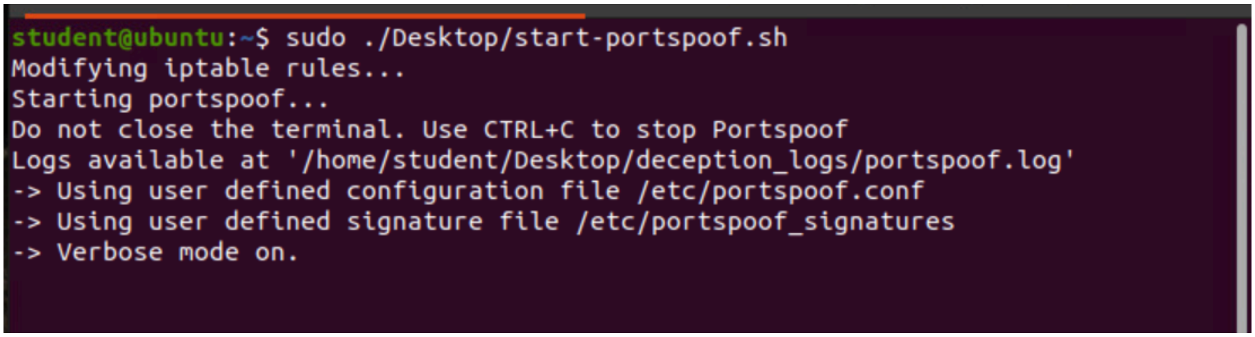
Step 1: Port Scan Deception

Prerequisite

To start our first deception tool, run the `start-portspoof.sh` script on Decoy-2. Do not close this window.

```
Unset

sudo /home/student/Desktop/start-portspoof.sh
```



```
student@ubuntu:~$ sudo ./Desktop/start-portspoof.sh
Modifying iptable rules...
Starting portspoof...
Do not close the terminal. Use CTRL+C to stop Portspoof
Logs available at '/home/student/Desktop/deception_logs/portspoof.log'
-> Using user defined configuration file /etc/portspoof.conf
-> Using user defined signature file /etc/portspoof_signatures
-> Verbose mode on.
```

Attacker View

You're now the attacker. Go to the Compromised-Server-(Attacker).

You were successful in breaking into network of Noogle Corporation and are now holding on to a compromised server. This compromised server doesn't seem too interesting to you so you're looking for other servers to jump to.

What's the one tool that might pop up in your mind when you want to search for other hosts on a network? Nmap!

Let's try to run an Nmap scan on the network again to see what we've got:

```
Unset
nmap 10.5.5.0/24
```

(This nmap scan might take ~2 mins to complete.)

You should now be able to answer question 2, 3, and 4 of the quiz.

One of the hosts in the network seems to be having a lot of open ports. Lets try to get more info about the service and version being run on that machine (using the following command):

Unset

```
nmap -sV <IP-address>
```

Did you notice anything different in the output of this command compared to the previous one? (Hint: look at the services running on the different ports.)

At this point you can run Grading Script 1 from the Decoy-2 machine using the following command:

Unset

```
python3 /home/student/Desktop/GradingScripts/gradingScript1.py
```

Defender View

The way we were able to show many open ports to the attacker is by using a tool called Portspoof. If you check out the prerequisite script that we executed earlier, you'll notice that we're modifying the `iptables` rules to redirect a range of ports to a particular port where Portspoof is running.

Portspoof is pretty good at handling Nmap probes. It doesn't just emulate fake ports, but also fake servers and their fake versions.

Two different goals could be achieved with this:

1. The first goal could be to waste the attackers time if they decide to dig deeper into the open ports.
2. The second goal could be to direct them to a different IP address, which might happen if the attacker suspects a trap and decides to ignore the open ports shown.

If the attacker chooses to dig deeper into the open ports, they'll be wasting their time and give our security analysts more time to react to this threat.

But what if the attacker chooses to ignore the open ports suspecting that this might be an obvious honeypot? The use of Portspoof would have gone to waste! Little do they know that we've taken this into account and are luring them into yet another trap (Hehe).

With both of these goals, the defender can collect logs of the attackers activity to learn more information about the attack.

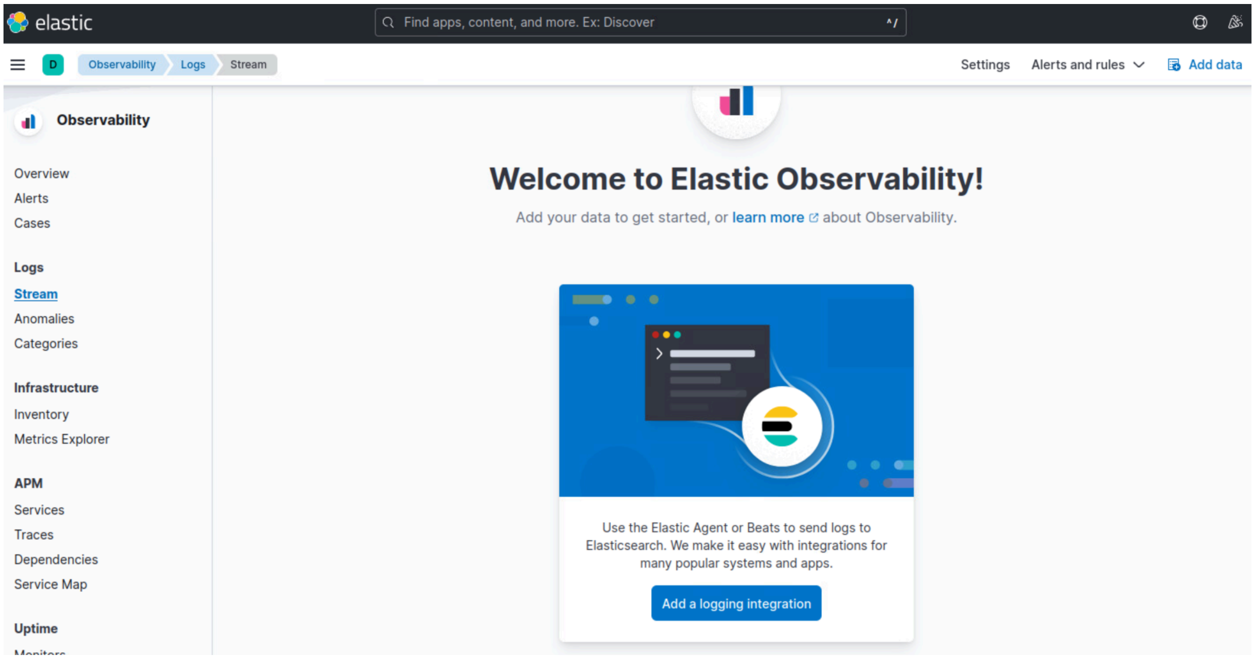
Now log onto the Monitoring-Server machine and open Kibana (`http://localhost:5601`) in the browser. Go to `Observability` --> `Logs` to examine the logs produced by Portspoof. (hint: we have set it up so you can filter by the type of attack, defined by the field `attack`. Complete details about a logged event can be viewed by clicking on `View Details` under the blue ellipsis menu.).

After examining the logs, you should now be able to answer question 5 on the quiz.

Please wait for a few mins and try refreshing the page if the logs don't show up instantly.

If you are still unable to view the logs after a few minutes (see screenshot below), you may need to run the following command on Monitoring-Server:

Unset
```
sudo systemctl restart NetworkManager
```



Welcome to Elastic Observability!

Add your data to get started, or learn more about Observability.

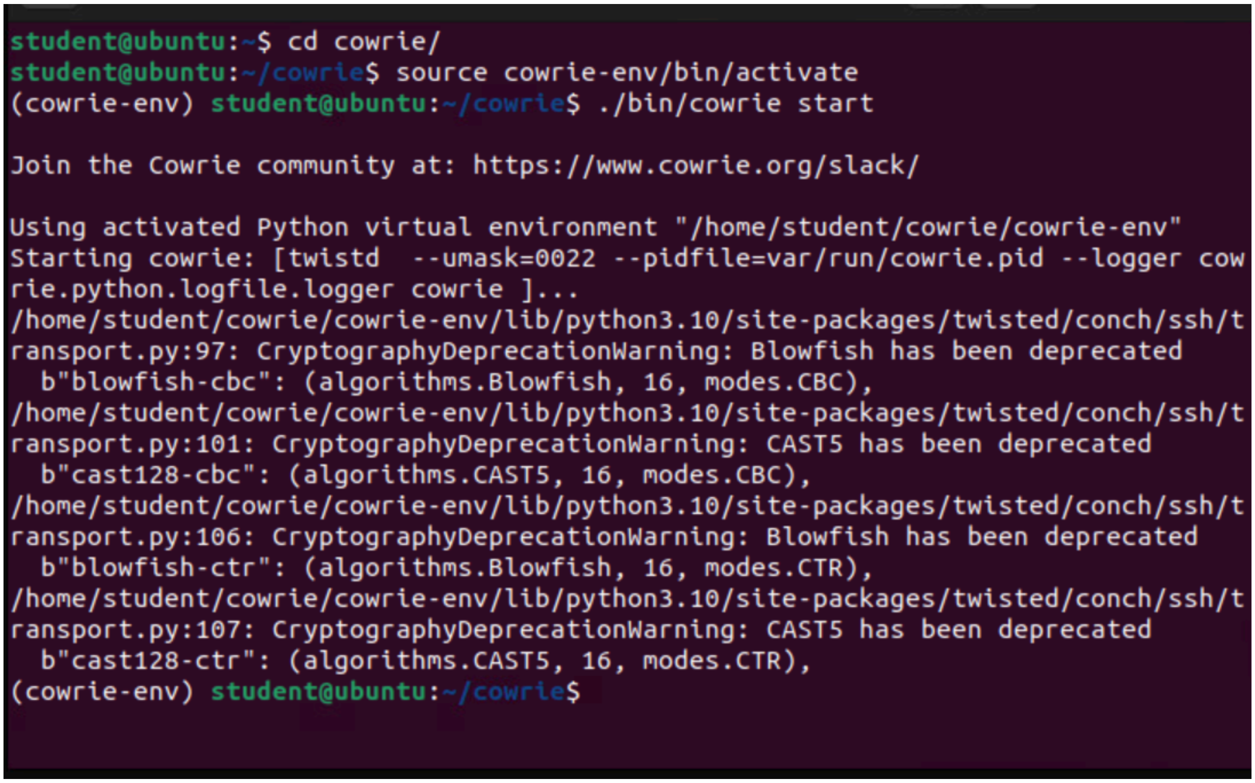
Use the Elastic Agent or Beats to send logs to Elasticsearch. We make it easy with integrations for many popular systems and apps.

Step 2: SSH Honeypot

Prerequisite

To run the next deception tool from Decoy-1, run the following commands:

```
Unset
cd /home/student/cowrie
source cowrie-env/bin/activate
./bin/cowrie start
```



```
student@ubuntu:~$ cd cowrie/
student@ubuntu:~/cowrie$ source cowrie-env/bin/activate
(cowrie-env) student@ubuntu:~/cowrie$ ./bin/cowrie start

Join the Cowrie community at: https://www.cowrie.org/slack/

Using activated Python virtual environment "/home/student/cowrie/cowrie-env"
Starting cowrie: [twistd  --umask=0022 --pidfile=var/run/cowrie.pid --logger cow
rie.python.logfile.logger cowrie ]...
/home/student/cowrie/cowrie-env/lib/python3.10/site-packages/twisted/conch/ssh/t
ransport.py:97: CryptographyDeprecationWarning: Blowfish has been deprecated
  b"blowfish-cbc": (algorithms.Blowfish, 16, modes.CBC),
/home/student/cowrie/cowrie-env/lib/python3.10/site-packages/twisted/conch/ssh/t
ransport.py:101: CryptographyDeprecationWarning: CAST5 has been deprecated
  b"cast128-cbc": (algorithms.CAST5, 16, modes.CBC),
/home/student/cowrie/cowrie-env/lib/python3.10/site-packages/twisted/conch/ssh/t
ransport.py:106: CryptographyDeprecationWarning: Blowfish has been deprecated
  b"blowfish-ctr": (algorithms.Blowfish, 16, modes.CTR),
/home/student/cowrie/cowrie-env/lib/python3.10/site-packages/twisted/conch/ssh/t
ransport.py:107: CryptographyDeprecationWarning: CAST5 has been deprecated
  b"cast128-ctr": (algorithms.CAST5, 16, modes.CTR),
(cowrie-env) student@ubuntu:~/cowrie$
```

You can check the status of Cowrie using the following command to verify if it is running.

```
Unset
./bin/cowrie status
```

Attacker View

You're now wearing the Attacker's hat again. Logon to the Compromised-Server-(Attacker). Since you suspected that the previous server might have been a honeypot, you've decided to jump to a different server.

Run the nmap scan again:

```
Unset
nmap 10.5.5.0/24
```

You'll find some new services running. What are that?

One of the services seems to be an SSH server! Let's see what lies on the other end of this seemingly lucky find.

Login to Compromised-Server-(Attacker) and connect to the SSH service by running the following command:

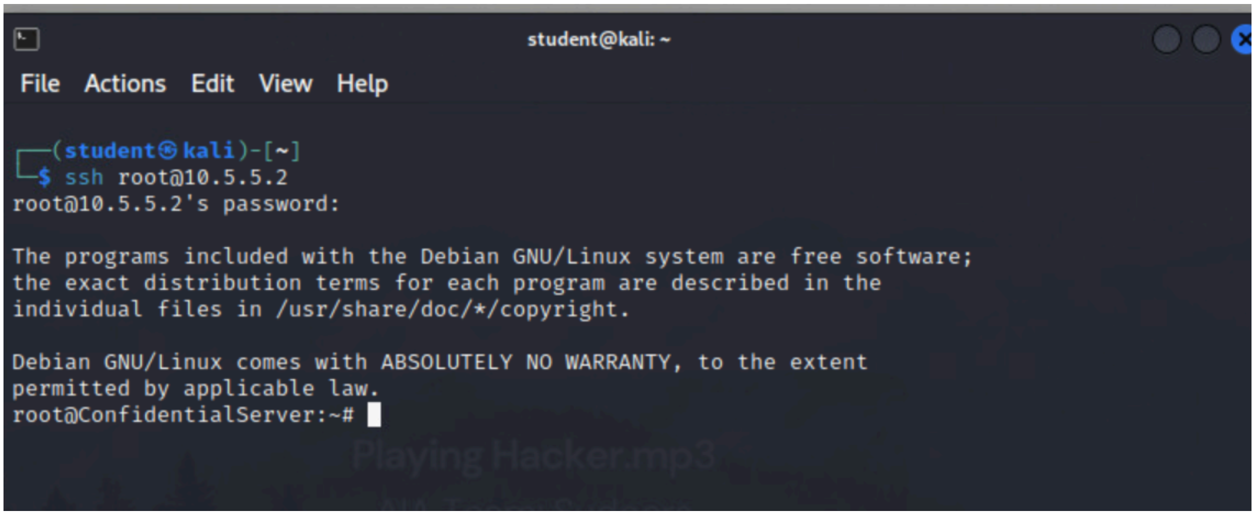
```
Unset
ssh root@<IP-address>
```

You'll be prompted for a password. Sheesh! This looks legit. As the attacker, you have written a script for yourself to help with the next part. Your script has some common passwords and it will try to brute force the login for `root`.

In a separate terminal on Compromised-Server-(Attacker) run the following commands:

```
Unset
cd /home/student/Desktop
python3 /home/student/Desktop/bruteforce.py
```

You should now be able to login via SSH using the password your script gave you. (That wasn't a very strong password choice, was it?)



```
┌──(student㉿kali)-[~]
└─$ ssh root@10.5.5.2
root@10.5.5.2's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@ConfidentialServer:~# 
```

Browse around and see if you can find anything interesting. You might want to use the `cd` command to change directories and `ls` command to list directory contents. You are looking for any information that might be interesting to you as an attacker! (Hint: there is a .txt file that contains an IP address and port number).

Once you have found the file use the `cat` command to read the contents. Make note of the IP address and port number mentioned in the text file.

You should now be able to answer question 6, 7, and 8 of the quiz. At this point you can also run Grading Script 2 from the Decoy-1 machine in a new terminal:

Unset

```
cd /home/student/cowrie
python3 /home/student/Desktop/GradingScripts/gradingScript2.py
```

You can now exit from the ssh session using the `exit` command.

Defender View

As you just saw, you (as the attacker) were able to connect to a server using SSH and access the various files within that server. As an attacker, this may seem like a successful attack, however the defense team had set this up as another way to deceive the attacker.

In this lab, we have used a tool called Cowrie to create a fake SSH server that an attacker can connect to. Although it appears to function like a normal SSH connection, there are a few key differences which make this tool useful for the defender.

With Cowrie, a completely fake file system can be set up, so even if an attacker reads those files, they are meaningless. The files you saw were specifically set up for the deceptive SSH connection, and do not show any of the real files on the machine. As an attacker in previous scenario, you found a secret file which contained some secret information. This was intentionally set up to collect more information about anyone who might be looking for hidden files.

Another important feature of Cowrie is the logging feature. Cowrie is set up to log when a connection is made, the credentials used to login, and any commands that are run. This information can be very useful to collect some actionable threat intelligence (for example, the commands that the attacker executed indicate what their intention was and what were they specifically looking for).

Now, log onto the Monitoring-Server and open Kibana (`http://localhost:5601`) in the browser. Go to `Observability` --> `Logs` to examine the logs produced by Cowrie. (hint: we have set it up so you can filter by the type of attack, defined by the field `attack`. Complete details about a logged event can be viewed by clicking on `View Details` under the blue ellipsis menu.).

After examining the logs, you should now be able to answer question 9 on the quiz.

Please wait for a few mins. and try refreshing the page if the logs don't show up instantly.

You can now stop cowrie by running the following command on Decoy-1.

Unset

```
/home/student/cowrie/bin/cowrie stop
```

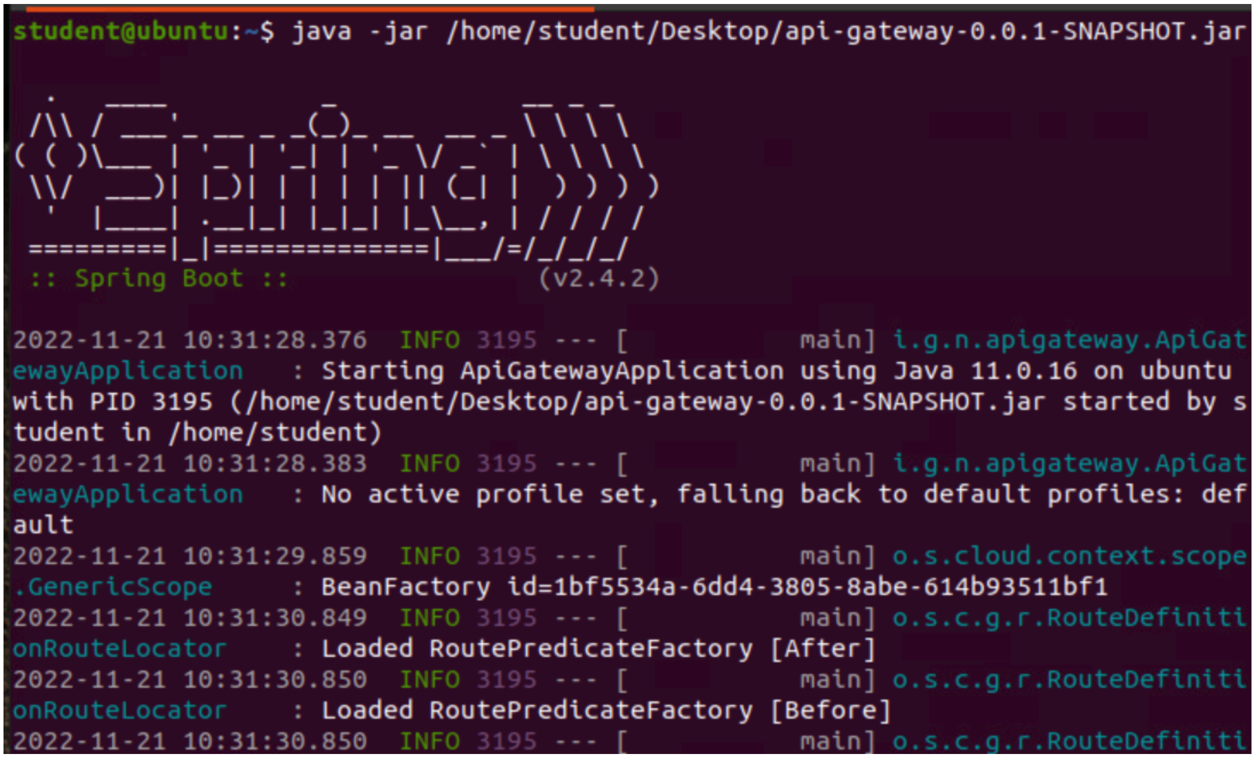
Step 3: Email Honeypot

Prerequisites

Open a new terminal on Decoy-2 and run the following command:

Unset

```
java -jar /home/student/Desktop/api-gateway-0.0.1-SNAPSHOT.jar
```



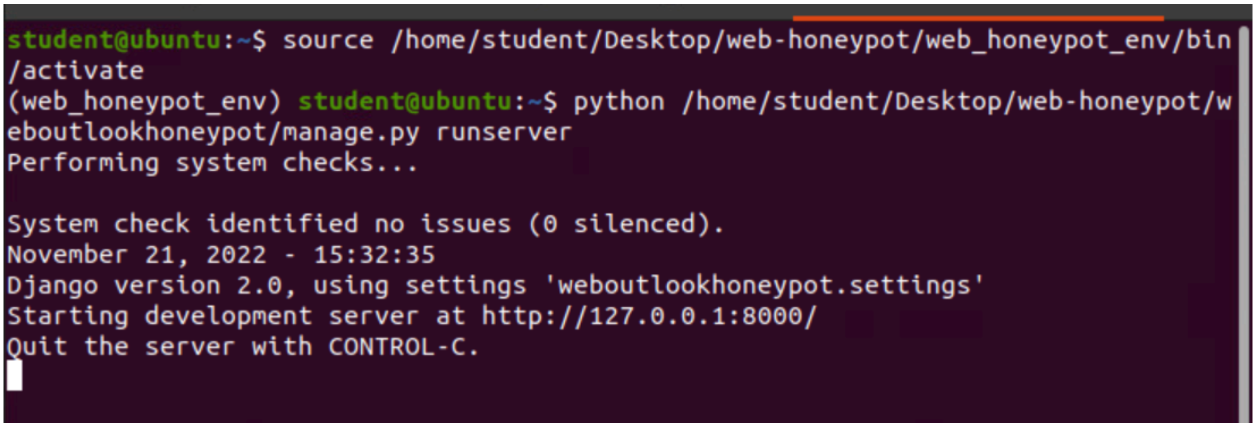
```
student@ubuntu:~$ java -jar /home/student/Desktop/api-gateway-0.0.1-SNAPSHOT.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.4.2)

2022-11-21 10:31:28.376  INFO 3195 --- [           main] i.g.n.apigateway.ApiGat
ewayApplication    : Starting ApiGatewayApplication using Java 11.0.16 on ubuntu
with PID 3195 (/home/student/Desktop/api-gateway-0.0.1-SNAPSHOT.jar started by s
tudent in /home/student)
2022-11-21 10:31:28.383  INFO 3195 --- [           main] i.g.n.apigateway.ApiGat
ewayApplication    : No active profile set, falling back to default profiles: def
ault
2022-11-21 10:31:29.859  INFO 3195 --- [           main] o.s.cloud.context.scope
.GenericScope      : BeanFactory id=1bf5534a-6dd4-3805-8abe-614b93511bf1
2022-11-21 10:31:30.849  INFO 3195 --- [           main] o.s.c.g.r.RouteDefiniti
onRouteLocator     : Loaded RoutePredicateFactory [After]
2022-11-21 10:31:30.850  INFO 3195 --- [           main] o.s.c.g.r.RouteDefiniti
onRouteLocator     : Loaded RoutePredicateFactory [Before]
2022-11-21 10:31:30.850  INFO 3195 --- [           main] o.s.c.g.r.RouteDefiniti
```

Now, open a *new terminal* again and run the following commands on Decoy-2:

```
Unset
source
/home/student/Desktop/web-honeypot/web_honeypot_env/bin/activate
python
/home/student/Desktop/web-honeypot/weboutlookhoneypot/manage.py
runserver
```



```
student@ubuntu:~$ source /home/student/Desktop/web-honeypot/web_honeypot_env/bin
/activate
(web_honeypot_env) student@ubuntu:~$ python /home/student/Desktop/web-honeypot/w
eboutlookhoneypot/manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
November 21, 2022 - 15:32:35
Django version 2.0, using settings 'weboutlookhoneypot.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

The first command activates the virtual environment for the django honeypot and the second command starts the django server.

Attacker View

(Logon to Compromised-Server-(Attacker))

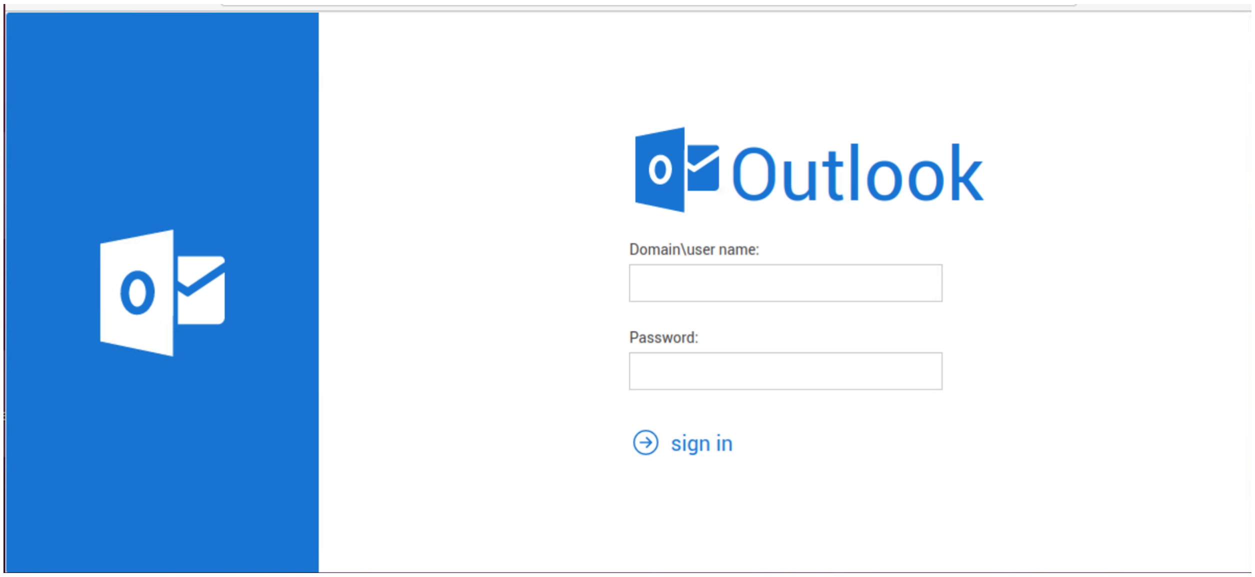
From the previous Attacker scenario, you found a .txt file on the SSH server. This seems to be an Email server judging by the URL (which has `/email/admin` at the end).

To test if it's legitimate, use curl to see what HTTP headers do you get:

```
Unset
curl -I -L <URL from file>
```

The -I (*capital i*, not lowercase L) option tells `curl` to just return the headers and -L options tells `curl` to follow the redirects.

As we can see, the Email server is accessible and running! Type the URL you found in the browser from Compromised-Server-(Attacker). An outlook login page should pop up.



Brute force time! Try logging in with a few random usernames and passwords. Can’t login, right? Yeah, it's really frustrating. Why couldn't they just have default username and passwords here too?!

At this point you can run Grading Script 3 from the Decoy-2 machine using the following command in a new terminal:

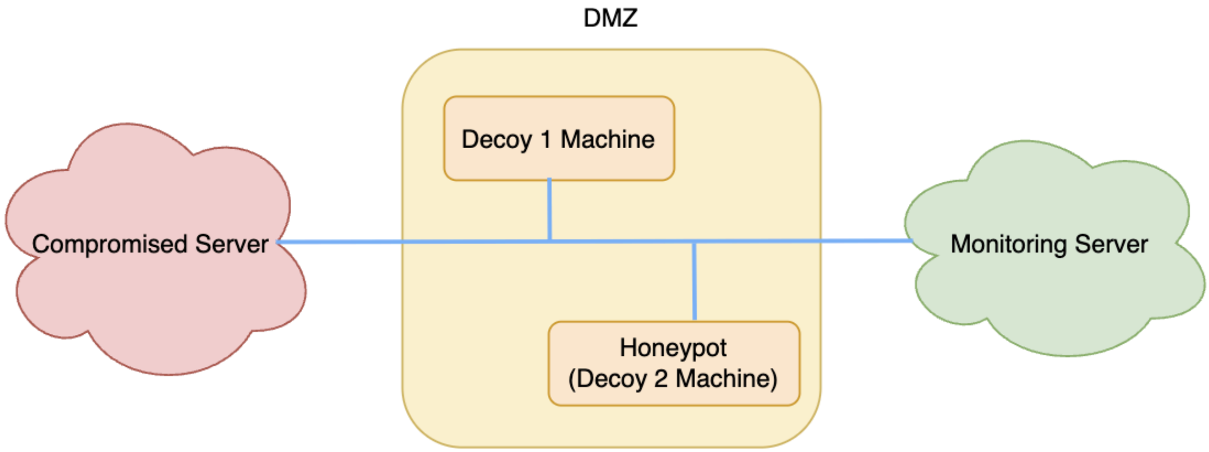
Unset

```
python3 /home/student/Desktop/GradingScripts/gradingScript3.py
```

Defender View

So, you as an attacker tried to attack and brute force into the company's "official" email server. No matter how many times you try with different usernames and passwords, you won't be able to access the server because it is a web honeypot, and it logs all your attempts and sends them to Kibana. Even Cowrie, the fake SSH server tool in the previous scenario, is also a honeypot! Ok that's enough. Let us formally introduce you to Honeypots!

So what exactly is a honeypot? A honeypot is a network-attached system that is set up as a decoy to lure the attackers and detect the hacking attempts made by them. The attacks can be logged and further analyzed to understand the nature of the attack and the details of the attacker's machine. Large cybersecurity organizations and companies that carry out security research often use honeypots to identify new threats and develop tools to defend against them. A security team can use honeypots as a cyber deception tool for their network defense strategy. It is important to note that honeypots are often placed into a demilitarized zone on the network, which keeps them isolated from the production network, thus ensuring that the critical data in the production environment is safe.



For this lab, Django-based Web Microsoft Outlook honeypot is used, a fake email server that displays a login screen when opened by the attacker. When the attacker tries to brute force, the honeypot logs the attack with details such as entered username and password, IP address of the attacker, and time and date of the attack and sends them to Kibana, which the defense team can analyze.

Now log onto the Monitoring Server and open Kibana (`http://localhost:5601`) in the browser. Go to `Observability` --> `Logs` to examine the logs produced by the Email honeypot. You should now be able to answer Question 10 of the quiz.

Please wait for a few mins and try refreshing the page if the logs don't show up instantly.

Step 4: Directory Enumeration Deception

Prerequisite

Run the following command on Decoy-2:

```
Unset
/home/student/Desktop/start-spidertrap.sh
```

Attacker View

(Logon to Compromised-Server-(Attacker).)

Since we were not successful with "/email/admin" URI path on that server in the previous scenario, why not check for other API endpoints? Maybe we have an open directory somewhere!

We'll be using Gobuster, a brute-force directory enumeration tool, to find the different URI's on the email web application. When you fill in the URL for this command, do not include the "/email/admin" path at the end. Run the following command to start the enumeration:

```
Unset
gobuster dir -u http://<IP>:<Port> -w
/usr/share/wordlists/dirbuster/directory-list-2.3-small.txt
--wildcard
```

Now sit back and watch the magic. Hah! We've got a lot of success responses.

But hold up… something seems off. It looks like the outlook application has a bunch of weird URI paths, and we're receiving success for every URI path that is tested.

Gobuster is not giving us reliable information (cue sad music). You can now stop gobuster the command by pressing Control+C.

You should be able to answer the Question 11 of the quiz. At this point you can also run Grading Script 4 from the Decoy-2 machine using the following command in a new terminal:

Unset

```
python3 /home/student/Desktop/GradingScripts/gradingScript4.py
```

Defender View

Why did Gobuster not work on Outlook? Well, that's because Gobuster was interacting with Spidertrap and not Outlook. Let us explain.

The email honeypot and spidertrap are two separate tools used for cyber deception on that server. However, we've integrated both the tools to demonstrate two types of deception in one scenario. Spidertrap and the email honeypot are running behind an API gateway. Port 8080 hosts the API gateway; not Outlook. The gateway forwards all the HTTP requests it receives to either Spidertrap or Outlook, based on the URI path. If the URI path is of the form “/email/admin/*”, the request gets redirected to the honeypot. Any other URI will go to Spidertrap.

Spidertrap returns SUCCESS for every HTTP request it receives. This wastes the attacker's time as `gobuster` would essentially be stuck in an infinite loop. Meanwhile, all the attackers attempts are being logged and sent to the monitoring server.

Now log onto the Monitoring-Server and open Kibana (`http://localhost:5601`) in the browser. Go to `Observability` --> `Logs` to examine the logs produced by Spidertrap. (hint: we have set it up so you can filter by the type of attack). You should now be able to answer Question 12 of the quiz.

Please wait for a few mins and try refreshing the page if the logs don't show up instantly.

Conclusion

In this lab we demonstrated various tools and concepts used for cyber deception and saw the way they work from both an attacker and defender's perspective.

The techniques used in this lab are just a small subset of how cyber deception can be used to secure a system and collect information about possible threat actors. There are many more open source tools for different types of cyber deception which can be explored and used to create different types of traps to catch an attacker.

We hope this lab activity has given you an exciting introduction to cyber deception and given you a glimpse of some of the possibilities for new ways to prevent attacks and catch malicious actors.

References

- https://github.com/dmpayton/django-admin-honeypot/blob/master/docs/manual/usage.rst
- https://www.techtarget.com/searchsecurity/definition/honey-pot
- https://github.com/strandjs/IntroLabs
- https://github.com/cowrie/cowrie
- https://cowrie.readthedocs.io/en/latest/INSTALL.html
- https://cryptax.medium.com/customizing-your-cowrie-honeypot-8542c888ca49
- https://www.elastic.co/kibana
- https://www.elastic.co/elastic-stack
- https://github.com/strandjs/IntroLabs/blob/master/IntroClassFiles/Tools/IntroClass/Portspoof.md
- https://github.com/adhdproject/spidertrap

Grading Scripts

Quiz

Python

```python
import hashlib
score = 0

answer = input("Question 1: What open ports do you see? (any
order, comma separated)\n")
if hashlib.md5(answer.encode()).hexdigest() ==
"4abdde2c274814332a677a165600187c" or
hashlib.md5(answer.encode()).hexdigest() ==
"01897d5f078f97daed2bf764e6569396":
    score = score + 1
    print("Correct!")
else:
    while hashlib.md5(answer.encode()).hexdigest() !=
"4abdde2c274814332a677a165600187c" and
```

```
hashlib.md5(answer.encode()).hexdigest() !=
"01897d5f078f97daed2bf764e6569396":
        answer = input("Try again...\n")
    score = score + 1


answer = input("Question 2: What IP address has the most open
ports?\n")
if hashlib.md5(answer.encode()).hexdigest() ==
"6230792f0cc24f80d2e0fb2a9d7a2b37":
    score = score + 1
    print("Correct!")
else:
    while hashlib.md5(answer.encode()).hexdigest() !=
"6230792f0cc24f80d2e0fb2a9d7a2b37":
        answer = input("Try again...\n")
    score = score + 1


answer = input("Question 3: How many open ports were found on the
IP address from question 2?\n")
if hashlib.md5(answer.encode()).hexdigest() ==
"17e62166fc8586dfa4d1bc0e1742c08b":
    score = score + 1
    print("Correct!")
else:
    while hashlib.md5(answer.encode()).hexdigest() !=
"17e62166fc8586dfa4d1bc0e1742c08b":
        answer = input("Try again...\n")
    score = score + 1

answer = input("Question 4: What service appears to be running on
port 3306 of the IP address from question 2?\n")
if hashlib.md5(answer.encode()).hexdigest() ==
"81c3b080dad537de7e10e0987a4bf52e":
    score = score + 1
    print("Correct!")
else:
    while hashlib.md5(answer.encode()).hexdigest() !=
"81c3b080dad537de7e10e0987a4bf52e":
        answer = input("Try again...\n")
    score = score + 1

answer = input("Question 5: What is the attack type shown in the
logs?\nHint: you can filter by attack type in Kibana\n")
if hashlib.md5(answer.encode()).hexdigest() ==
"171781bffcb95064bfa1475852ff9f4c":
    score = score + 1
    print("Correct!")
else:
    while hashlib.md5(answer.encode()).hexdigest() !=
"171781bffcb95064bfa1475852ff9f4c":
        answer = input("Try again...\n")
    score = score + 1

answer = input("Question 6: What password is used to login as
root?\n")
if hashlib.md5(answer.encode()).hexdigest() ==
"7b0630a94b5e66a9d03dba7fbcf827b9":
    score = score + 1
    print("Correct!")
else:
    while hashlib.md5(answer.encode()).hexdigest() !=
"7b0630a94b5e66a9d03dba7fbcf827b9":
        answer = input("Try again...\n")
    score = score + 1

answer = input("Question 7: What is the name of the hidden file
containing the IP and port?\n")
if hashlib.md5(answer.encode()).hexdigest() ==
"0e7c1c94cea673584a750ee36a3de4be":
    score = score + 1
    print("Correct!")
else:
    while hashlib.md5(answer.encode()).hexdigest() !=
"0e7c1c94cea673584a750ee36a3de4be":
        answer = input("Try again...\n")


answer = input("Question 8: What is the IP address and port found
in the hidden file? (format: IP:port)\n")
if hashlib.md5(answer.encode()).hexdigest() ==
"06859613e1e269feeb79334606f7572c":
    score = score + 1
    print("Correct!")
else:
    while hashlib.md5(answer.encode()).hexdigest() !=
"06859613e1e269feeb79334606f7572c":
        answer = input("Try again...\n")

answer = input("Question 9: What attack type is shown in the
logs?\n")
if hashlib.md5(answer.encode()).hexdigest() ==
"1787d7646304c5d987cf4e64a3973dc7":
    score = score + 1
    print("Correct!")
else:
    while hashlib.md5(answer.encode()).hexdigest() !=
"1787d7646304c5d987cf4e64a3973dc7":
        answer = input("Try again...\n")
    score = score + 1


answer = input("Question 10: What attack type is shown in the
logs?\n")
if hashlib.md5(answer.encode()).hexdigest() ==
"d02812cb1f9c52aee440acbb117f7946":
    score = score + 1
    print("Correct!")
else:
    while hashlib.md5(answer.encode()).hexdigest() !=
"d02812cb1f9c52aee440acbb117f7946":
        answer = input("Try again...\n")
    score = score + 1

answer = input("Question 11: What HTTP return code is shown when
gobuster is run?\n")
if hashlib.md5(answer.encode()).hexdigest() ==
"3644a684f98ea8fe223c713b77189a77":
    score = score + 1
    print("Correct!")
else:
    while hashlib.md5(answer.encode()).hexdigest() !=
"3644a684f98ea8fe223c713b77189a77":
        answer = input("Try again...\n")
    score = score + 1

answer = input("Question 12: What is the attack type shown in the
logs?\n")
if hashlib.md5(answer.encode()).hexdigest() ==
"acce479e3ac4f89c3233c1111feb78ae":
    score = score + 1
    print("Correct!")
else:
    while hashlib.md5(answer.encode()).hexdigest() !=
"acce479e3ac4f89c3233c1111feb78ae":
        answer = input("Try again...\n")
    score = score + 1


print("Your total score is: " + str(score) + "/12")
```

Grading Script 1

Python

```python
import subprocess
import os
from datetime import date

passed = True

#Check if nmap shows open port for portspoof
result = subprocess.run(["nmap 10.5.5.3"], shell=True,
capture_output = True)
if "4444/tcp open" not in str(result.stdout):
    passed = False

#Check if nmap scan was run
result = subprocess.run(["cat
/home/student/Desktop/deception_logs/portspoof.log"], shell=True,
capture_output = True)
if "Service_probe # SIGNATURE_SEND # source_ip::: # dst_port:"
not in str(result.stdout):
    passed = False

if passed == True:
    print("Grading Script Passed!")
else:
    print("Grading Script Not Passed...")
```

Grading Script 2

Python

```python
import subprocess
import os
from datetime import date

passed = True
os.chdir("/home/student/cowrie")
```

```
#Check if cowrie is running on port 22
result = subprocess.run(["nmap 10.5.5.2 | grep 22/tcp"],
shell=True, capture_output = True)
if "22/tcp" not in str(result.stdout):
    print(1)
    passed = False

#Check if brute force was successful as root (use logs to check
for username/password combo)
result = subprocess.run(["cat var/log/cowrie/cowrie.log"],
shell=True, capture_output = True)
if "login attempt [b'root'/b'aia123!'] succeeded" not in
str(result.stdout):
    print(2)
    passed = False

#Check if file was found and read from attacker machine (use logs
to show if user typed "cat filename")
result = subprocess.run(["cat var/log/cowrie/cowrie.log"],
shell=True, capture_output = True)
if "cat SecretEmailServer.txt" not in str(result.stdout):
    print(3)
    passed = False

if passed == True:
    print("Grading Script Passed!")
else:
    print("Grading Script Not Passed...")
```

Grading Script 3

Python

```python
import subprocess
import os
from datetime import date
```

```
passed = True

#Check if login was attempted was run
result = subprocess.run(["nmap 10.5.5.3"], shell=True,
capture_output = True)
if "8080/tcp open  http-proxy" not in str(result.stdout):
    passed = False

#Check if login was attempted was run
result = subprocess.run(["cat
/home/student/Desktop/deception_logs/web_honeypot.log"],
shell=True, capture_output = True)
if "Username" not in str(result.stdout):
    passed = False

result = subprocess.run(["cat
/home/student/Desktop/deception_logs/web_honeypot.log"],
shell=True, capture_output = True)
if "http://localhost:8000" not in str(result.stdout):
    passed = False

if passed == True:
    print("Grading Script Passed!")
else:
    print("Grading Script Not Passed...")
```

Grading Script 4

Python

```python
import subprocess
import os
from datetime import date

passed = True

#Check if Spidertrap was run
```

```
result = subprocess.run(["cat
/home/student/Desktop/deception_logs/spidertrap.log"],
shell=True, capture_output = True)
if "Starting server on port 8020" not in str(result.stdout):
    passed = False

result = subprocess.run(["curl 10.5.5.3:8080/test"], shell=True,
capture_output = True)
if "<html>" not in str(result.stdout):
    passed = False

#Check if gobuster was run
result = subprocess.run(["cat
/home/student/Desktop/deception_logs/spidertrap.log"],
shell=True, capture_output = True)
if "GET" not in str(result.stdout):
    passed = False

#Check if gobuster was run
result = subprocess.run(["cat
/home/student/Desktop/deception_logs/spidertrap.log"],
shell=True, capture_output = True)
if "200" not in str(result.stdout):
    passed = False

if passed == True:
    print("Grading Script Passed!")
else:
    print("Grading Script Not Passed...")
```