Department of Information Engineering and Computer Science

NETWORK SECURITY
LABORATORY REPORT

LAB 10
HONEYPOT

Da Rold Giovanni

224291

Meschini Marcello

220222

Academic year 2020/2021

Contents

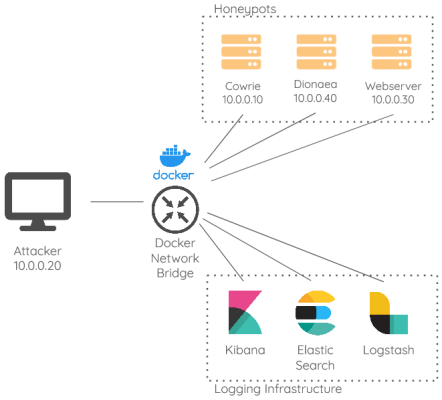
Info about the lab

Requirements

- Docker Engine version 17.05 or newer
- Docker Compose version 1.20.0 or newer
- 2 GB of RAM
- At least 20 GB of disk

Network Topology

To create a network for the laboratory we used Docker Compose and created the following topology:



The docker-compose.yaml file containing the definition for the containers can be found in the netsec-honeypot-lab folder on the desktop or in our Github public repository:

`https://github.com/Marcy-P/netsec-honeypot-lab`

The repository README also contains additional info for accessing the containers and the references to some Docker images we used.

Starting up the lab

To start the laboratory login into the VM with the credentials: username: *netsec* and password: *password*. Then open a terminal in the folder \netsec-honeypot-lab on the desktop and type the following command:

```
$ ./start.sh
```

Shutting down the lab

To shut down the docker-compose network type:

```
$ docker-compose down
```

To also clean the persistent data present in Elasticsearch type:

```
$ docker-compose down -v
```

1 What is a honeypot?

"A honeypot is a network-attached system set up as a decoy to lure cyber attackers and detect, deflect and study hacking attempts to gain unauthorized access to information systems"[1]. So it is a system that is unprotected and serves no business purpose but sits in the network waiting to be contacted. Every interaction with a honeypot is suspicious because no legitimate user should utilize it.

2 Characteristics of a honeypot

Honeypots have four main characteristics; they have to be: **Deceptive** , **Discoverable**, **Interactive**, **Monitored**.

2.1 Deception

Deception can be defined as an advantageous distortion of an adversary's perceptions of reality. Honeypots heavily use this concept as a tool because they appear as real systems, but they do not serve any functional purpose for a business.

There are various frameworks that try to classify various deception strategies, and we considered the taxonomy proposed by Bell and Whaley [3]. According to this model deception consists in two main parts: hiding the real (dissimulation) and showing the false (simulation).

Honeypots are meant to be reachable, so you do not hide it entirely, but you often have to hide specific features of them to make them less suspicious. The techniques to hide the real are:

- **Masking**: hide the real by making relevant objects be undetectable or blend into the background;
- **Repackaging**: hide the real by making it appear like it is something different;
- **Dazzling**: hide the real by altering an object to confuse the adversary;

The techniques to show the false are:

- **Mimicking**: show the false by using characteristics present in the actual real object;
- **Inventing**: show the false by giving the perception that a relevant object exist while it does not;
- **Decoying**: show the false by misdirecting and attracting the attacker attention away from real objects.

Note that not every honeypot has to strictly follow these techniques. It is just a theoretical framework that you may want to use when creating and deploying a honeypot.

2.2 Discoverability

Honeypots are not meant to be accessed by legitimate users but just the attacker. So when designing your honeypot you have to consider the point of view of the attacker. In particular, you can ask yourself the following questions:

- Where is the attacker more likely to enter your network?
  - User workstations

  - Vulnerable services exposed to the internet
  - Stolen VPN credentials
- What tool will they use to discover your asset?
- What assets will they be interested in?

To increase discoverability, you might also place **breadcrumbs** in systems that might be compromised. They are data that will lead the attacker to your honeypot while the attacker is gathering information needed to do lateral movement in the network. An example of breadcrumbs is a clear text document containing an URL or IP of a honeypot and some credential or SSH private keys.

2.3 Interactivity

Technically a single attacker interaction is enough to produce an alert. But is better if the honeypot responds back to the attacker for two main reasons:

- Each interaction that attacker does with the honeypot could provide you important information. For example, if they stole some credential, you may be able to know the account that was compromised. You may be able to find out what tool they use to enter your network
- Make the attacker waste time. Every second that the attacker spends interacting with the honeypot is a second you can invest into finding out which systems were compromised and isolate the attacker

2.4 Monitoring

Finally, honeypots have to be monitored. This means that every notable interaction with a honeypot must generate an event or alert. This alert generally gets fed to a central SIAM that is accessible to the network administrator or security team.

To manage all the event generated in an environment with multiple honeypots, you have to build a **logging and monitoring infrastructure** that stores the logs and alerts the analyst of the presence of an attack. The process of shipping logs from the honeypot to the monitoring server usually happens in four phases:

1. **Source**: the data are generated from the honeypot. Note that each honeypots produce logs in their own format. You need to be aware of such format because often it is not compatible with the Logging and Monitoring Server.
2. **Egress**: the data are transmitted to a central location. There has to be a sender capable of reading the logs and sending them to another location. In this phase you also can do some filtering if some of the information provided in the logs are not needed or conversion to other formats. NB: Since you are transferring data over the network you have to consider all the CIA properties because the logs are sensitive data. If you do not encrypt data, you are vulnerable to various attacks that we already see in other laboratories. Can you guess some of them? ARP cache poisoning and MITM. If the attacker can intercept the logs, it can know that a system is a honeypot without never interacting with it!
3. **Ingress**: the data are received. There should be a component that receives logs and parses them in a format compatible with the destination. In this phase it is also important to do additional filtering because some honeypots are very noisy. You may want to filter out interaction that are not important [e.g., not log whey they interact for the first time with the honeypot but just when they authenticate] or interaction from component of the network that may interact with the honeypot (e.g., scanners)
4. **Destination**: The data are stored in the destination inside a database. You also can implement a dashboard to see them. This component must be protected because like already said these logs are sensitive data. You may also want a system to backup important logs.

Another mechanism that you can implement in parallel to the logging infrastructure is a monitoring infrastructure. You do not want to miss an attack because a honeypot was down because of a crash. This infrastructure allows to keep track of the status of each honeypot and alert you when a honeypot goes down.

3 Honeypots classification

3.1 Based on level of interaction

Honeypots are classified by their degree of interaction with the communication destination (attacker) as a low interaction, medium interaction or high interaction type. However, this does not mean that the high interaction type is better than the low interaction type. These terms are used in different ways depending on implementation, usage and purpose.

- Low interaction: low interaction honeypots capture the information about the predetermined activities of attackers, and limit these activities by using emulation software. They exclusively utilize emulation software and its services, hence the quality of these honeypots is decided by the quality of emulation software. They are easier to maintain and deploy, and are also more secure compared to high interaction honeypots, because the attacker's action is limited in the emulated area of the computer. Nonetheless, because it does not imitate the actual OS or application completely, the attacker may notice this machine as a honeypot.

- High interaction: high interaction honeypots allow the attackers interacting with real systems and do not assume anything about the possible behavior of the attacker. They help the administrator to investigate the complete activities of the attackers, utilizing the operating systems (eg Linux) and applications (eg FTP) without using the emulation software (eg dionaea) to record the activities of attackers. In most cases attackers are unable to recognize the honeypot because it behaves similarly to the actual target host. However these systems require more resources and are difficult to set up and maintain; also the possibility of infection from the malware or hacking by the attacker is higher than for the low interaction type honeypot. Therefore, the system should be monitored properly, and it is necessary to be prepared to restore the system to the state before the infection of malware.

- medium-interaction honeypots are a combination of low-interaction and high-interaction honeypots, capable of emulating full services or specific vulnerabilities. Similar to low-interaction honeypots, their primary purpose is detection and are installed as an application on the host operating system with only the emulated services being presented to the public. The key feature of a medium-interaction honeypot is application layer virtualization. They do not aim at simulating a fully operational system environment, nor do they implement all details of an application protocol. They provide sufficient responses that known exploits waiting on certain ports will be tricked into sending their payload. Once this payload has been received, the shellcode can be extracted and analyzed. The medium-interaction honeypot then emulates the actions the shellcode would perform to download the malware.

3.2 Based on the goal

- Research honeypots: these are deployed by various security researchers to assess cyber threats, study the real motivation behind the attacks and capture various zero-day exploits. They help in gathering information about attacks and what kind of hacking (attacking) techniques are being used by the attackers. Various logging tools are used to log each and every activity happening on the honeypot, which can really help researchers to study the various advanced methodologies used by an attacker. They mainly consist of high interaction honeypots, in fact they are complex to deploy and maintain.

- Production honeypots: these are deployed by organizations to mitigate threats and protect themselves from various attacks. They require less resources and can be easily deployed. Production honeypots are often placed near security assets, in order to alert defenders of an attack; they are cheap and usually belong to the low interaction type. Use of low interaction type includes port scan identification, generation of attack signatures, trend analysis and malware collection. On the other hand, this is also a disadvantage, indeed it is not possible to watch how an attacker interacts with the operating system as all the services are emulated.

4 Advantages of honeypots

Honeypots present certain advantages over other defense systems, which is why they should be considered in the development of a well-secured network.

Firstly they consume few resources, especially if compared to traditional IDSs. While an IDS may have to handle gigabytes of data per second, a honeypot has just to handle the activities directed to itself. But it's important to remember that honeypots are not a substitute for IDS, because usually honeypots are more a reactive control, whereas IDSs act more as preventive controls. Also IDSs deal with known threats (but not only) while honeypots deal with unknown one.

A second advantage is the fact that they are simple and cheap to maintain. Especially for those belonging to the low interaction type, you can prepare a script that can deploy hundreds of honeypots in your network.

As a third advantage they produce a small number of logs with few false positives, because there is no reason for legitimate users to access the honeypot. So you do not need a whole team of security experts that have to keep checking alerts.

5 Cowrie

5.1 What is Cowrie?

Cowrie is an open-source medium interaction SSH, and Telnet honeypot designed to log brute force attacks and the shell interaction performed by the attacker [2]. It can run in two modes:

- In Medium interaction mode (shell) it emulates a UNIX system. To do so it provides:
  - An authentication mechanism via SSH
  - A post authentication shell
  - The ability to create fake file system with files that attacker can cat
  - A mechanism to save files downloaded inside the shell with wget, curl or uploaded with SFTP and scp
- In High interaction mode (proxy) it functions as an SSH and telnet proxy to observe attacker behaviour to another system. To use this mode you have to provide a real secure backend environment where the attacker can execute any Unix command.

In this lab we will just focus on the Medium interaction mode because is the most popular one and can be used easily inside a virtual machine.

5.2 Start-up Cowrie

To enter in the Cowrie Docker instance open a terminal in the *netsec-honeypot-lab* folder and type:

```
$ ./startup/cowrie.sh
```

Cowrie is already installed and you can start it by typing:

```
$ cowrie start
```

You will find the logs in the folder `var/log/cowrie/cowrie.log`. To print them as they get generated execute the following command:

```
$ tail -f var/log/cowrie/cowrie.log
```

5.3 Brief exercise - The attacker point of view [4-5 minutes]

Enter in the attacker machine by typing in another terminal:

```
$ ./startup/attacker.sh
```

As the attacker, you successfully installed nmap on the compromised machine and identified a host with the IP 10.0.0.10 (Cowrie IP address). Try to execute a probe scan to see what services run on that host. You can do it with the command:

```
$ nmap -sV 10.0.0.10
```

You should see that with the Cowrie default configuration the (fake) SSH service run on port 2222. Now you should try to connect using common credential for example username: *root* and password: *password*. To do that type the command:

```
$ ssh root@10.0.0.10 -p2222
```

Once you are logged in inside the honeypot, play a bit in it and list some strange behaviours you encountered that suggest you that the system is a honeypot. Things to keep in mind:

1. With the default configuration there is a time-out that will log you out every two minutes
2. While inside the honeypot don't worry about breaking things.
3. If you get stuck for some reason type in the cowrie terminal: $ cowrie stop and then start it again.

5.4 Brief exercise - Solution

Here a list of behaviour you may have encountered:

- If you type the command:

  ```
  $ uname -v
  ```

  You get that the system is a Debian Linux distribution. So you may want to try to install some software. For example by typing:

  ```
  $ apt-get install ftp
  ```

  But after the installation screen every time you try to execute the newly installed command you will get the error SEGMENTATION FAULT.
- When you try to `cat` some files they seem to exist but you will get the error "*No such file or directory*"
- You cannot kill processes
- You are able to downloads file with wget but when you check their size with the command ls -al you see them as 0 Byte files
- If you delete all the files in the root folder nothing breaks. And if you re-login into the honeypot all the file will return.

5.5 A consideration about the previous exercise

Some may think that is not that hard to figure out that the system is a honeypot, but you have to keep in mind that when the attacker is inside the honeypot we already logged every character he typed. You can actually replay each session of interaction with the command:

```
$ bin/playlog var/lib/cowrie/tty/[session_log_ID]
```

And also check the files that the attacker downloaded by checking the following folder:

```
$ ls var/lib/cowrie/downloads/
```

5.6 Customizing Cowrie

The honeypot with the default configuration is almost empty, but the goal is to make the attacker believe that it is a legitimate system. In this way you can increase its interaction with the system and gather intel about what is seeking or what tool it uses. To make Cowrie resemble a legitimate system you have to start by modifying the cowrie config files.

The first thing to do is to copy the default configuration file `cowrie.cfg.dist` in a file named `cowrie.cfg`, Cowrie will give priority to this file if it exist:

```
$ cd etc
$ cp cowrie.cfg.dist cowrie.cfg
```

Now you can access the file with a text editor (e.g. vim or nano) and edit the following sections:

- Scroll down a bit and change the `hostname`. You might want to mimic similar server name existing in the network range of the honeypot. Or you can also stand out with something original.
- Search for timeout to change the interaction and authentication timeouts to a new value
- Search for `listen_endpoints` in the `[shell]` section (second match of the search) and modify the port from 2222 to 22

NB: for every modification you want to apply remember to restart Cowrie with the command:

```
$ cowrie restart
```

5.6.1 Modifying the login credentials

Another fundamental part to customize are the credentials that the attacker can use to SSH into the honeypot. To achieve this copy the file `userdb.example` to `userdb.txt`:

```
$ cd etc
$ cp userdb.example userdb.txt
```

Inside this file you will find an explanation of the syntax to use. Try to use generic credentials that can be easily guessed but without raising suspicion.

5.6.2 Modifying the process list

Cowrie allow you to create a list of fake processes that will be printed with the command `ps`. To do that modify the JSON file at the following path: `share/cowrie/cmdoutput.json`. The syntax is self explanatory. When you edit it try to remove processes that are not likely to appear in your system and add processes that are running on a real systems

5.6.3 Modifying the file system

Cowrie provide an emulated file system so when an attacker login it will get its own personal copy of this filesystem that will be deleted when they log off. This file system is implemented into two components:

- **pickle file**: contains metadata for the files (their name, size, permission, directory, ...)

- **honeyfs directory**: contains the file contents

NB: to be visible (`ls`) a file must be in the pickle file, and to be also accessible (`cat`) it needs to be in the honeyfs directory.

You can find the `honeyfs` directory in the cowrie installation folder. An interesting file you can modify is `motd` that contain the SSH post login banner.

```
$ cd honeyfs
$ vim etc/motd
```

If you created a new user credentials in the section 5.6.1 remember to create also the home directory for that user. Also, for the next section create a file `/bin/nmap`.

Finally you can generate the pickle file automatically by typing:

```
$ rm ./share/cowrie/fs.pickle
$ ./bin/createfs -l honeyfs -d 5 -o ./share/cowrie/fs.pickle
```

The flag `-l` is used to specify the location of the honeyfs folder, the flag `-d` sets the depth of the file system you want to include. Finally the flag `-o` sets the location of the pickle file you are creating. Cowrie will search for the pickle file in the `./share/cowrie/` directory.

5.6.4 Adding new commands

To add a new command add a file in the folder `share/cowrie/txtcmds/bin` containing a static output for that command. Previously we created the binary file for nmap so now you can try to create an output for nmap.

```
$ vim share/cowrie/txtcmds/bin/nmap
```

Now when executing the command `nmap` the attacker will receive the static output you specified.

6 Dionaea

7 Honeyfiles

A honeyfile is an intrusion detection mechanism based on deception. This mechanism works by creating a bait file that when accessed set off an alarm or log the access to the document. In this way you can detect unauthorized access to a computer. This is a very simple technique, but it is also very effective. Usually, in many organizations important information gets stored in office documents. So, when an attacker compromises a machine, he often checks for valuable documents.

When "setting up" a honeyfile you have to think about what the attacker is interested in. Usually they search for four things:

1. Information for accessing other systems (e.g., passwords, network details) or security documentation
2. File containing important information's like intellectual property, customer lists, product design.
3. File containing personal information that could be used to harm reputation or allow further compromise of the system
4. Log files that can contain evidence of attacks.

Then you have to think what the attacker will see, so:

- You must place the file in a way that can be easily found. But if it is a system used by legitimate users that are unaware of the honeypot, you should place it in a way that is unlikely to be accidentally opened.
- Is very effective when the file stands out from other files. You can achieve this in various ways:
  - FILE NAME IN CAPS
  - Usage of a document type different from the other documents around it
  - Setting a unique or odd file size (e.g., a 23 MB word document between 15 kB documents)

NB: Do not exaggerate by placing to many honeyfiles, because it will also increase the chance of triggering false positives by legitimate users. And also do not try to hard with the naming conventions otherwise the attacker will become suspicious.

7.1 How to create a honeydoc

8 Lab monitoring infrastructure

To implement our logging infrastructure we used the Elastic Stack (also known as ELK stack) an open-source log management system composed of four components:

1. **Beat**: Is the component that assumes the role of the egress. More technically is a family of lightweight data shippers that can be installed in each honeypot system to send various types of information. For each information there is a specific beat agent, for example:
   - **Filebeat**: it is used to send logs and files. It allows also to parse common logs format. We used this agent to send our honeypot logs to Logstash.
   - Metricbeat: agent used to ship metric data.
   - Winlogbeat: to ship windows event logs.
   - Heartbeat: to ship data regarding uptime monitoring
2. **Logstash**: This component assumes the role of the ingress. It receives data from the various Filebeats, parse and filter them. Then sends its output (that is in JSON format) to Elasticsearch. NB: It is very powerful because you can use it to structure unstructured data and enrich them with new information (for example geo coordinates based on the IP address inside a log).
3. **Elasticsearch**: it is the destination of the data. It receives the data from Logstash, store them in a database and indexes them. This process allows to run queries on the data and aggregate them into useful information.
4. **Kibana**: is a web-based dashboard that is used to visualize the data present in Elasticsearch. It allows you to create real-time histograms, pie charts and maps

9 Honeypots for IoT

Bibliography

[1] Honeypot definition. https://searchsecurity.techtarget.com/definition/honey-pot.

[2] What is cowrie. https://cowrie.readthedocs.io/en/latest/README.html.

[3] Barton Whaley. Toward a general theory of deception. *Journal of Strategic Studies*, 5(1):178–192, 1982.