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

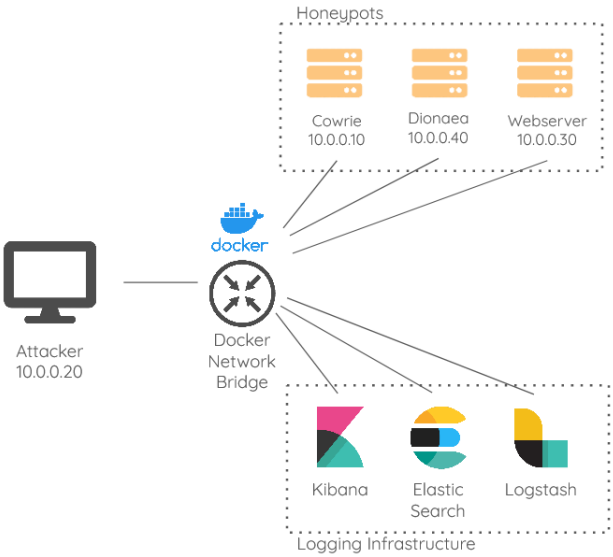
Info about the lab

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 information 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"[2]. 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** [12].

2.1 Deception

Deception can be defined as an advantageous distortion of an adversary's perception 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 [15]. According to this model deception consists of two main parts: hiding the real (*dissimulation*) and showing the false (*simulation*).

Honeypots are meant to be reachable, so you do not hide them entirely, but you often have to hide specific features 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. A more in depth analysis of the usage of deception for cyber defense can be found in the following paper "Deception used for Cyber Defense of Control Systems" [8].

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 tools will they use to discover your asset?

  When attackers obtain access to one of your system, usually, they do not stop there. They will try to expand their control over other systems. For example, they could use network scanners, check for shared folders, or listen to the network in a passive way.

- What assets will they be interested in?

  If you have important confidential data (e.g. intellectual property) in your organization an attacker may want to steal them. You should place the honeypot near these assets to detect the attack before it happens.

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 it is better if the honeypot responds back to the attacker for two main reasons:

- Each interaction that the attacker does with the honeypot could provide you important information. For example, if they stole some credentials, you may be able to know the account that was compromised. You may also be able to find out what tools they used 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 events 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 is generated from the honeypot. Note that each honeypot produces logs in its 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 is 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. 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 like ARP cache poisoning and MITM. If the attacker intercept the logs, it will know that a system is a honeypot without ever interacting with it.

3. **Ingress**: the data is 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 interactions that are not important or interaction from components of the network that may interact with the honeypot (e.g., scanners)

4. **Destination**: The data is stored in the destination inside a database. You can also 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 you to keep track of the status of each honeypot and alerts you when a honeypot goes down.

3 Honeypots classification

Honeypots can be classified on the basis of different characteristics. The two main classifications are based on the level of interaction and on the goal.

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 one type is better than the others. 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 an 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 recognize this machine as a honeypot.

- *High interaction*: high interaction honeypots allow the attackers to interact 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 (e.g. Linux) and applications (e.g. FTP) without using the emulation software (e.g. 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*: 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 an operational system environment, nor do they implement all details of an application protocol. They offer enough answers to deceive known exploits waiting on specific ports into transmitting their payload. Once this payload has been received, the shellcode can be extracted and analyzed. After that, the honeypot emulates the actions that the shellcode would take 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. Different logging tools are utilized to record every action on the honeypot, which may greatly aid researchers in studying the various advanced tactics employed by an attacker. They mainly consist of high interaction honeypots that are complex to deploy and maintain.

- *Production honeypots*: these are deployed by organizations to mitigate threats and protect themselves from various attacks. They require fewer 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. 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. Port scan identification, attack signature generation, trend analysis, and malware capturing are some examples of tasks performed by these honeypots.

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 ones.

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 [4]. 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 a 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 into 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 it is the most popular mode and it can be easily deployed 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 runs 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 credentials. 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 behaviours 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 can download files 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 files will return.

5.5 A consideration about the previous exercise

Some may think that it 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. So for him, it is already too late. 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 tools it uses. To make Cowrie resemble a legitimate system you have to start by modifying the Cowrie configuration 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 names already 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 is 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 allows 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 provides an emulated file system so when an attacker logins, 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 contains the SSH post login banner.

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

If you created new user credentials in the section 5.6.1 remember to create also the home directory for these users. 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

6.1 Dionaea Architecture

6.1.1 Introduction

As described in [1], Dionaea is the successor of the Nepenthes system and belongs to the low interaction type. The main goal of Dionaea is to trap malwares exploiting vulnerabilities exposed by services offered to a network, the ultimate goal is gaining a copy of the malwares.

Dionaea, like any software, is likely to contain bugs that may be exploited. In order to reduce the effect, Dionaea runs in a limited environment with no administrative access. It emulates services such as FTP, HTTP, MSSQL (Microsoft SQL server), MySQL, SMB, TFTP, SIP and others; these services emulate a vulnerable server running Windows OS. SMB is the most vulnerable service running in Dionaea, which utilizes well-known port 445.

In addition, this honeypot has the following features:

1. It is composed of a modular architecture, embedding Python as its scripting language in order to emulate protocols;
2. It supports IPv6 and TLS (Transport Layer Security);
3. It can obtain a detailed analysis of captured malwares, by sending them automatically to the VirusTotal platform.

6.1.2 Exploitation

Dionaea has to detect and evaluate the payload to be able to gain a copy of the malware. In order to do so, Dionaea uses libemu. Given certain circumstances, libemu can detect shellcode, measure the shellcode, and if required even execute the shellcode [1].

Shellcode measurement/profiling is realized by executing the shellcode in the libemu virtual machine and recording API calls and arguments. In most of the cases shellcode profiling is sufficient,

since the recorded API calls and arguments reveal enough information to get an idea of the attackers intentions. For multi-stage shellcode, where the first exploitation stage of the shellcode would retrieve a second shellcode from the attacker, profiling is not sufficient, as we don't know what will happen from the second stage of the shellcode. In this case we need to make use of shellcode execution: this is basically the same as shellcode profiling, the only difference is that API calls are not recorded, and also the shellcode is allowed to perform certain actions.

After the shellcode profiling has been done, Dionaea has to guess the intention, and act upon it.

6.1.3 Logging

Dionaea offers a logging system which logs all the activities in a clear text file, which can be used for analysis and attack prediction. However, this type of logging to a text file is very chatty, and you do not want to use it unless for debugging or development purposes.

In addition to that, Dionaea uses some internal communication system which is called incidents. Incident handlers (ihandlers) are used to transmit incidents, which contain information about the origin and characteristics of an "attack". An ihandler can register a path for incidents he wants to get informed about, and the paths are matched in a glob like fashion. Therefore logging information using an ihandler is superior to text logging, because you get the information you are looking for, and can write it to a format you choose yourself; in other words, it's a more scalable solution.

The most useful ihandler is the `log_sqlite`, which writes interesting incidents to a sqlite database. In particular, this ihandler is very interesting because it has the capacity to cluster incidents based on the initial attack when getting the data from the database; another interesting one is the `log_json` ihandler, which permits to save the logs in JSON format. JSON logs can then be used, for example, to configure Dionaea to work with ELK stack.

Dionaea can also dump a connection as a bi-directional stream. These bi-directional streams are pretty useful when debugging, as they allow replaying an attack on ip-level. You can specify the directory where these are stored in the configuration file. Default is storing them in the directory `/opt/dionaea/var/lib/dionaea/bistreams`.

6.1.4 Analysis

After Dionaea has obtained a copy of a malware, it can either store the binaries locally or submit the file to some external tools or services (e.g. CWSandbox, Norman Sandbox, VirusTotal, etc.) for further analysis.

6.2 Let's see in practice

6.2.1 Access to Dionaea

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

```
$ ./startup/dionaea.sh
```

At this point you are in the Ubuntu file system. In order to enter in the directory of the honeypot, type:

```
$ cd opt/dionaea
```

Dionaea has already started automatically when the docker container was created. Otherwise, if the machine has not already started, it is possible to start it using the command:

```
$ /opt/dionaea/bin/dionaea
```

The startup options can be displayed adding the `-h` flag. Another interesting one is the `--config=FILE` option, with which you can use FILE as the configuration file; the default behaviour is using the file `/opt/dionaea/etc/dionaea/dionaea.cfg`.

6.2.2 Directory Structure

In the Dionaea directory you can find different folders: those of greatest interest are the `etc` and `var` folders. In the first, you can find the configuration file, the services and ihandlers available, and the services and ihandlers enabled, whereas the second folder contains the log files, the binaries

downloaded by Dionaea, and also the sqlite database. In particular the text log file is at the path `/opt/dionaea/var/log/dionaea/dionaea.log`.

You can see that there is also a python script named `readlogsqltree.py`: this is a python3 script which queries the Dionaea sqlite database for attacks, and prints out all related information for every attack; this tool provides information about the exploited vulnerability, the time, the attacker, information about the shellcode, and the file offered for download (if any). To create such report for your own honeypots activities for the last 24 hours run:

```
$ python3 readlogsqltree.py -t $(date '+%s')-24*3600
/opt/dionaea/var/lib/dionaea/dionaea.sqlite
```

At the moment no connections are printed, because there are not any yet.

6.2.3 Port scan

As a first task let's perform a port scan on the honeypot. This will show us the services running in the honeypot, and it will also reveal a strange result. In the same way done for Cowrie, enter in the attacker machine and run the command:

```
$ nmap -sV 10.0.0.40 -Pn
```

It will take about 2 minutes.

As you can see there are a lot of services running on the machine, and in particular the services running on port 445 and 1433, which are respectively the SMB and the MSSQL, are detected as Dionaea honeypot services. This happens because nmap recognises them from the responses received, which are the default responses of Dionaea. Having said that, if in the implementation of Dionaea these responses are changed, nmap will no longer be able to detect them.

Now, if you go back to Dionaea and execute again the `readlogsqltree.py` script, you will see that many connections are reported: in fact the port scan creates a lot of traffic in order to find running services, and the honeypot has recorded all of it.

6.2.4 SIP OPTIONS scan

If you paid attention to the port scan, you should have seen that there is the SIP service open on port 5060/tcp, so as a second step let's launch a SIP OPTIONS scanning. For the next tasks, in order to simulate attacks, we will use the Metasploit Framework. Inside the attacker's machine, type:

```
$ ./msfconsole
```

to run the Metasploit console.

When Metasploit is ready, use the command `$ search sip options_tcp` to search the right module; after this use the command `$ use 0` to use the module found. At this point you have to set the target host: to do this type `$ set rhost 10.0.0.40`. Finally, run `$ exploit` to launch the scan.

Now go back to Dionaea and execute the python script another time. You should see that the last reported connection is a SipSession tcp connection. In this case the attack does not trigger any vulnerability, because it is only a SIP OPTIONS scan. This type of scanning is executed to spot VoIP devices and determine their capabilities. It might be a reconnaissance step of a multi-stage attack. These scans are used to identify targets and the next stage will likely target only those devices which responded during scanning. [3]

6.2.5 SMB exploitation

The next task is to try to exploit a vulnerability on the SMB protocol on port 445. As already said, this is the main protocol offered by Dionaea. SMB has a decent history of remote exploitable bugs, and is a very popular target for worms. The vulnerability that will be attempted to exploit is that detailed in Microsoft Bulletin MS10-06; it is a vulnerability in the Print Spooler service. The vulnerability could allow remote code execution if an attacker sends a specially crafted print request to a vulnerable system that has a print spooler interface exposed over RPC.

Inside the Metasploit console, use the command `$ search ms10-061` to search the right module; after this use the command `$ use 0` to use the module found. At this point you have to set the printer share name to use on the target host: to do this type `$ set pname XPSPrinter`. After this

step you have to set the target host: to do this type `$ set rhost 10.0.0.40`. Finally, run `$ exploit` to launch the exploit.

You can see the exploit completed, but no remote code execution session was created; this happens because Dionaea is a low interaction honeypot. However, if you go back to Dionaea and run the python script, you can see that there has been a connection from the attacker on the smb protocol; also you can see that, under the "download" section, it is reported that something has been downloaded (together with the download URL). To check what has been downloaded, enter in the binaries folder, typing:

```
$ cd /opt/dionaea/var/lib/dionaea/binaries
```

Here you can find two files: one is the MD5 hash value of the corresponding malware, calculated by Dionaea. This MD5 hash value is used for analysis purposes: you can either submit the hash value to VirusTotal or you can submit it to any antivirus software. You can consider this hash value as a feature of a malware [5]. The other downloaded file is a tmp file.

6.2.6 MSSQL server login

Another interesting task that you can try to perform is to log into the Microsoft SQL server, which runs on port 1433. To do this, you will again use Metasploit. Inside the Metasploit console, run the command `$ search mssql_login` to search the right module; after this run the command `$ use 0` to use the module found. There is a default username already set, which is '`sa`': this is the administrative username on the Microsoft SQL server. At this point you have to set the password. You can choose whatever you want, indeed the login will be always successful and the task is only performed in order to see the result on the honeypot. So, for example, use the command `$ set password pass` to set the password as `pass`. Next you have to set the target host: in order to do this run `$ set rhost 10.0.0.40` as in the previous tasks. Finally, run `$ exploit` to launch the exploit. The result is that you logged in successfully.

Now return to Dionaea and run the python script again. You can see the last connection, which was on the mssql service, and also the username and the password used for the login. In this way, in a real scenario, it is possible to observe which are the most popular names used by hackers, and finding a way to substitute them with unknown ones will elevate the system's security [11].

The same approach of login could be used for the FTP or MySQL servers.

6.2.7 VirusTotal analysis

If you want to enable Virustotal to analyse the malwares you capture, you first need a Virustotal API key. To obtain it, you have to go to the website `https://www.virustotal.com/` and sign up; in this way you will automatically obtain a key. Once you have the API key, you have to go to the folder `/opt/dionaea/etc/dionaea/ihandlers-available` and modify the ihandler file named `virustotal.yaml`; here you have to paste your API key in the indicated place. In this case there is already a pasted key, which is the key utilized by us. After this, save and exit. Next change the directory to `/opt/dionaea/etc/dionaea/ihandlers-enabled`. Now, you need to create a symbolic link here (which has already been created) to the edited `virustotal.yaml` file typing:

```
$ ln -s ../ihandlers-available/virustotal.yaml virustotal.yaml
```

After following this procedure, in theory your new captured malware should now be identified, and you should be able to look at the analysis running the python script used for the previous tasks. In practice however, there is actually a bug in the current version of Dionaea for which the feature does not work (but this bug will certainly be fixed soon).

Also, in the directory `/opt/dionaea/var/lib/dionaea` there is a sqlite database named `vtcache.sqlite`. This is used to cache the results of the performed analyzes.

7 Honeyfiles

A honeyfile is an intrusion detection mechanism based on deception. This mechanism works by creating a bait file that when accessed sets off an alarm or logs 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 types of files [16]:

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

Then you have to think about 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 too many honeyfiles, because it will also increase the chance of triggering false positives by legitimate users. And also do not try too hard with the naming conventions otherwise the attacker will become suspicious.

7.1 Honeydoc

A honeydoc is a particular type of honeyfile. In particular, it is an office document that contains a token used for tracking purpose (called web bug or web beacon). This token references a resource from an external URL. In this way when the document is opened, the external resource will be requested and the server serving this resource will log the interaction. In the next exercise we will hide this web bug by shrinking it into a 1x1 pixel image.

7.2 Create a LibreOffice honeydoc

For this laboratory exercise there is already an Nginx server running that serves an image that you can use as a web bug. The page is accessible by opening Firefox on the virtual machine and opening the page at the URL `localhost:8080`. It is also possible to access the Nginx instance to see its logs. Just open a terminal and execute the commands:

```
$ docker exec -it honeydocserver /bin/bash
$ tail -f /var/log/nginx/access.log
```

Now you can create the honeydoc. First, open LibreOffice and create a new document. Then follow these steps:

1. Create a footer by clicking at the bottom of the page. By placing the web bug there, is harder to be accidentally found or deleted
2. Click on the Insert tab. Then insert an image. Now select just a temporary image and then select the box "Insert as a link".
3. LibreOffice will ask if you really want to include a link. You have to choose "Keep link"
4. Now right-click on the image. And click on properties.
5. In the first tab resize the image as small as possible (0.2")

6. In the "option" tab check the protect content, position and size boxes.
7. In the "image" tab modify the field "File Name" to reference the remote image present on the Nginx server and click OK. After these steps, the image should be hard to spot. To still be able to see it we choose a black image but by using a white or transparent image it will be very difficult to detect.
8. Add some content to the document and save it with a relevant name (e.g. PASSWORDS.txt)

Now every time we open the document, Office will attempt to load the image from the URL, thus generating requests to our web server that will log the interaction.

7.3 Another way to create honeytokens

An easy way to generate honeytoken without repeating the above procedure is provided by Thinkst with their service called Canary Tokens. You can access it by going to the website `https://canarytokens.org`. You will just need to specify the type of token, an email address where to receive the alerts and a reminder note for when the token is triggered. Then you can download the honeytoken document and place it where it is needed.

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**: 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, stores 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

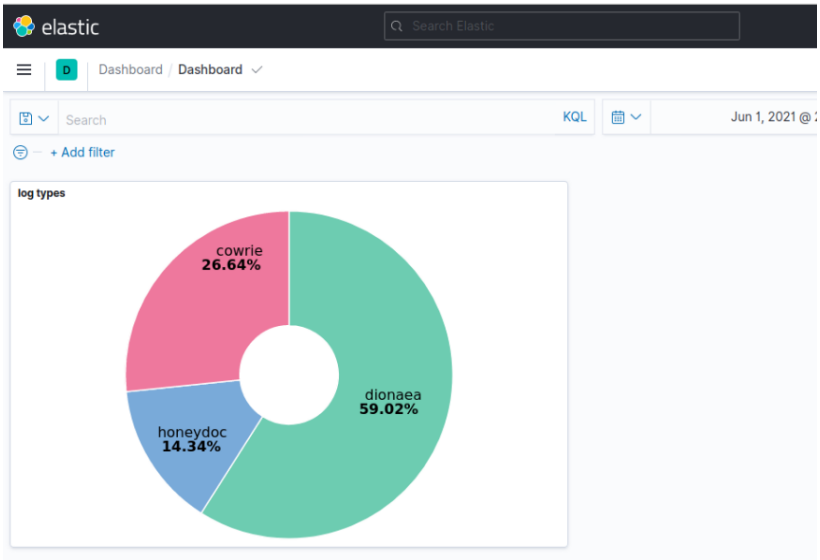
8.1 Exercise - Create simple dashboards in Kibana

Now that we have multiple honeypots running that generated multiple events we can log into Kibana to see the logs from a central location.

To do that, open Firefox and go to the URL `http://localhost:5601` and login with username: `netsec` and password: `password`.

After logging in, open the menu on the left side of the screen and click on Recently viewed -¿ raw logs. On this page, you should be able to see all the logs received and stored in the elastic search database. You can use the search bar to filter for specific values and modify the time frame of the logs that are shown by specifying a time interval in the input on the right of the search bar.

To create a dashboard click on the menu on the left side of the screen and click on New Dashboard. Now click on the button "Create panel" and select the panel Lens. Now select a relevant field you are interested in and drag it into the center of the screen. For example you can choose the field `Fields.document_type.keyword` that tell from what honeypot the log was generated. Then select a visualization type (e.g. donut) and click on the button "Save and return" to save the visualization. Once you are done, you can save the dashboard by clicking on "Save" in the right top corner of the screen.



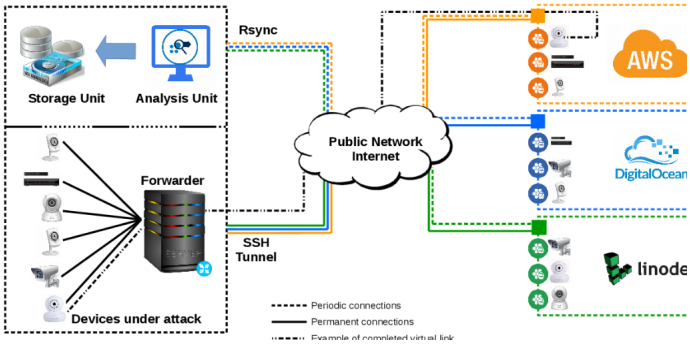
9 Honeypots for IoT

The tremendous growth in the interconnection of Internet of Things (IoT) devices has also increased the possibility of data breach. Thus, there is a need to protect the connected devices or a system from disclosure or network-based attack. Recent Distributed Denial of Service (DDoS) attacks which used inadequately secured IoT devices, further highlight that existing defenses are slow to detect zero day exploits and capture attack traffic. So here's the necessity for countermeasures and honeypots have long proven effective in capturing malware, helping to counter spam and providing early warning signals about upcoming threats [10].

Below are reported some recent IoT honeypots that have been developed in order to face this security problem.

9.1 SIPHON

SIPHON is a prototype of a scalable and reliable high interaction honeypot with the aim to learn about existing and novel attack vendors, be it human or automatic, targeting IoT devices. By constructing tunnels between public IP addresses and physical devices utilizing the entrypoint (wormhole), SIPHON enables the portrayal of physical IoT devices in a broad variety of geographic locations. The traffic data is gathered in order to be processed and analyzed further. The figure below represents the SIPHON architecture with IP cameras and a networked video recorder (NVR) linked to three different cloud servers [7].



9.2 HIoTPOT

HIoTPoT functions as a camouflaged monitoring camera on the attackers, diverting them from their goals by keeping them busy in a fake environment.. It will gather any important information about the attackers. When it notices an attacker, HIoTPoT sends out notifications to all nearby IoT devices in the network, telling them not to communicate with that attacker, and so keeping the active environment secure.. HIoTPoT can act as a production honeypot as well as a research honeypot. The HIoTPOT will maintain a database for all authenticated users. Whenever a user connects to the IoT network, HIoTPOT will check that user against the database. If it finds the user in the database, then it will permit the user to enter in the real IoT environment. If it does not find the user in the database, then it will send the user to a fake IoT environment which is only an image of the real IoT environment. The intruder then believes he has gained access to the IoT infrastructure and begins interacting with IoT devices. At the same time the HIoTPOT will keep one database that will record all the intruder's communications information [6].

9.3 IoTPOT

IoTPOT is a medium-interaction honeypot which emulates Telnet services of IoT devices to analyze ongoing attacks. IoTPOT has a function that automatically exeutes an active scanning of the attacking IP addresses to obtain their banner profiles. IoTPOT design is composed of a front-end low-interaction responder and back-end high-interaction virtual environment called IoTBOX which works as a malware sandbox. This IoTBOX manages various virtual environments and supports different CPU architectures [9].

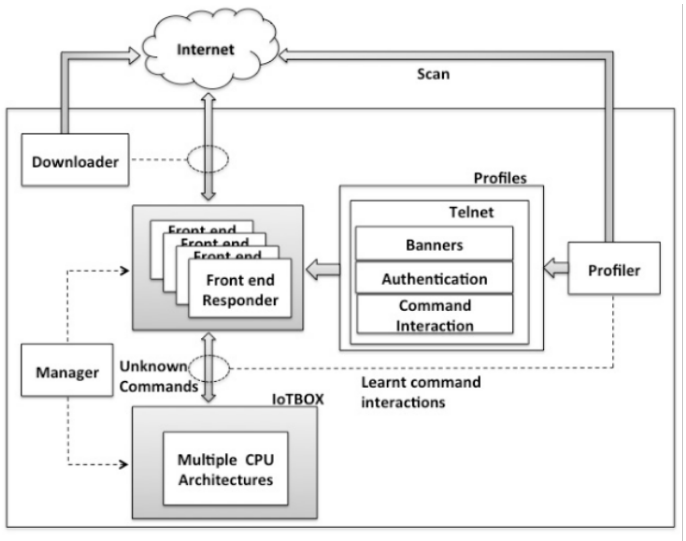


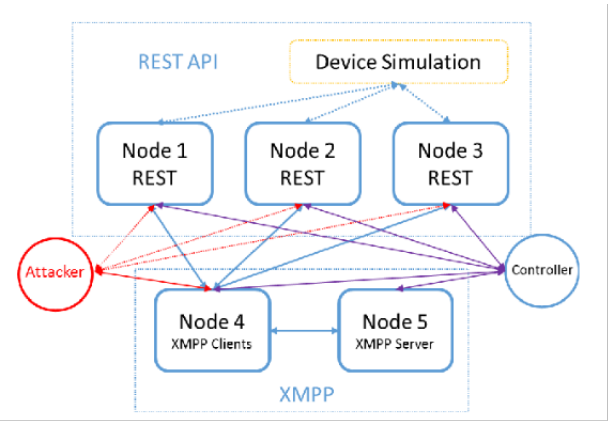
Figure 3 - Overview of IoTPOT

9.4 ThingPot

ThingPot is a honeypot implementation of XMPP protocol (XMPP is a communication protocol that provides basic instant messaging (IM) and presence functionality) which focuses on the use case of

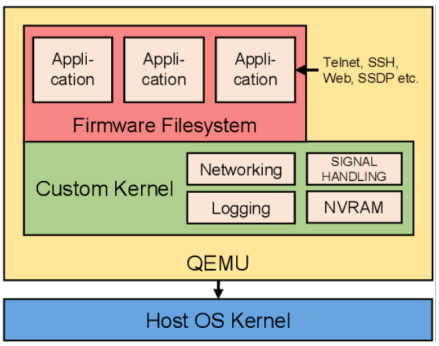
Philips Hue (Philips Hue consists of wireless LED light bulbs and a wireless bridge).

It can simulate an entire IoT platform rather than a single application layer communication. Additionally, the ThingPot's source code is open to the public and is available online at Github; this includes two modules: client and server. To reduce the chances of an attacker gaining access to the real system, the ThingPot runs in an isolated simulated environment.. REST is used to build a backend API. Given its popularity, XMPP has been selected as the IoT protocol for real-time communication. A basic HTTP web service is used to implement the frontend. [14].



9.5 Honware

As discussed in [13], this is the first flexible and generic framework to efficiently and effectively deploy honeypots for networked devices on the Internet to log attacker traffic and their actions. Instead of using real devices, honware utilises device firmware images (which are largely available for download) and a special prebuilt Linux kernel to emulate device behaviour within a virtual environment. The performance of honware is equivalent to that of actual devices, and it is also immune to trivial timing-based fingerprinting attacks. Furthermore using firmware images is cheaper and therefore more scalable. Below is represented the honware architecture overview.



Bibliography

[1] Dionaea documentation. https://dionaea.readthedocs.io/en/latest.

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

[3] Honeypots cert exercise handbook - enisa.

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

[5] P Dilsheer Ali and T. Gireesh Kumar. Malware capturing and detection in dionaea honeypot. pages 1–5, 2017.

[6] Usha Gandhi, Priyan M K, Ramachandran Varatharajan, Gunasekaran Manogaran, Revathi Sundarasekar, and Shreyas Kadu. Hiotpot: Surveillance on iot devices against recent threats. *Wireless Personal Communications*, 103, 11 2018.

[7] Juan Guarnizo, Amit Tambe, Suman Sankar Bhunia, Martín Ochoa, Nils Tippenhauer, Asaf Shabtai, and Yuval Elovici. Siphon: Towards scalable high-interaction physical honeypots, 2017.

[8] Miles A. McQueen and Wayne F. Boyer. Deception used for cyber defense of control systems. pages 624–631, 2009.

[9] Yin Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. Iotpot: A novel honeypot for revealing current iot threats. *Journal of Information Processing*, 24:522–533, 05 2016.

[10] Mohamad Faiz Razali, Muhammad Nazim Razali, Fawwaz Zamir Mansor, Gopinath Muruti, and Norziana Jamil. Iot honeypot: A review from researcher's perspective. pages 93–98, 2018.

[11] Koki Saikawa and Vitaly Klyuev. Detection and classification of malicious access using a dionaea honeypot. 2:844–848, 2019.

[12] C. Sanders. *Intrusion Detection Honeypots: Detection Through Deception.* Applied Network Defense, 2020.

[13] Alexander Vetterl and Richard Clayton. Honware: A virtual honeypot framework for capturing cpe and iot zero days. pages 1–13, 2019.

[14] Meng Wang, Javier Santillan, and Fernando Kuipers. Thingpot: an interactive internet-of-things honeypot, 2018.

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

[16] Jim Yuill, Michael Zappe, Dorothy Denning, and F. Feer. Honeyfiles: Deceptive files for intrusion detection. pages 116 – 122, 07 2004.