Network
Security Lab
2020/2021

Honeypot

Giovanni Da Rold
Marcello Meschini

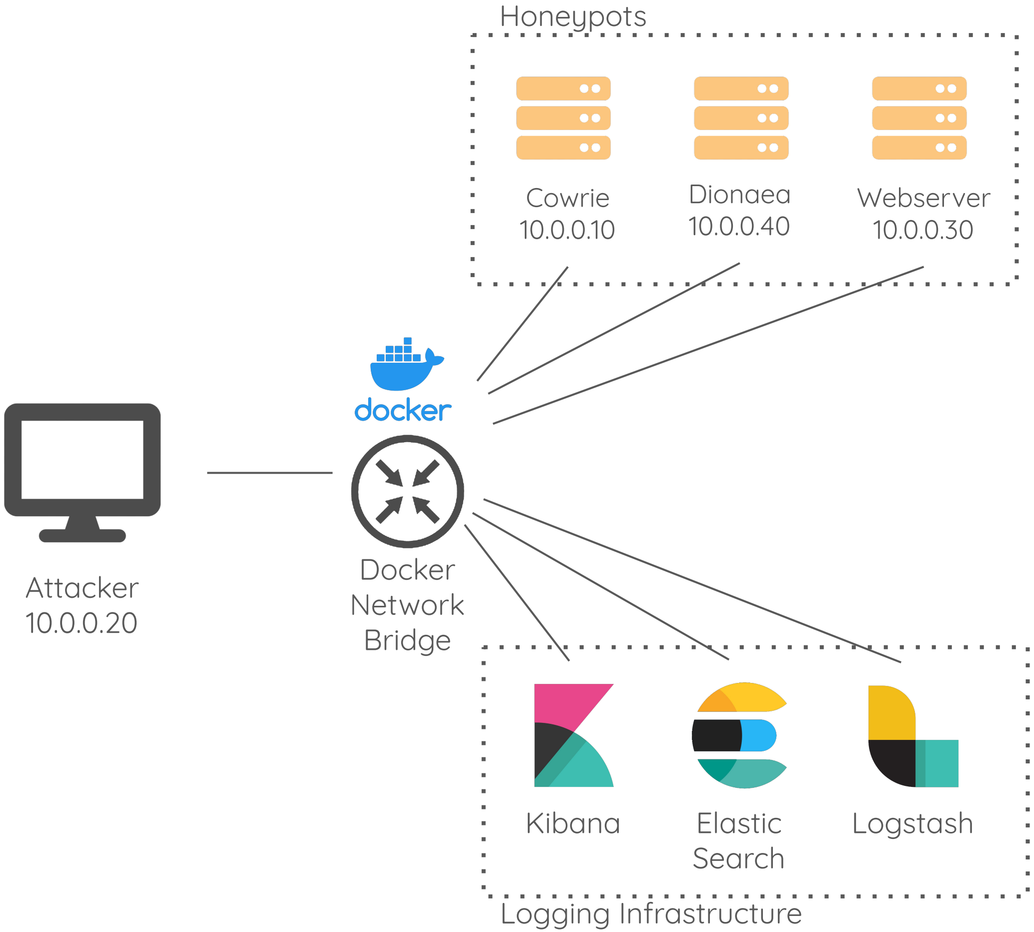
Outline

- Network topology
- What is a honeypot?
- The characteristics of honeypots
- Logging and monitoring infrastructure
- Honeypots classification
- Advantages of honeypots
- Cowrie
- Dionaea
- Honeyfiles
- Lab monitoring infrastructure
- Honeypot for IOT

Network topology

We created a docker network with the following containers:

- Cowrie 10.0.0.10
- Attacker 10.0.0.20
- Honeydoc server 10.0.0.30
- Dionaea 10.0.0.40
- Elasticsearch
- Logstash
- Kibana

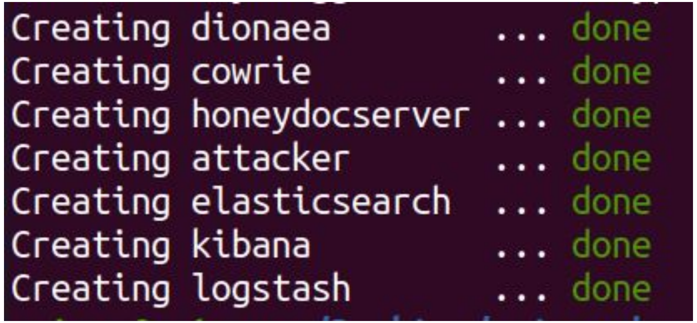


Starting up the lab

- Open the VM
- Credentials:
  username: *netsec*
  password: *password*
- Open a terminal in the folder netsec-honeypot-lab on the desktop
- Execute the start.sh script:

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

You should see this output



```
Creating dionaea       ... done
Creating cowrie        ... done
Creating honeydocserver ... done
Creating attacker      ... done
Creating elasticsearch  ... done
Creating kibana        ... done
Creating logstash      ... done
```

Shutting down the lab

Type in the terminal:

```
$ docker-compose down
```

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”

Characteristics of honeypots

01. Deceptive

02. Discoverable

03. Interactive

04. Monitored

01. Deception

Deception = advantageous distortion of an adversary’s perceptions of reality

According to the taxonomy proposed by Bell and Whaley it consists of two parts:

1. Hiding the real
   - **Masking**: by making relevant objects be undetectable or blend into the background
   - **Repackaging**: by making it appear like it is something different
   - **Dazzling**: by altering an object to confuse the adversary
2. Showing the false
   - **Mimicking**: by using characteristics present in the actual real object
   - **Inventing**: by giving the perception that a relevant object exist while it does not
   - **Decoying**: by misdirecting and attracting the attacker attention away from real objects

02. Discoverability

- Honeypots are not meant to be accessed by legitimate users
- But they have to be discoverable in the right context

You have to think like an attacker when designing your honeypot

- Where the attacker is likely to enter your network?
- What tool will they use to discover your assets?
- What assets will they be interested in?

You can increase discoverability by placing **breadcrumbs**

03. Interactivity ⇄

- The attacker at some point will interact with the honeypot
- Technically a single interaction is enough to produce an alert
- But it is better if the honeypot responds back to the attacker

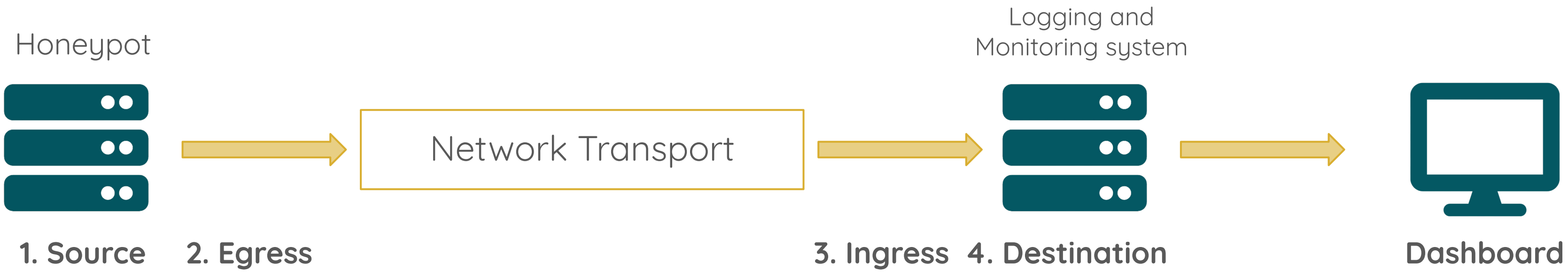
  Why?
  - Each interaction that the attacker does with the honeypot could provide you important information
  - You make the attacker waste time

03. Monitoring

- Every notable interaction with a honeypot must generate an event or an alert
- NB: An alert does not always mean that an attacker is trying to compromise the network
- But checking every honeypot by hand is not scalable in an environment with hundreds of honeypots

Logging and monitoring infrastructure

Allows to transfer the logs produced by the honeypot to a central system



Honeypots classification

Based on
level of interaction

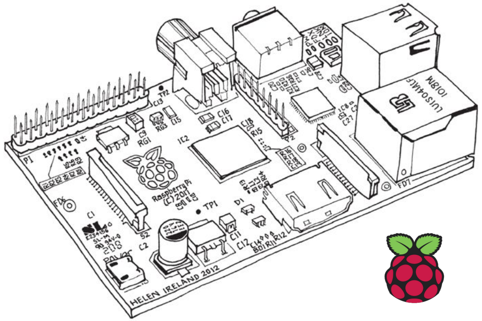
- Low interaction
- High interaction
- Medium interaction

Based on the goal

- Research honeypots
- Production honeypots

Advantages of honeypots

- They consume few resources, especially if compared to traditional IDS sensors
- They are simple and cheap to maintain
- They produce a small amount of logs with few false positives



Cowrie

- Is an open-source medium interaction SSH and telnet honeypot
- It can run in two modes:
    - Medium interaction (shell)
    - High interaction (proxy)

Let’s enter the honeypot by typing in the terminal:

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

You can start cowrie by typing:

```
$ cowrie start
```

We can already check the honeypot logs:

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

Now open the attacker shell with the command:

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

Attacker point of view

- The attacker successfully installed nmap on the compromised machine and identified a host with the IP 10.0.0.10
- He executes a TCP service probe scan against this machine:

```
$ nmap -sV 10.0.0.10
```

- Finds the service SSH running on port 2222
- Then he tries to connect using common credentials

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

```
$ password
```

- Try to play a bit inside the honeypot. Tell me some strange behaviour you encountered

A consideration

- You may think that it is not that hard to figure out that the system is a honeypot...
- But when the attacker is inside the honeypot, for him it is already too late
- Keep in mind that you logged every character that he typed!

Let’s execute this command:

`$ bin/playlog var/lib/cowrie/tty/` and select a session log

And if you used “wget” inside the honeypot check the folder:

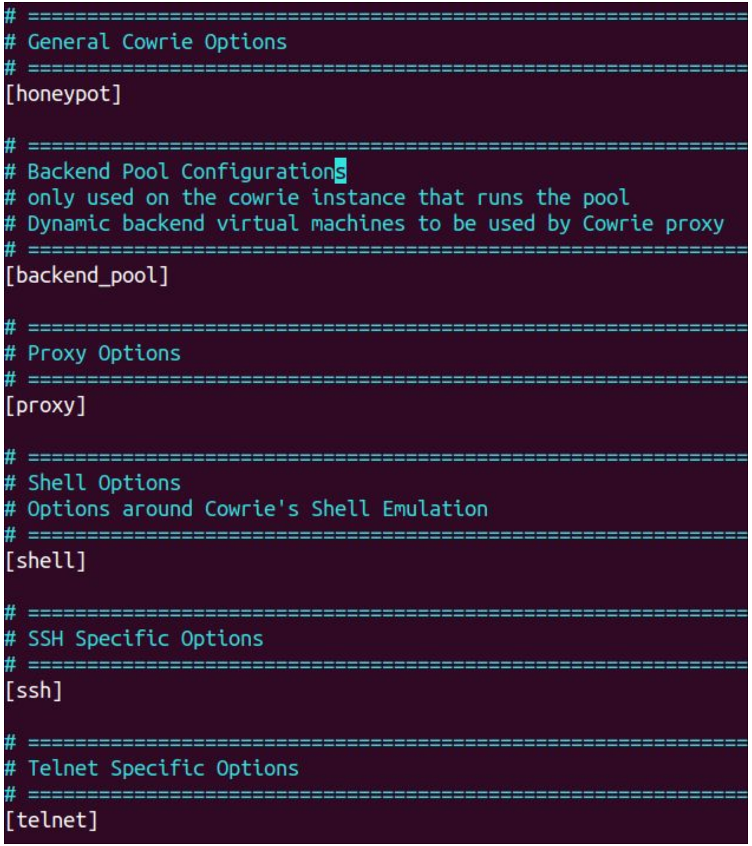
```
var/lib/cowrie/downloads/
```

Customizing Cowrie

- We want to make Cowrie resemble a legitimate system
- To do that we have to modify the Cowrie config files (located in etc folder)
- First let's copy the default config file to a file named cowrie.cfg

```
$ cp cowrie.cfg.dist cowrie.cfg
```

- We can now modify it:
  - change hostname
  - modify the timeout for the SSH session
  - modify the honeypot SSH port -> search listen_endpoint in the [shell] section



```
# ============================================================
# General Cowrie Options
# ============================================================
[honeypot]

# ============================================================
# Backend Pool Configurations
# only used on the cowrie instance that runs the pool
# Dynamic backend virtual machines to be used by Cowrie proxy
# ============================================================
[backend_pool]

# ============================================================
# Proxy Options
# ============================================================
[proxy]

# ============================================================
# Shell Options
# Options around Cowrie's Shell Emulation
# ============================================================
[shell]

# ============================================================
# SSH Specific Options
# ============================================================
[ssh]

# ============================================================
# Telnet Specific Options
# ============================================================
[telnet]
```

Modifying the login credentials

- Copy the default password file “userdb.example” in a file named “userdb.txt”

```
$ cp userdb.example userdb.txt
```

- More information are provided inside the file
- Tip: it is better to use generic credentials that can be easily guessed but without raising suspicion

Modifying the process list

- Cowrie allows you to implement a list of fake processes
- They are inside a JSON file in:

```
$ cat share/cowrie/cmdoutput.json
```

- Tip: remove processes that are not likely to appear in your system and add processes that are running on a real systems

Modifying the file system

- The emulated file system is implemented in two components:
  - A **pickle file**: contains metadata for the files (their name, size, permission, directory, ...)
  - The **honeyfs directory**: contains the file contents

    NB: to be visible a file must be in the pickle file, and to be also accessible it needs to be in the honeyfs directory

- Let’s modify the filesystem that is already present. For example let’s open the file: `honeyfs/etc/motd`
- We can also add a file in the bin directory, we will use it later
- The pickle file is located at the following path: share/cowrie/fs.pickle. And it is generated automatically using the command:

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

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

- Cowrie will handle the rest

Let’s check if everything works

Dionaea

- Low interaction honeypot, successor of Nephentes system
- The main goal of Dionaea is to trap malwares exploiting vulnerabilities exposed by services, the ultimate goal is gaining a copy of the malwares
- It emulates services such as FTP, HTTP, MSSQL (Microsoft SQL server), MySQL, SMB, TFTP, SIP and others
- It features a modular architecture, embedding Python as its scripting language in order to emulate protocols
- It supports IPv6 and TLS (Transport Layer Security)

Exploitation

- Dionaea uses libemu, which can detect shellcode, measure the shellcode, and if required even execute the shellcode
- Shellcode profiling is done by running the shellcode in the libemu vm and recording API calls and arguments
- Once we got the profiling and payload, Dionaea has to guess the intention, and act upon it
- Common techniques used by an attacker
  - *Shells - bind/connectback*
  - *URLDownloadToFile*
  - *Multi-Stage Payloads*

Logging

- Dionaea offers a logging system which logs all the activities in a clear text file

  N.B. Dionaea's logging to text files is really chatty

- Dionaea also uses some internal communication system which is called incidents; these are managed by handlers (ihandler)
- Logging information using an ihandler is superior to text logging, because you get the information you are looking for => more scalable solution
- Dionaea can also dump a connection as bi-directional stream

Analysis

Once you’ve obtained a copy of a malware, you have the option to 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

Let’s see it in practice...

In order to access the honeypot machine launch a terminal window, 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
```

Directory structure

In the dionaea directory you can find different folders: those of greatest interest are the ‘etc’ and ‘var’ folders

The ‘etc’ folder contains the configuration file (dionaea.cfg), the services and ihandlers available, and the services and ihandlers enabled

The ‘var’ folder contains the log files, the binaries downloaded by Dionaea, the sqlite database and other files

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

To create such report for your own honeypots activities for the last 24 hours run:

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

Port scanning

Let’s perform a port scan on the honeypot

If you’re not already in, enter in the attacker machine and run the command:

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

This will detect the open ports, and which services are running there

When it’s done, go back to Dionaea and execute again the readlogsqltree.py script: you will see that many connections are reported

SIP OPTIONS scanning

Using the port scanning you 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

Inside the attacker’s machine, type `$ ./msfconsole` in order to run the Metasploit Framework console

Next 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

SIP OPTIONS scanning

Now go back to Dionaea and execute again the python script

Check that the last reported connection is a SipSession TCP connection; you can also see that the method OPTIONS has been used

In this case the attack does not trigger any vulnerability. It is a SIP OPTIONS scan

It might be a reconnaissance step of a multi-stage attack

SMB exploitation

The next task is to try to exploit a vulnerability on the SMB protocol on port 445

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

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

Next 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: `$ set rhost 10.0.0.40`

Finally, run `$ exploit` to launch the exploit

SMB exploitation

You can see the exploit completed, but no remote code execution session was created; this happens because Dionaea is a low interaction honeypot

Go back to Dionaea and run again the python script: you can see that there has been a connection from the attacker on the smb protocol

Now go to the binaries folder typing:

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

and check what has been downloaded

MSSQL Server Login

Microsoft® SQL Server®

Another interesting task that you can try to perform is to log into the Microsoft SQL Server: this server runs on the port 1433

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

At this point you have to set the password (the username is already set as ‘sa’), so for example run `$ set password pass` to set it as ‘pass’

Next you have to set the target host: `$ set rhost 10.0.0.40`

Finally, run `$ exploit` to launch the exploit

MSSQL Server Login

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 it is possible to observe which are the most popular names used by hackers

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

VirusTotal Analysis

VIRUSTOTAL

With Dionaea is possible to automatically submit the captured malware samples to the VirusTotal service for further analysis

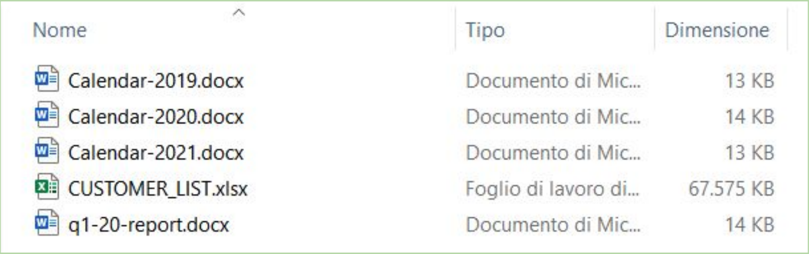
If you want to use Virustotal to identify the malwares you capture, you first need a Virustotal API key. You just need to sign up with Virustotal to get one

Once you have the API key, you have to enable the VirusTotal ihandler and paste the API key in the ihandler file named ‘virustotal.yaml’

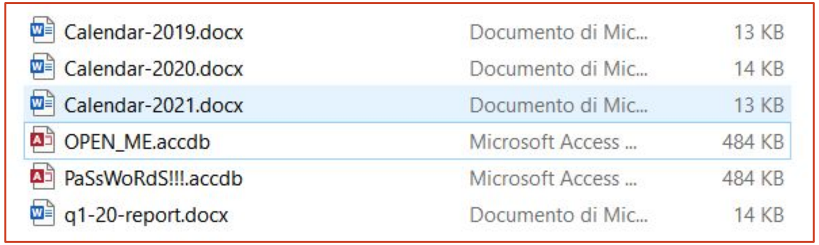
Your new captured malware should now be identified

Honeytokens / Honeyfiles

- The honeypots that mimic legitimate services are not the only ones
- Honeytoken honeypots works by creating a bait file that when accessed sets off an alarm or logs the interaction
- Usually an attacker is interested in:
  - File containing important information
  - File containing personal information
  - Information for accessing other systems
  - Log files that can contain evidence of attacks
- Tips for designing a honeytoken:
  - Place the file where it can be easily found
  - Make the file standout from the others
  - Do not exaggerate



Nome	Tipo	Dimensione
Calendar-2019.docx	Documento di Mic...	13 KB
Calendar-2020.docx	Documento di Mic...	14 KB
Calendar-2021.docx	Documento di Mic...	13 KB
CUSTOMER_LIST.xlsx	Foglio di lavoro di...	67.575 KB
q1-20-report.docx	Documento di Mic...	14 KB



Calendar-2019.docx	Documento di Mic...	13 KB
Calendar-2020.docx	Documento di Mic...	14 KB
Calendar-2021.docx	Documento di Mic...	13 KB
OPEN_ME.accdb	Microsoft Access ...	484 KB
PaSsWoRdS!!!.accdb	Microsoft Access ...	484 KB
q1-20-report.docx	Documento di Mic...	14 KB

Honeydocs

- In particular a honeydoc: is an office document that contains a token for tracking purpose (we will use a **web bug** or **web beacon**)
- A Nginx web server containing our image is already configured:

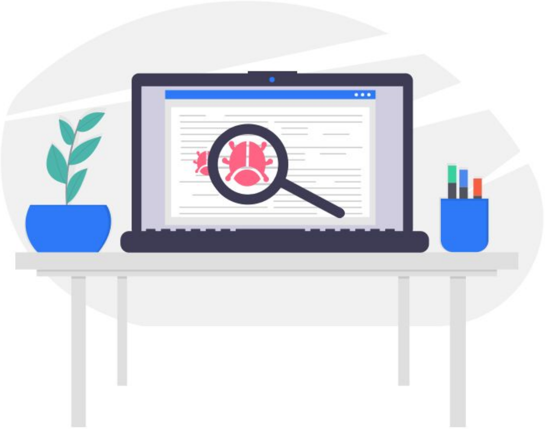
Open Firefox and go to: localhost:8080

Now let’s access the server by typing:

```
$ docker exec -it honeydocserver /bin/bash
```

And see its logs:

```
$ tail -f /var/log/nginx/access.log
```



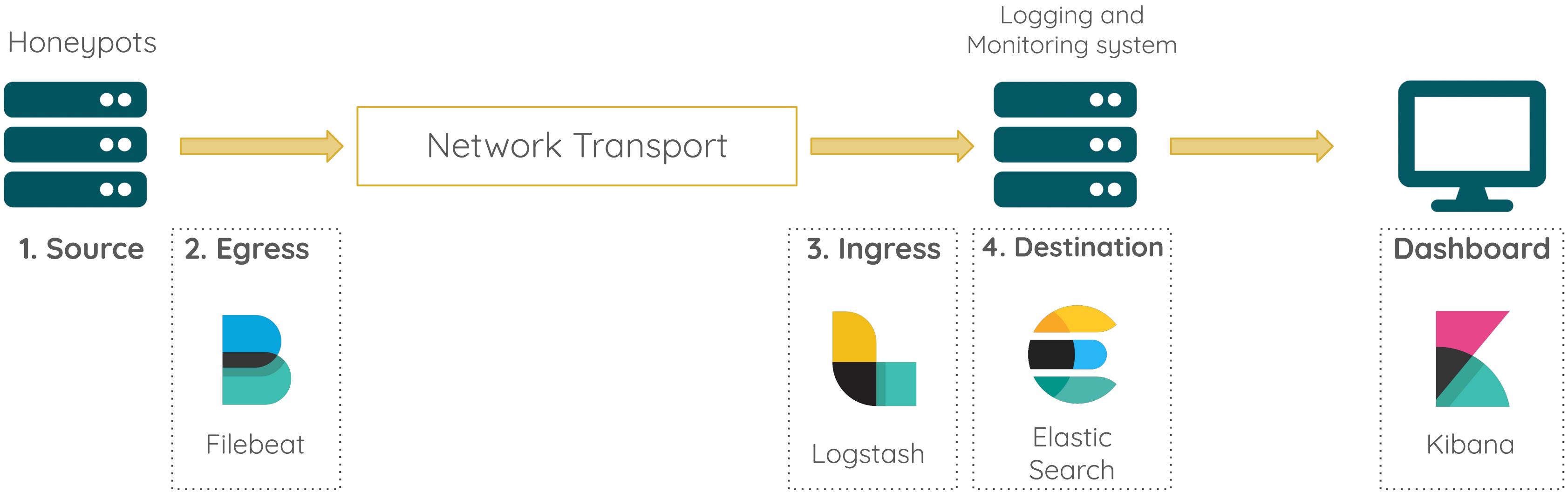
How to create a honeydoc

Open LibreOffice and create a new document. Then:

1. Create a footer
2. Insert tab -> insert image -> select a temporary image -> check Insert as Link
3. Right click on the image -> properties
4. Resize the image as small as possible -> (0.02)
5. Option tab, check protect contents, position, and size
6. In the Image tab and in the field File Name input the remote reference to the image in the logging web server and click OK
7. Add some content and save the document with a relevant name

Another way to generate honeytokens: https://canarytokens.org/generate

Lab monitoring infrastructure



Let’s log into Kibana and create a simple dashboard.

Go to localhost:5601 and login with the credentials: *netsec / password*

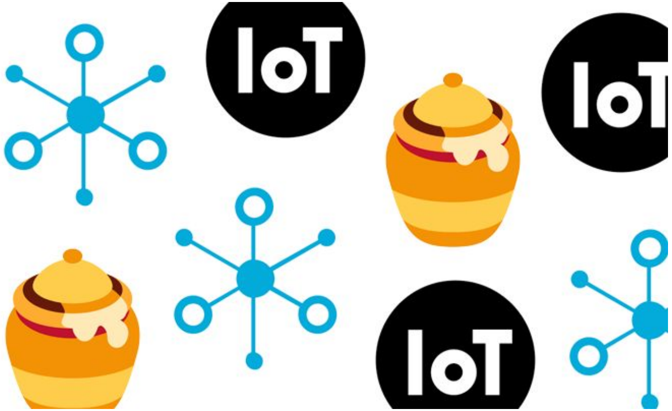
Honeypots for IoT

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 attacks

Honeypots have long proven effective in capturing malware, helping to counter spam and providing early warning signals about upcoming threats

In the next slides are reported some recent IoT honeypots that have been developed in order to face this security problem



SIPHON

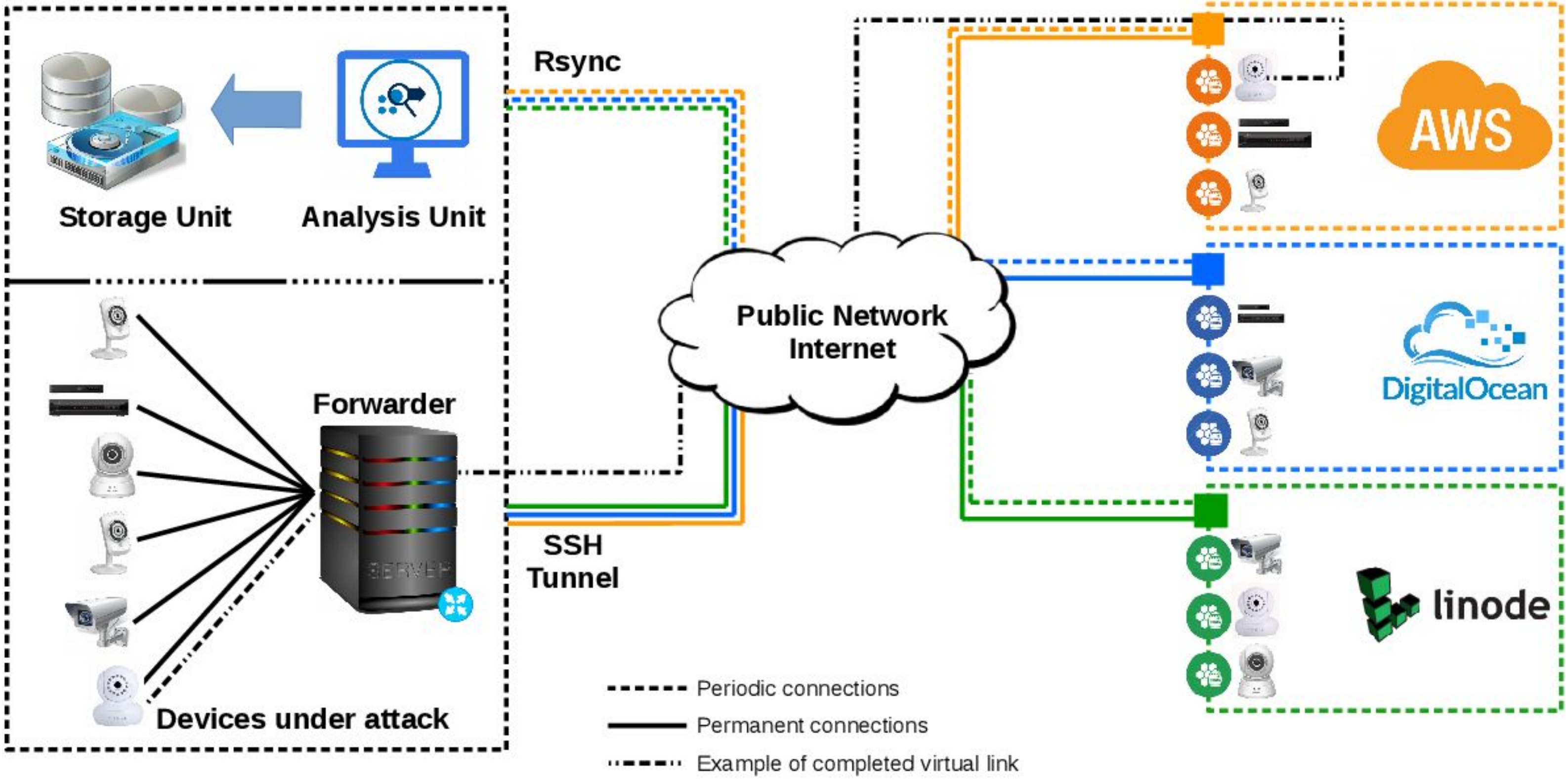
- A prototype of scalable and reliable HIH with the aim to learn about existing and novel attack vendors, be it human or automatic, that target IoT devices

- It allows the portraying of physical IoT devices in a wide range of geographic locations, by establishing tunnels between the public IP addresses and the physical devices using the entrypoint (wormhole)

- The traffic information is collected for further processing and analysis

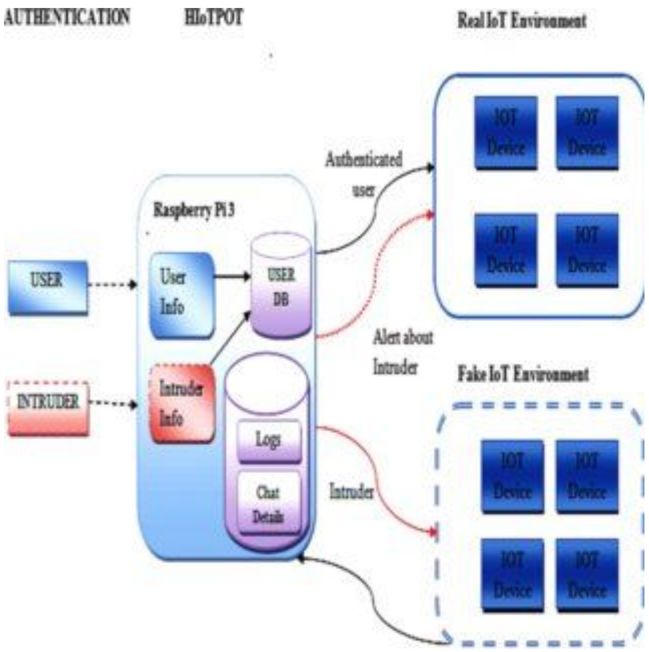
Guarnizo et al., “SIPHON: Towards Scalable High-Interaction Physical Honeypots”, 2017

SIPHON


Storage Unit
Analysis Unit
Rsync
Forwarder
Devices under attack
SSH
Tunnel
Public Network
Internet
AWS
DigitalOcean
linode
Periodic connections
Permanent connections
Example of completed virtual link

HIoTPOT

- HIoTPoT acts as a camouflage monitoring camera on the attackers by making them busy in the fake environment and distracting them from their aims
- Once the attacker is detected, HIoTPoT generates alerts to the surrounding IoT devices in the network about not communicating with that attacker
- The HIoTPOT will maintain a database for all authenticated users. Whenever any user is going to access the IoT network the HIoTPOT will compare that user with the database
- If the user is present in that database, then it will allow the user to the real IoT environment, otherwise it will send the user to a fake IoT environment



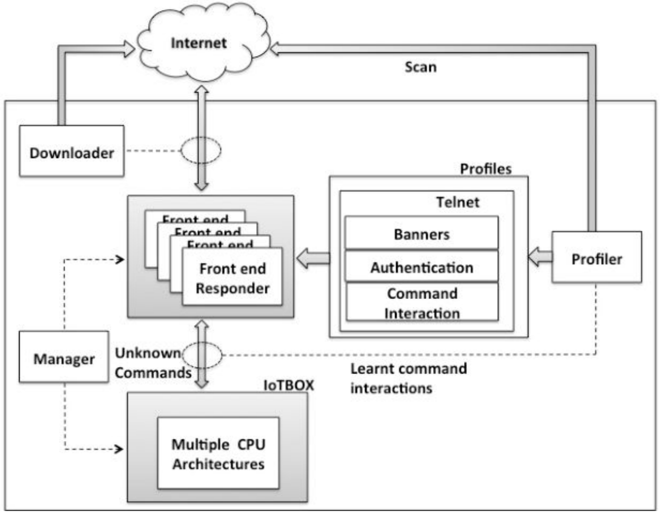
Gandhi et al., “HIoTPOT: Surveillance on IoT Devices against Recent Threats”, 2018

IoTPOT

- IoTPOT is a medium-interaction honeypot that emulates Telnet services of IoT devices to analyze the ongoing attack

- IoTPOT is designed with a combination of a front-end low-interaction responder and back-end high-interaction virtual environment known as IoTBOX, which acts as a malware sandbox

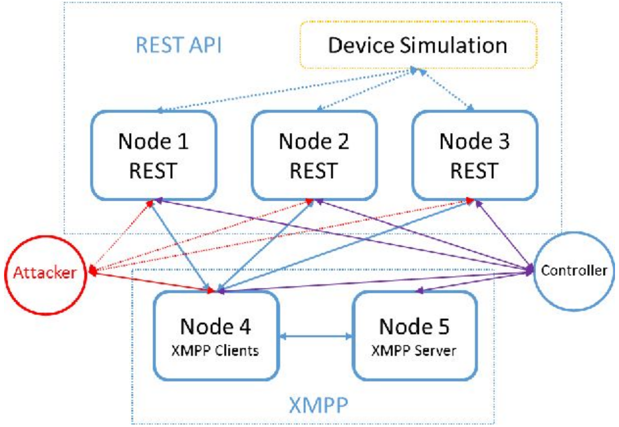
- This IoTBOX operates various virtual environments usually used by the embedded systems for different CPU architectures



Yoshioka et al., “IoTPOT: A novel honeypot for revealing current IoT threats”, 2016

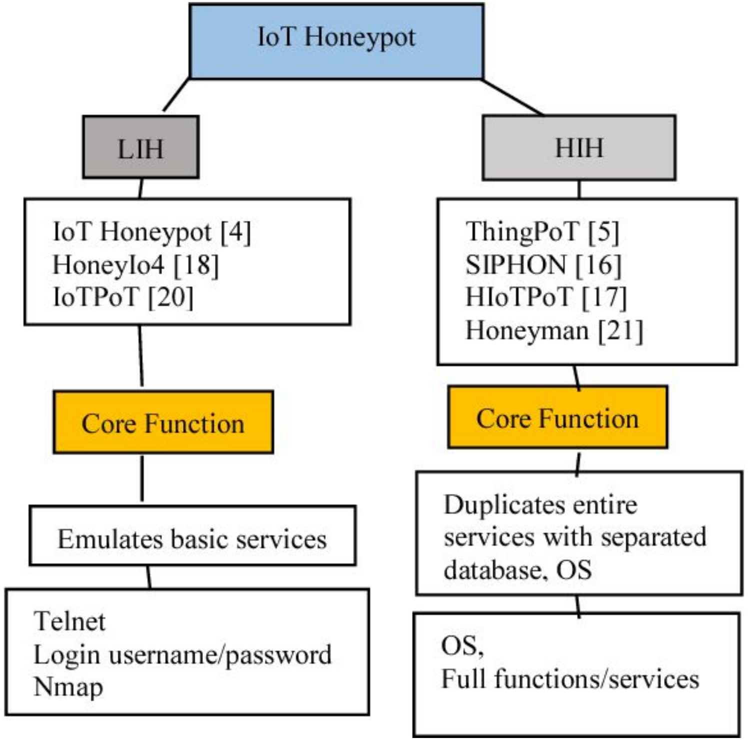
ThingPot

- ThingPot is a honeypot implementation of XMPP protocol
- 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
- REST is used to build a backend API. Given its popularity, XMPP has been selected as the IoT protocol for real-time communication. The frontend is realized through a simple HTTP web service



Wang et al., "ThingPot: an interactive Internet-of-Things honeypot", 2018

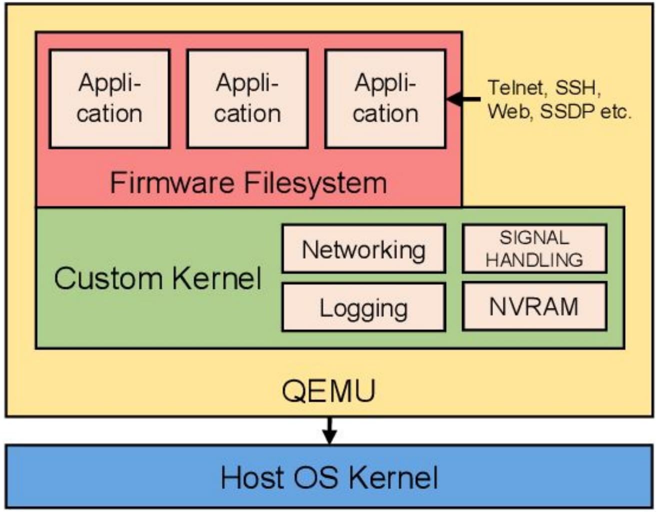
Honeypots taxonomy



Taxonomy related to the exposed honeypots

Honware

- A flexible and generic framework to efficiently and effectively deploy honeypots for networked devices on the Internet to log attacker traffic and their actions
- Honware utilises device firmware images (which are widely available for download) and a special prebuilt Linux kernel to emulate device behaviour within a virtual environment (so it doesn't utilise real devices)
- The performance of honware is comparable to real devices and is not susceptible to trivial fingerprinting based on timing attacks
- Using firmware images is cheaper and therefore more scalable



Vetterl and Clayton, "Honware: A Virtual Honeypot Framework for Capturing CPE and IoT Zero Days", 2019

Thank you for the attention :)

If you have any questions feel free to ask!