# Honeypot Strategies

## Report Deliverable, + repository

**Members:** Klivens Ziu
Antonin Alves Cardoso
Yohann Desravines

**16/05/2022**

# Quick description of our project

- Chosen a hybrid malware honeypot setup for our Scenario 4 (Hospital).

- Our setup imitates services and systems that would make our client seem like an easy target (more on that later), so that we can induce or provoke the most common type of attack against our client hospital, and that is ransomware, denying access to files that carry worth only to the client.

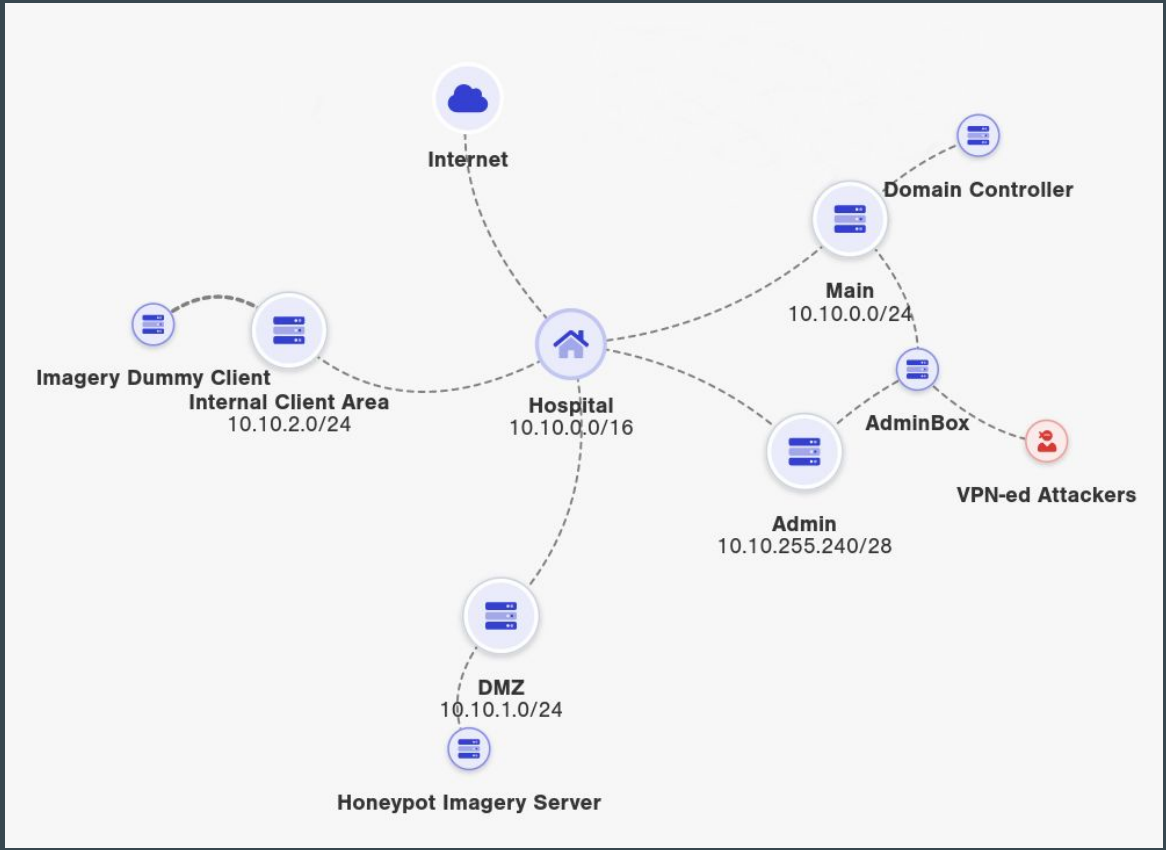
# Quick description of our project

- Created dummy server on honeypot VM and separate dummy clients, simulating imagery server/client HTTP or websocket connections, luring attacker to believe that the honeypot machine carries valuable data, therefore it's worth it to stage an attack.
-
- Honeypot is a 'weak-looking' VM while still being realistic, running an outdated OS which has severe public exploits (Ubuntu 16.04, kernel v4.4.x).
-
- This honeypot VM dummy server will sit inside a subnet DMZ, which only allows inbound connections and no outbounds to the real client network. Inside client network we have another subnet which contains dummy imagery clients, sending fake scanned images to the server into the DMZ. This client subnet will only allow outbound connections to the DMZ, and no incoming ones. Log software is installed into the server VM, reporting back any shell commands or files that attacker tries to alter, through any attack vector.
-
- This way, even if attacker puts effort into enumerating and analyzing our network stack, by default our internal genuine machines are unreachable by the attacker even after they gain a foothold on the honeypot (that's what we want.)

# Pros and cons of selected method

- Everything is based off VM setups, new instances can be spun up or down quickly, config files and different automated scripts make us write code once and forget.
-
- It would be considered a high-interaction honeypot even though the honeypot software being used has many low-interaction capabilities, like simulating genuine SSH & telnet. This carries the cost of running VMs, but gives flexibility by being layered. An attacker might think they fooled 1 layer, only to waste time on the next layer, Inception style. Even if the final ‘flag’ is achieved and ransomware is deployed, the server data is decoy and the VM can be cleaned and spun up without worrying of persistence.
-
- By having fragmenting subnets, the network stack has a genuine feel to it, while also being secure by default
-
- With this method, even if attacker puts effort into enumerating and analyzing our network stack, the internal genuine machines are unreachable by them even after they gain a foothold on the honeypot (that’s what we want.)
-
- No false-positives can occur since an attacker would actively move between layers and try to escalate.

# Infrastructure Recap



# Perceived initial attacks

Considering the low-interaction honeypot services running, one of our chosen existing software, Dionaea is running multiple simulated daemons including: (SERVICE / PORT) FTP/21, MongoDB/27017, MySQL/3306, UPNP/1900, DNS/53, HTTP/80/443.

Each entry is ready to analyze any attack thrown its way. Regarding the high-interaction attacks (meaning that the attacker takes a nice look at our running server and thinks its a valid target) that might occur, it might include:

- Public exploits related to outdated linux OS
- Compromised credentials leading to access through RDP or VNC or SSH, then exploiting all the way to a reverse shell being established, with admin rights.
- Misconfigured or weak-password account, exploited through the protocols mentioned above.
- Tricking a user to interact with a malicious attachment or executable, then progressing as described above.

# Why this methodology?

A layered approach seemed best, this can maximise usage of software like Dionaea, low-interaction services running along the logs functionality needed for the high-interaction element of tricking the attacker to... well, attack.

The software, Dionaea and Artillery can be tweaked with low effort into becoming anonymous (source code is openly available), by changing its folder routes and file namings (getting rid of keywords that will be used to fingerprint it), and by changing descriptions on its daemon services, such as the SSH one.

It offers protection or evasiveness against both lower and higher skilled attackers.

# Our Reasoning

All our choices were made keeping in mind the thread landscape for our Hospital client, considering that it will be targeted by a wide band of hackers, from script kiddie pasters to experienced global APTs.

Their common tactics, techniques, procedures and attack scenarios were identified and build into the honeypot architecture.

The simplicity of deploying snapshots of VMs, with automated scripts also gives a nice nod to the DevOps engineer in all of us.

# Demo Time

# Decoy server + client screenshots (high-interaction)

```
ubuntu@ip-10-10-0-71:~/honeypot_strategies/client$ nano .env
ubuntu@ip-10-10-0-71:~/honeypot_strategies/client$ npm run start

> client@1.0.0 start
> node app.js

Upload complete
```

Client on his own subnet, sends a random file every X seconds

```
ubuntu@ip-10-10-0-104:~/server$ ls -a
.  ..  .env  app.js  node_modules  package-lock.json  package.json  route.js
ubuntu@ip-10-10-0-104:~/server$ sudo npm run start

> server@1.0.0 start /home/ubuntu/server
> node app.js

Server active on port 8080
{
  fieldname: 'file',
  originalname: 'heart_scan_124.jpeg',
  encoding: '7bit',
  mimetype: 'image/jpeg',
  destination: 'uploads/',
  filename: '116cd8ffc31cfaf775d9909cb478b15d',
  path: 'uploads/116cd8ffc31cfaf775d9909cb478b15d',
  size: 133398
}
{
  fieldname: 'file',
  originalname: 'brain_scan_123.jpg',
  encoding: '7bit',
  mimetype: 'image/jpeg',
  destination: 'uploads/',
  filename: '92d47d5e74a2537f8a55156de2a411e9',
  path: 'uploads/92d47d5e74a2537f8a55156de2a411e9',
  size: 163297
}
```

Server inside DMZ subnet, receives the file and stores it inside “/upload”, usually inside “/var/www”

```
ubuntu@ip-10-10-0-104:~/server$ cd uploads/
ubuntu@ip-10-10-0-104:~/server/uploads$ ls -a
.  ..  116cd8ffc31cfaf775d9909cb478b15d  4e66485593bd30ab5789505e5f04809d  50d0a14b8a4e3ee3b0e089bd736c21ce  92d47d5e74a2537f8a55156de2a411e9
ubuntu@ip-10-10-0-104:~/server/uploads$
```

# Dionaea install and config

```
-- Installing: /opt/dionaea/var/lib/dionaea/fail2ban
-- Installing: /opt/dionaea/var/lib/dionaea/ftp/root
-- Installing: /opt/dionaea/var/lib/dionaea/http/root
-- Installing: /opt/dionaea/var/lib/dionaea/http/template
-- Installing: /opt/dionaea/var/lib/dionaea/http/template/example/form.html.j2
-- Installing: /opt/dionaea/var/lib/dionaea/http/template/nginx/autoindex.html.j2
-- Installing: /opt/dionaea/var/lib/dionaea/http/template/nginx/error.html.j2
-- Installing: /opt/dionaea/var/lib/dionaea/printer/root
-- Installing: /opt/dionaea/var/lib/dionaea/printer/root/0
-- Installing: /opt/dionaea/var/lib/dionaea/sip/
-- Installing: /opt/dionaea/var/lib/dionaea/sip/rtp
-- Installing: /opt/dionaea/var/lib/dionaea/tftp/root
-- Installing: /opt/dionaea/var/lib/dionaea/upnp/root
-- Installing: /opt/dionaea/lib/dionaea/curl.so
-- Installing: /opt/dionaea/lib/dionaea/emu.so
-- Installing: /opt/dionaea/lib/dionaea/nfq.so
-- Installing: /opt/dionaea/lib/dionaea/pcap.so
-- Installing: /opt/dionaea/bin/dionaea
-- Installing: /opt/dionaea/etc/dionaea/dionaea.cfg
-- Up-to-date: /opt/dionaea/var/lib/dionaea
-- Installing: /opt/dionaea/var/lib/dionaea/binaries
-- Installing: /opt/dionaea/var/lib/dionaea/bistreams
-- Installing: /opt/dionaea/var/log/dionaea
ubuntu@ip-10-10-0-104:~/dionaea/build$
```

```
ubuntu@ip-10-10-0-104:/opt/dionaea$ ls -a
.  ..  bin  etc  lib  share  var
ubuntu@ip-10-10-0-104:/opt/dionaea$ cd bin/
ubuntu@ip-10-10-0-104:/opt/dionaea/bin$ ls -a
.  ..  dionaea
ubuntu@ip-10-10-0-104:/opt/dionaea/bin$ cd ..
ubuntu@ip-10-10-0-104:/opt/dionaea$ cd var/
ubuntu@ip-10-10-0-104:/opt/dionaea/var$ ls -a
.  ..  lib  log
ubuntu@ip-10-10-0-104:/opt/dionaea/var$ cd log/
ubuntu@ip-10-10-0-104:/opt/dionaea/var/log$ ls -a
.  ..  dionaea
ubuntu@ip-10-10-0-104:/opt/dionaea/var/log$ cd dionaea/
ubuntu@ip-10-10-0-104:/opt/dionaea/var/log/dionaea$ ls -a
.  ..
ubuntu@ip-10-10-0-104:/opt/dionaea/var/log/dionaea$
```

Empty logs at startup, nothing has happened yet

```
ubuntu@ip-10-10-0-104:/opt/dionaea/bin$ sudo ./dionaea -D &
[1] 16192
ubuntu@ip-10-10-0-104:/opt/dionaea/bin$
Dionaea Version 0.11.0-7-g4e459f1
Compiled on Linux/x86_64 at May 16 2022 09:26:56 with gcc 7.5.0
Started on ip-10-10-0-104 running Linux/x86_64 release 5.4.0-1072-aws
```

Daemon instance running with -D flag

# Nmap screenshots

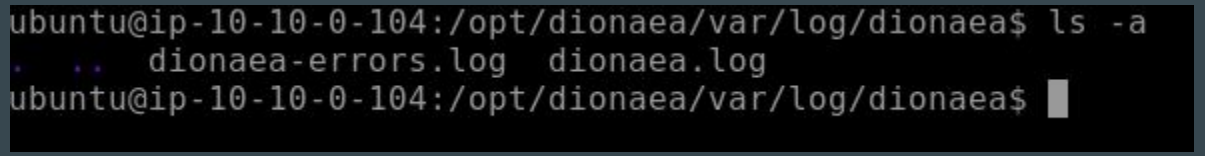
```
ubuntu@ip-10-10-0-71:~$ nmap 10.10.0.104
Starting Nmap 7.80 ( https://nmap.org ) at 2022-05-16 10:23 UTC
Nmap scan report for 10.10.0.104
Host is up (0.0078s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh

Nmap done: 1 IP address (1 host up) scanned in 0.12 seconds
```

Nmap scan of the server before activating Dionaea

```
Nmap scan report for 10.10.0.104
Host is up (0.00048s latency).
Not shown: 1981 closed ports
PORT     STATE         SERVICE      VERSION
21/tcp   open          ftp          Synology DiskStation NAS ftpd
22/tcp   open          ssh          OpenSSH 7.6p1 Ubuntu 4ubuntu0.6 (Ubuntu Linux; protocol 2.0)
23/tcp   open          telnet?
42/tcp   open          tcpwrapped
53/tcp   open          domain?
80/tcp   open          http         nginx
135/tcp  open          msrpc?
443/tcp  open          ssl/https?
445/tcp  open          microsoft-ds Dionaea honeypot smbd
1433/tcp open          ms-sql-s     Dionaea honeypot MS-SQL server
1723/tcp open          pptp          (Firmware: 1)
3306/tcp open          mysql        MySQL 5.7.16
5060/tcp open          sip?
9100/tcp open          jetdirect?
53/udp   open|filtered domain
68/udp   open|filtered dhcpc
69/udp   open|filtered tftp
123/udp  open|filtered ntp
1900/udp open|filtered upnp
MAC Address: 12:17:3A:FE:AA:ED (Unknown)
```

Nmap scan after activating Dionaea, left detectable on purpose, to show it is running. Removing this fingerprint is a very basic trivial task, the source code is open.

# Log file after scan and some requests




Dionaea log file sample after scan (low-interaction services), uploaded to github. (very verbose and detailed, on purpose)

# Artillery install and setup

```
ubuntu@ip-10-10-0-104:~/artillery$ sudo ./setup.py

Welcome to the Artillery installer. Artillery is a honeypot, file monitoring, and overall security tool used to protect your nix systems.

Written by: Dave Kennedy (ReL1K)

Do you want to install Artillery and have it automatically run when you restart [y/n]: y
[*] Beginning installation. This should only take a moment.
[*] Adding artillery into startup through init scripts..
[*] Triggering update-rc.d on artillery to automatic start...
[*] Do you want to keep Artillery updated? (requires internet) [y/n]: n
[*] Copying setup files over...
2022-05-16 11:10:05: A brand new config file '/var/artillery/config' was created. Please review the file, change as needed, and launch artillery (again).
[*] Would you like to start Artillery now? [y/n]: y
Starting Artillery...                              Ok
[*] Installation complete. Edit /var/artillery/config in order to config artillery to your liking
```

```
ubuntu@ip-10-10-0-104:/var/artillery$ ls -a
.  ..  .git  .gitattributes  .gitignore  README.md  artillery.py  artillery_start.bat  banlist.txt  config  database  readme  remove_ban.py  restart_server.py  setup.py  src
ubuntu@ip-10-10-0-104:/var/artillery$ sudo python3 restart_server.py
a bytes-like object is required, not 'str'
[*] 2022-05-16 11:12:30: Restarting Artillery Server...
ubuntu@ip-10-10-0-104:/var/artillery$ sudo python3 artillery.py &
[1] 16567
ubuntu@ip-10-10-0-104:/var/artillery$ 2022-05-16 11:12:48: Checking existing config file '/var/artillery/config'
2022-05-16 11:12:48: Artillery has started
2022-05-16 11:12:48: If on Windows Ctrl+C to exit.
2022-05-16 11:12:48: Console logging enabled.
2022-05-16 11:12:48:
2022-05-16 11:12:48: Artillery is running from '/var/artillery'
2022-05-16 11:12:48: Running auto update (git pull)
2022-05-16 11:12:50: Creating iptables entries, hold on.
2022-05-16 11:12:50: iptables entries created.
2022-05-16 11:12:50: Launching honeypot.
2022-05-16 11:12:50: Launching SSH Bruteforce monitor.
2022-05-16 11:12:50: Launching monitor engines.
2022-05-16 11:12:50: Check system hardening.
2022-05-16 11:12:50: [!] Insecure configuration detected on filesystem: [!] Issue identified: /etc/ssh/sshd_config. SSH is running on the default port 22. An attacker commonly scans for thes
e type of ports. Recommendation: Change the port to something high that doesn't get picked up by typical port scanners.
2022-05-16 11:12:50:
2022-05-16 11:12:50:
2022-05-16 11:12:50: Launching thread to get source feeds, if needed.
2022-05-16 11:12:50: All set.
```

# Artillery Log file

```
ubuntu@ip-10-10-0-104:/var/artillery$ ls -a
.  ..  .git  .gitattributes  .gitignore  README.md  artillery.py  artillery_start.bat  banlist.txt  config  database  readme  remove_ban.py  restart_server.py  setup.py  src
ubuntu@ip-10-10-0-104:/var/artillery$ cd database/
ubuntu@ip-10-10-0-104:/var/artillery/database$ ls -a
.  ..  integrity.database
ubuntu@ip-10-10-0-104:/var/artillery/database$
```

Artillery database, scanning for changes to protected folders like “/etc” and “/var/www”, uploaded to github.

# The End

Thank you for your attention.