

Forensic Investigations (FORIN)
REPORT

-

Content

Introduction	3
Literature review\design	4
Design elements	4
Implementation	6
Results/analyse	9
Json	9
IP address'	9
Attack methods	11
Types of attacks	12
Time stamps	12
Cowrie log	13
Conclusion	16
References	17

Introduction

I'm performing this study on behalf of a developing cloud provider to show how using a honeypot to safeguard your services and machines from attackers may be very helpful. Since they can monitor the attacks, controlling how to stop upcoming attacks is possible. This report's primary objectives are:

1. Describing the honeypot and the protection it provides for the system and how I made my own to fit this
2. Show the business how to integrate Cowrie into their systems.
3. To comprehend what the log files indicate and how we might utilise them to anticipate future attacks by figuring out what individuals would try.
4. Advise on what the business should do to stop upcoming attacks.

Honeypots are system accounts that serve as decoys for outside attackers. Honeypots like Cowrie, Dockpot, HonSSH, and SSH are types of machines that can let a user who is hosting one gain information on what attackers could possibly do on their system. This can allow the company to patch these issues out to make their system really secure.

Literature review\design

Honeypots are a “fake” system used mainly to trick attackers, allowing security professionals to study their tactics, techniques, and procedures (TTPs) and gain insights into their methods. A honeypot in cybersecurity is a device intentionally set up to lure and detect attack activity. Typically, these systems are exposed to the internet and mimic or include existing systems that attackers can target (Tabari, Ou, & Singhal. (2021)). Honeypots have two main vital elements when it comes to design for the security of the system it is trying to protect. (Fan, Du, Smith-Creasey & Fernandez (2019)), Suggest It is essential to employ all-encompassing honeypots that enhance sensitivity, countermeasures, and stealth to address the problem effectively.

Design elements

For the development of my honeypot, I will be using Cowrie. The three critical factors for making a honeypot the best it can be is sensitivity, countermeasure and stealth. These key factors should help benefit your system by allowing it a false sense of access to any attackers. This review will discuss how these are important to a honeypot design and how they keep an overall system safe.

The honeypot system is skilled at spotting different types of attacks and can classify and analyse attack data precisely. Instead of only warning about dangerous activities, it represents an advanced detecting capacity (Fan, Du, Smith-Creasey & Fernandez (2019)). All three of these systems link together. They are all based on the concept of allowing analysis of the system using them. The sensitivity gives up a precise way to analyse the data by monitoring the attacking methods the attackers are trying.

Instead of becoming a victim of the assault, the honeypot system might respond by gathering high-quality attack data (Fan, Du, Smith-Creasey & Fernandez (2019)). This is the concept of countermeasure; this allows the potential thought that the attacker is in the actual machine since they can run code and try to execute, but they will do nothing since the account is borderline nothing. This allows the idea that the device is “real” but is just a way for the company to analyse data imputed from attackers to let them improve their systems.

In my honeypot, this is present since the ability to log in to the actual user account of “cowrie” is available, and they can run commands in the system to try and do whatever they want.

Stealth is the idea of making it so much like a system that it is unrecognisable to an attacker. This is very important to the concept of a honeypot. (Fan, Du, Smith-Creasey & Fernandez (2019)), In honeypot systems, stealth keeps the machine hidden from attackers

while monitoring attack behaviours. The importance of this is that the company is running this honeypot for its own protection since it will be a fake system that they can analyse.

As demonstrated in my honeypot, I have set the SSH port for the honeypot from 2222 to 22 to create a dummy that the attackers will think is the main machine. Also, I changed the hostname so attackers will see the "fileserv" and assume this is a company-hosted machine.

By setting up a honeypot system that runs actively through a port on a server or system, companies can effectively create a "fake" environment. This primarily aims to entice attackers into attempting to steal information from the system, thereby alerting the operators. This approach allows the operators to identify the specific areas that attackers are targeting and improve security measures in preparation for a potential attack on their system.

However, with the implication of a honeypot, there comes a way to compromise a honeypot where an attacker could know it is one. Fingerprinting is the collective information that attackers use to check for software, network protocols, operating systems and devices on a network. Some people (Naik, N., & Jenkins, P. (2018)) argue that while fingerprinting poses minor damage to many systems, it poses a severe risk to the honeypot system since keeping its anonymity is crucial to catching an attacker. This is why we need to ensure the honeypot is secure and misleading enough to ensure people get confused and do not recognise that it is a honeypot.

Implementation

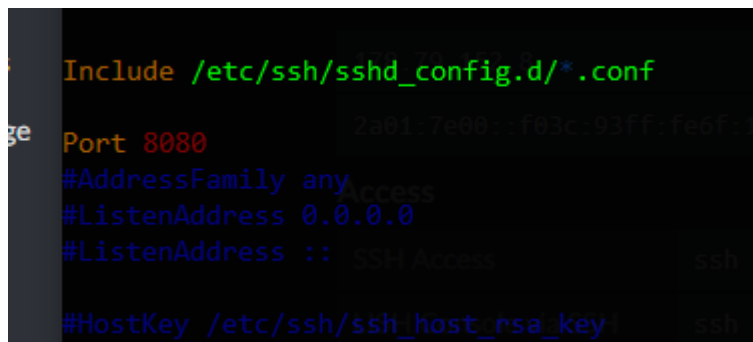
To create my honeypot, I used the cloud service Linode. Linode allows me to create a “virtual machine/web server.” Firstly, I ran the command to update and upgrade the system (Figure 1); this lets me ensure everything is up-to-date and has all the necessary versions to ensure everything works. We must also ensure we install all developer programs, such as python3-virtualenv and lib dev.



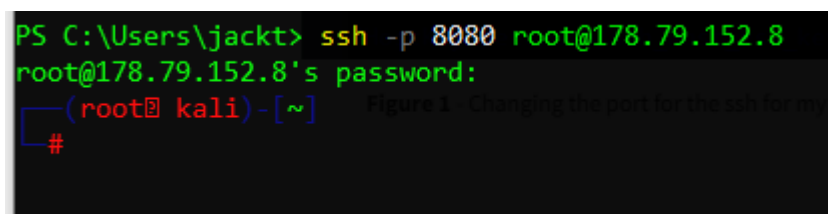
```
(root@kali)-[~]
# apt-get update && apt-get upgrade
```

Figure 1 - updating the machine and applications

With this machine, I will go to “etc/ssh/sshd_config.d” and change the port for SSH from 21 to 8080 (figure 1.2); this allows me to allocate the Cowrie account/”machine” to 22 (figure 1.10). After restarting SSH, I can access my virtual machine through that port (figure 1.3).



```
Include /etc/ssh/sshd_config.d/*.conf
Port 8080
#AddressFamily any
#ListenAddress 0.0.0.0
#ListenAddress ::
#HostKey /etc/ssh/ssh_host_rsa_key
```



```
PS C:\Users\jackt> ssh -p 8080 root@178.79.152.8
root@178.79.152.8's password:
(root@kali)-[~]
#
```

Figure 1.3 - Ssh new port

For the next step, I need to create a new user for the Cowrie honeypot (figure 1.4); this is so we can have a “fake” user account and download the needed applications for it. We first switched to the new cowrie account (figure 1.5). Then I went and get a downloadable directory through “git cloned <https://github.com/micheloosterhof/cowrie>”; this gave me all the necessary files for the Cowrie system (figure 1.6). I then ran a virtual environment, which allowed me to interact with the computing environment and other user accounts, and I then activated the cowrie (figure 1.7).

```

root@178.79.152.8's password:
(root@kali)-[~]
# adduser --disabled-password cowrie

```

Figure 1.4 - The creation of a new user account for Cowrie

```

(root@kali)-[~]
# su - cowrie
(cowrie@kali)-[~]
$

```

Figure 1.5 - Switch to cowrie users

```

(cowrie@kali)-[~/cowrie]
$ ls
bin          docs          Makefile      requirements-output.txt  src
CHANGELOG.rst  etc          MANIFEST.in   requirements.txt         tox.ini
CONTRIBUTING.rst  honeyfs      pyproject.toml  setup.cfg               var
cowrie-env     INSTALL.rst  README.rst     setup.py
docker         LICENSE.rst  requirements-dev.txt  share

```

Figure 1.6 - the results of me getting the files needed for the machine

```

(cowrie@kali)-[~/cowrie]
$ source cowrie-env/bin/activate

```

Figure 1.7 - The activation of the cowrie virtual environment

By default, after cloning the cowrie file to the machine, there are two files for configurations, but we need one that is not installed by default so that we will be going and copying the old “cowrie.cfg.dist” and naming it “cowrie.cfg” (figure 1.8). In the new configuration file, we need to go and change two things, the “hostname” and “listening_endpoint” I made the hostname “fileservr” (figure 1.9); this provides a false sense of assertion when getting access to the machine, also, went and changed the listening port to 22 this is so cowrie is listening on 22 and not 2222 to allow for the ssh(figure 1.10).

```

(cowrie@kali)-[~/cowrie]
$ ls etc
cowrie.cfg  cowrie.cfg.dist  userdb.example

```

figure 1.8 - Copying of the old file and making a new one

```
# (default: svr04)
hostname = fileserver
```

Figure 1.9 - changing the hostname of cowrie

```
# change interface below to allow connections from
listen_endpoints = tcp:22:interface=127.0.0.1
```

Figure 1.10 - changing the ssh tcp

Before finalising the Cowrie pot, we must run the command that adds rules to the NAT table. The command we will be running will affect the prerouting chain, redirecting traffic coming from TCP on port 22 to port 2222 (figure 1.11). This allows the ssh on port 22 to go to the worry honeypot

```
(root@kali)-[~]
# iptables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT --to-port 2222
```

Figure 1.11 - The iptables command

Finally, to activate the honeypot, we needed to run one final command that activates the cowrie pot (figure 1.12). This command activates the system, and we will be waiting a few days to see if any logs come through; if they do, we can see that the system is fully working, and we can start to analyse what happens.

```
bin          docker    INSTALL.rst
CHANGELOG.rst docs      LICENSE.rst
CONTRIBUTING.rst etc       Makefile
cowrie-env   honeyfs  MANIFEST.in

(cowrie@kali)-[~/cowrie]
$ source cowrie-env/bin/activate
```

Figure 1.12 - Activating cowrie

Results/analyse

At this point in the report, I have left my machine running to collect data on potential outside sources trying to attack my machine; the honeypot was running for about two weeks. We will be checking the log files in the “var/log/cowrie” This will allow us to check the JSON and log files. Predominantly, in the JSON file, we will see that there is mainly the way the attackers are trying to gain access to the system; we could see some form of dictionary tracks with brute force elements. The log file showed us when they accessed the account and what they tried to look for on the accounts.

Json

The “cowrie.json” file will allow us to discover information about the potential things attackers are trying to do to gain access to our systems. Examples of the data we could discover are

IP address: the location of where people are attacking from

Attack methods: ssh, how many total attacks came through, with what succeeded and failed

Types of attacks: dictionary brute force attacks, what credentials they are using in the attacks

Time stamps: allowing us to understand when people will attack

IP address'

With the information, I will be going and analysing the JSON file to see potential attacks; I will be using commands to help us examine the file.

for the beginning of this analysis, I will see where the IPs are coming from, and I will be using the command (figure 2.1); this command will go and extract all the IPs from the cowrie.json file that have tried to attack my honeypot, and all the unique ones so they will not duplicate (Figure 2.2).



```
(root@kali)-[~]  
# jq -r '.src_ip' cowrie.json | sort -u > final.txt
```

Figure 2.1 - Command used

```
(cowrie@kali)-[~/cowrie/var/log/cowrie]
$ cat final.txt
103.138.71.242
103.86.49.28
103.96.151.129
107.170.208.6
113.88.209.126
114.35.42.193
114.44.144.120
116.30.223.29
121.146.142.226
122.255.40.202
```

Figure 2.2 - the result of the ips collected

To understand the results fully, I will use a Linux application called "geoipllookup" This allows me to run this command (Figure 2.3), this command will echo the ip address and its location separated in the text file (figure 2.4). I used one more command to analyse these IPs to check which area is most frequent when trying to attack my machine (figure 2.5), this command goes and grabs the specific countries that match by text, In this command using its countries anagram example CN = China. Then it counts each specific one while sorting them into their particular groups, letting me see the overall count after completion (figure 2.6).

This information gives us an idea of potential ways to block specific attacks by knowing where they are coming from and researching possible block methods for particular areas.

```
(root@kali)-[~]
# while read -r ip; do echo "$ip: $(geoipllookup "$ip")"; done < ips.txt > locations.txt
```

Figure 2.3 - Command used

```
(cowrie@kali)-[~/cowrie/var/log/cowrie]
$ cat general_locations.txt
103.138.71.242: GeoIP Country Edition: ID, Indonesia
103.86.49.28: GeoIP Country Edition: TH, Thailand
103.96.151.129: GeoIP Country Edition: HK, Hong Kong
107.170.208.6: GeoIP Country Edition: US, United States
113.88.209.126: GeoIP Country Edition: CN, China
114.35.42.193: GeoIP Country Edition: TW, Taiwan
114.44.144.120: GeoIP Country Edition: TW, Taiwan
116.30.223.29: GeoIP Country Edition: CN, China
121.146.142.226: GeoIP Country Edition: KR, Korea, Republic of
122.255.40.202: GeoIP Country Edition: LK, Sri Lanka
125.140.246.14: GeoIP Country Edition: KR, Korea, Republic of
128.199.171.119: GeoIP Country Edition: SG, Singapore
129.150.51.244: GeoIP Country Edition: SG, Singapore
```

Figure 2.4 - countries that attacked unsorted

```
(root@kali)~# grep -oP '(?<=GeoIP Country Edition: )\w+' general_locations.txt | awk '{count[$0]++} END {for (i in count) print count[i], i}' | sort -rn > final_countries.txt
```

Figure 2.5 - Command used

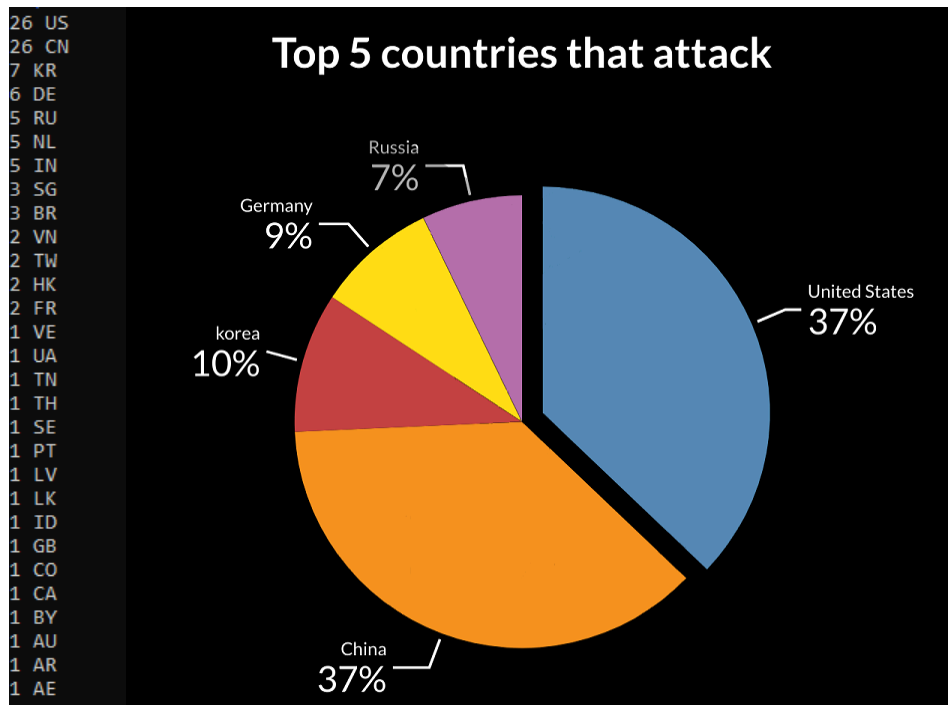


Figure 2.6 - the result of the sorting

Attack methods

Since creating this honeypot, we know that attackers can only attack through ssh. Allowing me to majority-check the JSON file to see how many successful attacks and failures. Let me analyse the successful attempts and see what we need to improve to make sure that there are no successful attacks (figure 2.7). The command will go into the cowrie.json file and only give us the attacks that succeeded by removing the failed attempts (figure 2.8).

```
(root@kali)~# grep "login attempt"cowrie.json | grep -v "failed" > Successful.txt
```

Figure 2.7 - Command used

```

on": "eb413113c117"}
: "login attempt [root/3245gs5662d34] succeeded"
ion": "ad0d5a6a8c86"}
: "login attempt [root/3245gs5662d34] succeeded"
on": "48fe1e903aa2"}
: "login attempt [root/3245gs5662d34] succeeded"
on": "1d58cbded583"}
: "login attempt [root/3245gs5662d34] succeeded"
sion": "4acf074285fe"}
: "login attempt [root/3245gs5662d34] succeeded"
sion": "edcdf8edc03a"}
: "login attempt [root/3245gs5662d34] succeeded"
sion": "e123906631d9"}
: "login attempt [root/3245gs5662d34] succeeded"
on": "969297cc99a4"}
: "login attempt [root/3245gs5662d34] succeeded"
ion": "4ef6cd469537"}

```

Figure 2.8 - The successful attempts of the attacks

This shows me that this type of password is commonly found online in a word list. I need to make sure that the password is a lot more secure, involving more characters, symbols, numbers, and anything that can make it secure enough that a word list would not have any ability to crack it.

Types of attacks

After looking through the file, we can denounce the types of attacks attempted on the system. Looking through the JSON file, we can see many duplicate ip addresses and many attempts to log in through ssh. This tells me that the system's most common attack so far is a brute force attack through ssh using wordlists. Having this idea let me know that we need to make sure we prevent the possibility of the password from being cracked

Time stamps

The Cowrie JSON file provides every single event ID with a timestamp that falls under the format "year-month-date:hour:minute: second" (figure 2.9); for the concept of protection of our systems this is very useful. It can allow us to analyse the file, it can identify attackers patterns and behaviours lettings take that knowledge and be able to patch out those attacks they perform. These can also link with the log file since it will provide an exit time so we can see when they entered and where they left.

```

, "timestamp": "2023-04-15T09:29:57.697776Z", "src_ip": "51.210.254.
, "timestamp": "2023-04-15T09:29:57.698602Z", "src_ip": "51.210.254.
": "7f01121d716c", "protocol": "ssh", "message": "New connection: 49.
nsor": "kali", "timestamp": "2023-04-15T09:30:21.158461Z", "src_ip":
a256@libssh.org, ecdh-sha2-nistp256, ecdh-sha2-nistp384, ecdh-sha2-
llman_group1, sha1_ext_info: c:a05256_gcm@openssh.com a05128_gcm@

```

Figure 2.9 - timestamps present in the JSON file

Cowrie logs

The second file we will be viewing is the “cowrie.log” file. The file will give me more detail on what the attackers actually did on the user account after gaining access. The file will show us:

Commands: if they run any commands, we can see what they tried doing

SSH: transporting and debugging information

Time stamp: Can see when they attacked and when they stopped, plus the date

I will be looking through the cowrie.log file and analysing anything I find that is interesting.

```
3.160.187.222] Command found: cd ~
3.160.187.222] Command found: rm -rf .ssh
3.160.187.222] Command found: mkdir .ssh
3.160.187.222] Command found: echo ssh-rsa AAAAB3NzaC1yc2EAAAABJQAAAQEArDp4cun2
g9eLBHpgLMuakb5+BgTFB+rKJAw9u9FSTDengvS8hX1kNFS4Mjux0hJOK8rvcEmPecjdySYMb66nylA
+i6rBLAsPKgAySVKPRK+oRw== mdrfckr >> .ssh/authorized_keys
3.160.187.222] Command found: chmod -R go= ~/.ssh
3.160.187.222] Command found: cd ~
```

Figure 3.1 - First set of commands found in the log file

I will deconstruct what the attacker is trying to do here. Firstly I can see the attacker tries to “cd ~” change the directory to the home folder; after that, they try running the command “rm -rf .ssh!”. They are trying to remove a folder called “.ssh”; they do this with the “-rf” flag for rm since they are trying to remove a root-owned folder.

After removing the “.ssh” folder, they go and remake it. With this “echo ssh-rsa” plus a long “key”, the beginning of this command generates a new key. After doing some research, the long “AAAAB3+” has been complained about online to be related to crypto mining, but some people do not know 100%, but from what I assume, they are using it to generate a key, combined with this, they used the command “>> .ssh/authorized_keys”, this writes everything they did to a file called “authorized_keys” this authenticates ssh connections used.

Finally, they do “chmod -R go= ~/.ssh” in this command set. This command changes the permission set using “chmod -R go=” This part removes all of the read, write and execute permissions, leaving only the permissions for the file owner.

For this attack, the overall idea is that the user could have been doing this to modify the ssh configuration of a user’s account; this could lead to unauthorised access to the system, which will pique interest to a host running this system to make sure they can add methods to try to mitigate it: limiting user privileges, monitoring the logs as we did here and keep an eye out for attempts, However, this can be used for a legitimate reason such as a user on the system creating new keys for themselves, but should still monitor the activity.

```
1.193] CMD: uname -s -v -n -r -m
1.193] Command found: uname -s -v -n -r -m
```

Figure 3.2 - uname commands and flags

The uname command has a bunch of flags that can be used with the command to give information about the system; examples:

- s - The operating systems name
- v - The operating system version
- n - The hostname of the system
- r - The kernel release number
- m - The hardware name

For the attack they may run this command so they can see the specifics of the system, this can allow them to find a bunch of potential vulnerabilities and known exploits on that version. If there are any that they try to use on this system, I can see them and go and patch them out if they are possible on my main system.

```
Setting shell
.230] CMD: ls
.230] Command found: ls
.230] CMD: ls
.230] Command found: ls
.230] CMD: su root
.230] Command found: su root
.230] CMD: ls
.230] Command found: ls
.230] CMD: ls
.230] Command found: ls
.230] CMD: whoami
.230] Command found: whoami
.230] CMD: exit
.230] Command found: exit
itCode: 0
nection#debug] sending request b'ex
.230] Closing TTY Log: var/lib/cowrie
```

Figure 3.3 - random commands ran

(Figure 3.3) demonstrates how an attacker can just do some simple things to determine if the machine was attackable, they got access, ran some simple commands to see what was present and tried switching the user, since they do not really display anything and the user was already “root” for the cowrie machine.

```
59] Saved redir contents with SHA-256 a8460f446be540410004b1a8db4083773fa46f7fe76fa842
a8db4083773fa46f7fe76fa84219c93daa1669f8f2
59] Closing TTY Log: var/lib/cowrie/tty/cc1eb03e9b5926d8076e25826664a04400de854bf5cc66
```

Figure 3.4 - File transfer

```
49:25.485666Z [HoneyPotSSHTranspo
var/lib/cowrie/downloads/a8460f44
49:25.486200Z [HoneyPotSSHTranspo
fter 0.5 seconds
```

Figure 3.5 - saved location

(figure 3.4) shows the ability of a file transfer that has taken place and has been saved to the downloads folder of the system (figure 3.5). Now the potential damage of this could be drastic depending on the type of file they actually are.

```
01ba4719c80b6fe911b091a7c05124b64eece964e09c058ef8f9805daca546b  abbc58371b6f5262ebe669c4bc33e7e2b95f5b7f9233e7bffd64205fcff2c3d  tmp1txh35po  tmpk1d1c_gc
08f0712639365a9a358b17aa4cf05522ab0fd4e5ddea99daa11a0ab9de375114  c689354ea82ea3c621f05b7ae0aac44adb68da43e94311e99d31e4d4e0dc7efe  tmp2fq1lj5s  tmp1fyzt468
29574dc19a017adc4a026deb6d9a90708110eafe9a6acdc6496317382f9a4dc7  ca281ea977de0ce219913fc8e018c3718320e1c2aef6b399736a0e80cfacff34  tmp4pb64dqs  tmpmhqcl0k7
6288c7119ffc9b2628dd82f0386262ffc066ac0c76e51c7e39f746466695e1e  dcd69fb81b3a487cdfb20b8e5549fdd8421696dc9b89e4e2c9640ee868e5dcb8  tmp7od7xj08  tmppuc6f_pd
7aae334219ed0d7af2eaff729050ae2fc1bdf2c286fdc8a00be39c8f4907ff19  e3b0c44298fc1c149afb4fc8996fb92427ae41e4649b93ca495991b7852b855  tmpcdtpvgxt  tmppsflv02hs
80ad3c4b6686c680178b1f2cf87321c7a8cd61d28d16d88b7baf1be505b57f7a  ea40eccec0b30982fbb1662e67f97f0e9d6f43d2d587f2f588525fae683abea73  tmpd9q_k9hh  tmprsgz86rc5
94f2e4d8d4436874785cd14e6e6d403507b78750852f7f2040352069a75da4c00  eb045debbdbdbb3cf5157445d2314e0207edaf5b5cf02eb611afaa167a54120e  tmppegqtwy7x  tmpx8ksq5vw
968854e7bc88c586dfd98a5369ac3e4658262ae8039b5659789eb2b755dd30ea  f6beba5a89985bbd86ceadac5d72f9f660be979c254c189a7301ec7dd7769d3c  tmpfvcp3995  tmpyx50ltba
a8460f446be540410004b1a8db4083773fa46f7fe76fa84219c93daa1669f8f2  ff6f81930943c96a37d7741cd547ad90295a9bd63b6194b2a834a1d32bc8f85d  tmpgm90_nsw  tmpzaf6adr9
```

Figure 3.6 - The files and there hashes

I went to the file location where these fields were saved (figure 3.6), I checked out some of these hashes by researching if other people have seen these before or if I could potentially crack them. I searched some of the hashes, and they provided some results. The links are provided below. From what I researched, I found that these links are some form of malware installed on the system. This tells me that the attackers are trying to potentially use these viruses to perform some form of attackers or even just abuse my system

Links:

AlienVault - Open Threat Exchange. (n.d.). AlienVault Open Threat Exchange.
Retrieved April 20, 2023, from

<https://otx.alienvault.com/indicator/file/01ba4719c80b6fe911b091a7c05124b64eece964e09c058ef8f9805daca546b>

Gnostis. (2023, April 17). [GS-224] Mirai Botnet IOCs. SEC-1275-1.

<https://1275.ru/ioc/1775/gs-224-mirai-botnet-iocs/>

Conclusion

As predicted we stated that the use of a honeypot will aid in the ability to help mitigate potential security risks that can come when having a machine on an open IP. From the

results, it can potentially help a company maintain a more secure system. On that basis, I have suggested some recommendations to help with the data that we gain from a week's run:

- Ip address: Block specific ip addresses that attack more than once. Use a blacklist that keeps information on all “known” malicious IPs and blocks them. One main option is to just monitor your logs for malicious activity.
- Brute force password access: Make sure to use strong, unique passwords. With a mix of characters like “£\$%!^&*)” plus capital letters and numbers. Also, enforce a firm password policy for all users using the machine
- Commands: Implement a stricter privilege policy for all system accounts to ensure they can do what they need to but not run unneeded commands. Monitor the users' logs.

Overall, this shows how even the littlest of time I could run the honeypot demonstrates how much it can drastically affect a company.

References

Fan, W., Du, Z., Smith-Creasey, M., & Fernandez, D. (2019). Honeydoc: an efficient honeypot architecture enabling all-round design.

IEEE Journal on Selected Areas in Communications, 37(3), 683-697

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8545323>

Mokube, I., & Adams, M. (2007, March). Honeypots: concepts, approaches, and challenges.

In Proceedings of the 45th annual southeast regional conference (pp. 321-326)

<https://dl.acm.org/doi/pdf/10.1145/1233341.1233399>

Naik, N., & Jenkins, P. (2018, October). Discovering hackers by stealth: Predicting fingerprinting attacks on honeypot systems.

In 2018 IEEE International Systems Engineering Symposium (ISSE) (pp. 1-8). IEEE.

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8544408>

Tabari, A. Z., Ou, X., & Singhal, A. (2021). What are Attackers after on IoT Devices?

An approach based on a multi-phased multi-faceted IoT honeypot ecosystem and data clustering. arXiv preprint arXiv:2112.10974.

<https://arxiv.org/pdf/2112.10974.pdf>