The Honeypot

Mauricio Aquino & Nathan Sanders
University of Colorado Colorado Springs

Throughout this report we will discuss the design of the project, tools/techniques we studied, implementation, testing, lessons learnt, and future work. A honeypot is a very unique and easy way to gather valuable information about attackers and methods of intrusion without allowing vulnerabilities in your system.

1. Introduction

"The art of war teaches us to rely not on the likelihood of the enemy's not coming, but on our own readiness to receive him; not on the chance of his not attacking, but rather on the fact that we have made our position unassailable" (M. Valentine). This quote speaks to this project as well as the entire cyber security field. Attacks can come from anywhere at any time and it is our job to be ready for whatever is sent our way. For this project, we wanted to test the boundaries of the idea that you can't stop a determined attacker. While you can prepare and develop reaction plans, a determined hacker will always find a way.

Similarly, Chris Sanders writes in his book *Intrusion detection honeypots: detection through deception*, "No matter how deliberate your efforts, an attacker who is relentlessly motivated or resourced will eventually gain access to a device on your network. Therefore, you should deploy detection mechanisms to aid in investigating and responding to attacks as quickly as possible" (Sanders, 4). We can't create a perfect system, we can get close, but we will almost always leave something vulnerable. For this project, we wanted to create something that allows us to implement deception and get the upper hand on the attacker.

As Jack Dempsey famously said, "The best defense is a good offense." (W.Safire) Similarly, the best way to learn about the attacker is by learning their methods. To test this idea of offensive security we created a honeypot. Honeypots are a computer security resource that provide system administrators the ability to detect network intrusions and learn about the tactics bad cyber actors use to exploit vulnerabilities. Sanders defines Honeypots as "a security resource whose value lies in being probed, attacked, or compromised" (Sanders, 21). Our goal was to create an intentionally vulnerable system. Attackers gain access to the system and while they think they have successfully bypassed our security measures, they actually fell into our honeypot.

2. Challenges

The two main challenges with honeypot technology are the overhead and the risks that they present for system administrators. Fortunately, these two factors are able to be mitigated and controlled through honeypot interactivity. Honeypot interactivity can be defined as how much the system allows the user to interact with the system through the use of commands, services, etc. Currently, there are a variety of honeypot technologies that offer a wide range of interactivity, but with that comes increased levels of risk and overhead. The upside to the vast variation of options available in this field is that there is a honeypot solution for virtually any business or organization.

3. Related work and their shortcomings

Currently as it stands, honeypots are a growing and innovative technology. This means that presently there aren't many other technologies that exist that are similar to honeypots. However, as previously mentioned there are a wide variety of types of honeypots and available software. There are three main categories of honeypots that all honeypots fall into: systems, services, and tokens. (Sanders, 22) Honey systems mirror operating systems, honey services mirror a service, and honey tokens mirror some type of data or document. Presently there are a wide variety of free and open source honeypot softwares available to use on the internet. "As of this

writing, I count over 80 different free honeypot tools focused across a broad spectrum of systems and services." (Sanders, 20) Some popular examples are Cowrie, Conpot, Dionaea, RDPy, Open Canary, Glastopf, and HoneyPress. (Sanders, 20)

It is our role as future system administrators to defend networks. Knowing how to detect network intrusions and learn about common cyber attacks are important lessons to learn for our career. Honeypots are a new solution to the old problem of cyber security. With time honeypots will gain much more attention for the innovative and unique design. From our experience as computer security majors, we have learned the value of protecting data and staying ahead of our attackers. Through coursework, real world examples, and designing, implementing, and testing our own systems, we have learned and acquired new skills for how to best protect systems and keep information safe.

5. Design

The first element that we will guide you through for our project is the high level design considerations we made before starting our project. To test the honeypot and truly understand what value it can add to a security system, we created our own honeypot and experimented with its capabilities.

We implemented a ssh honeypot using the software called Cowrie. Cowrie allows individuals to connect and interact with a fake computer system through the use of the ssh protocol. The advantage here is that you are able to detect intrusions and get information about the attacker through their interactions with the honeypot. We chose this honeypot software because of our familiarity with the ssh protocol and because it gave this project the right amount of interactivity and complexity we were looking for. For our honeypot, we decided to emulate an Ubuntu 18 server system as it is familiar to our team and simple to use. The system we placed our Honeypot on is a Raspberry PI 3+ due to its versatility and its affordability.

There were multiple design considerations that we wanted to make sure would be fulfilled during the planning of this project. The first is that we wanted to make the system secure so that there would be little risk for an attacker to turn the honeypot against us. The second design consideration we made is we wanted the honeypot to appear believable so that we do not scare off attackers and can get as much information from them as possible. While the honeypot is indeed a completely fake system, it mirrors a real system in appearance and functionality to convince the attackers that they have successfully penetrated a real system.

6. Tools/techniques studied

For this project, we studied the see, think, do methodology covered in Chris Sander's book. He mentions you want the attacker to see systems, services or data that are actually honeypots, you want them to think honeypots are valuable, and you want the attacker to do something that causes an interaction with the honeypot. (Sander, 42) We also studied different ways to make ssh connections secure in order to maintain a secure system. Our goal was to be able to remotely connect to the system without adding any unnecessary risk and without leaving the system vulnerable. The rest of our preparation revolved around finding ways to make the system appear real and be interactive to the attacker.

7. Implementation

The first step towards implementation was installing virtual machine software onto a windows machine. This was a key component to our project because of our goal to make the system appear real. The best way to make our honeypot seem real was by emulating a real system. The reason we chose to use VMs to do this is because it would give us the ability to create a system quickly from the ground up without needing additional hardware outside of the honeypot system. The VM allows us to control information as well as create a more safe testing environment than our personal computers. We also used multipass and Hyper-v on linux and windows respectively to create Ubuntu 18 server systems which we could emulate.

The second step for implementation was installing linux on a clean system for our honeypot to run on. We used a Raspberry PI for our honeypot system and burned a clean install of the Ubuntu Server 20 LTS OS on an SD card. During this step, we also updated the system and created the necessary user accounts for this project.

The third step of our implementation was creating a private/public rsa key pair we could use to connect remotely to our Raspberry PI system. We decided to do this because it would make our remote connection much safer than using a traditional password when connecting through ssh. Then we added the public key to our authorized key file on the new admin account we set up for the Raspberry PI server. Next we set up the config files on both the client and server side machines to make connections easier and more secure. Some of the important changes made in the server config are disabling password authentication, disabling tcp forwarding, and changing the default port for ssh. These all help make using ssh much more secure and less of a security risk.

The fourth step we made was continuing to make our remote connections to the Raspberry PI secure by enabling and configuring the firewall. To do this we used the default built-in firewall software on Ubuntu called ufw. Using this software allowed us to configure what ports to keep open and on what ips that to allow traffic in from on those ports. Most importantly, it blocked all other ports so that unwanted connections cannot be made or abused.

The next major step we made was installing the honeypot software and all of the required dependencies for that software. We utilized the official documentation to complete the installation. One of the main issues we ran into is the documentation is a bit outdated. We had to do some research into how to install the required python libraries since python has changed in updates over time.

The next implementation step our team took is we further configured our two virtual environments and started to harvest basic information off these machines in order to emulate an Ubuntu 18 server system. We used the ssh -V command to capture the correct output that a user would expect from an Ubuntu system. We alo used ssh -Q cipher and ssh -Q mac to get the list of supported ciphers and macs on a Ubuntu system. Then we used a script to gather the running processes on the system so that we could reference these in a honeypot when a user interacts with it. We also used the rsync command to copy the file system of an Ubuntu VM into the honeypot and created a PIckle file that the honeypot could use to emulate that system. Our next task we accomplished is the creation of a custom script that displays an MOTD with fake information and looks like one found in a traditional Ubuntu system. We had to configure the windows hyper-v VM to allow ssh connections to it from outside the host computer in order to utilize the rsync command which was quite a challenge.

The final step in the implementation process was to modify the config file to utilize the data we captured from our Ubuntu VM systems. Here we added the ssh version, the ciphers and macs, the path to the file system PIckle file, and the custom MOTD. Currently we left the server to allow any user to log in with root except for a few restrictions, we also created some basic accounts which could be used that the attacker could use to login with.

8. Testing

Once we had completely set up and configured our honeypot to emulate our target system, we were ready to begin testing. We started our testing by running the honeypot software and opening the port in the firewall to allow the ssh connection to the honeypot. To start the honeypot, we used the command bin/cowrie start. During implementation, we decided to keep our honeypot on the port 2222 as we felt it would make the server seem natural and approachable since that is a common ssh port. The command we used to allow this connection was sudo ufw allow 2222/tcp.

Now that we had set up our honeypot for testing, we used the nmap command to scan the Raspberry PI server and determine what ports are available. Using

this command revealed the open port 2222 and the version of ssh. From this point, we decided the next logical step as a pseudo attacker would be to attempt connecting to the system. We connected to the open port 2222 and attempted to log into the root account of the server. In the implementation, configured root login as default to accept almost any passwords. This makes it easier for the attacker to get to make the honeypot more useful for the defender. We used the credentials username: root and password: banana and the system allowed us in. Our next goal was to explore the file system and see if we could navigate around the system to see what files we could find. After that, we were disconnected from the server.

Our next goal in testing was to view the log and see what information the honeypot was able to capture from our little escapade. The server recorded the connection, credentials used to log in, commands we used, ciphers and macs, etc. Upon further investigation we found a few pieces of information we found interesting. The first was that the system recorded the version of ssh that we had used to connect. The honeypot also reached a predefined timeout and forcefully closed the connection with the attacker. We found this interesting and something worth exploring later.

9. Performance Evaluation and Feature Comparison

This honeypot offers interaction with connecting through the ssh protocol, authentication, a fake file system, and many other configurable options to make the system appear real to the attacker. As for the honeypot performance, it did a great job presenting a system that appeared to be real through the use of the honeypot's complex and functional interaction. What was enjoyable about this honeypot software is that it does not stop at ssh connection. Cowrie allows the attacker to log into a fake system and interact with it. The added benefit of this feature is that we can learn more about the tactics the attacker uses and what information they may be after. Specifically, it was enjoyable to see that the software allowed us to log in as root and explore a file system. The file system made the honeypot feel like a real system and made it interesting to interact with.

10. Lesson Learned

The main lesson we learned for our project is how we can take the offensive side in cyber security. This gave us tools we can use to defend computer networks by being able to detect network intrusions and create the possibility to respond/recover from cyber attack by providing a way to learn about our attackers. It is great to learn how we can detect network intrusions because it gives us the ability to know if our network has been compromised. We also can gain valuable information from honeypots without losing our system. In most cases, security systems are attacked, and cyber security professionals are able to learn from their mistakes and correct the vulnerabilities so it doesn't happen again, but the honeypot presents a different process. Instead we can learn about our vulnerabilities before the damage is done. Honeypots can give us valuable attacker information about an attacker and how they are getting into the system, without actually granting them access to the system.

11. Future Work

Future work for this project that we would like to pursue is to continue to configure and secure our honeypot to make it less likely to be vulnerable from attacks and also to make it more interactive and believable for the attacker. We also would like to research and implement a capture the flag type game where attackers can look for specific important files and try to gain access to that information within the honeypot. Similarly, we would like to further research some of the other honeypots and experiment with their capabilities like we did in this project. It would be very fun to compare different honeypots based on strengths and weaknesses to see what the best honeypot is for different situations and systems.

12. Suggestions

Our suggestion for anyone who is considering implementing this is to find a honeypot software that meets the needs in terms of interactivity and feature set provided by the honeypot. Similarly, we suggest testing your honeypot and experimenting with the different functions it grants.

13. Conclusion

Throughout this project, we explored the concept of computer security from a different perspective. The cyber security field is built around protection and defending but for this project we chose to investigate an offensive security method and bite back at the attacker. We found honeypots to be unique and innovative and wanted to implement our own. Through design work, multiple step implementation, and thorough testing we were able to see the capabilities of a honeypot and experience first hand how effective they can be.

Honeypots are an unique and exciting way for students to test and learn new computer security skills. While in their simplest form, honeypots can be a great tool to explore cyber security, honeypots can be a very effective tool for real world network defense. The information honeypots provide is valuable, and removes most of the risk involved trying to obtain that information.

By implementing honeypots, not only can we detect network intrusions, but we can learn about our enemies and respond to attacks before significant damage is done. We can defend an attack before it even happens with the information a honeypot provides. Attackers won't even know that they are in a fake system. With honeypots. we use the attacker's knowledge to our benefit and capture them in our gooey, sticky honeypot.

14. References

C. Sanders, *Intrusion detection honeypots: detection through deception*. Chris Sanders, 2020.

M. Valentine, “5 Life-Altering Lessons from The Art of War by Sun Tzu,” Goalcast, 19-Sep-2019. [Online]. Available: https://www.goalcast.com/2018/06/08/5-lessons-from-sun-tzus-art-of-war/. [Accessed: 09-Dec-2020].

W. Safire, “The Best Defense,” The New York Times, 14-Apr-2003. [Online]. Available: https://www.nytimes.com/2003/04/14/oPInion/the-best-defense.html#:~:text="The best defense is a,". [Accessed: 04-Dec-2020].

15. Appendix

Figure 1

```
mo@mauricio-HP-EliteBook-840-G1:~$ ssh-keygen -b 4096 -C "Lab Access" -f ~/.ssh/id_rsa_lab
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/mo/.ssh/id_rsa_lab
Your public key has been saved in /home/mo/.ssh/id_rsa_lab.pub
The key fingerprint is:
SHA256:k25v/XcSZqKd71PlfDcT7ToviFsAEHCaYPz2IyuTsho Lab Access
The key's randomart image is:
+---[RSA 4096]----+
| .o ..+.         |
| ... + .         |
|   .o   .      .|
|    o    o    .o|
|   . .  S .    +o|
|    . o. . .. +oB|
|E  . o .o  =.* +=|
|..+ .    . .+.= * o|
|+o o     .o. ++Bo|
+----[SHA256]-----+
```

Figure 2

```
cowrie@ubuntu:~$ git clone http://github.com/cowrie/cowrie
Cloning into 'cowrie'...
warning: redirecting to https://github.com/cowrie/cowrie/
remote: Enumerating objects: 75, done.
remote: Counting objects: 100% (75/75), done.
remote: Compressing objects: 100% (63/63), done.
remote: Total 13931 (delta 27), reused 30 (delta 9), pack-reused 13856
Receiving objects: 100% (13931/13931), 8.86 MiB | 3.02 MiB/s, done.
Resolving deltas: 100% (9585/9585), done.
cowrie@ubuntu:~$ cd cowrie
cowrie@ubuntu:~/cowrie$ pwd
/home/cowrie/cowrie
cowrie@ubuntu:~/cowrie$
```

Figure 3

```
mo@mauricio-HP-EliteBook-840-G1:~$ multipass launch bionic
Launched: endless-scup
mo@mauricio-HP-EliteBook-840-G1:~$ multipass list
Name                    State             IPv4             Image
endless-scup            Running           10.113.121.242   Ubuntu 18.04 LTS
mo@mauricio-HP-EliteBook-840-G1:~$ multipass info endless-scup
Name:           endless-scup
State:          Running
IPv4:           10.113.121.242
Release:        Ubuntu 18.04.5 LTS
Image hash:     04a92109979d (Ubuntu 18.04 LTS)
Load:           0.02 0.02 0.00
Disk usage:     1.0G out of 4.7G
Memory usage:   74.1M out of 985.1M
mo@mauricio-HP-EliteBook-840-G1:~$
```

Figure 4

```
ubuntu@endless-scup:~$ ssh -V
OpenSSH_7.6p1 Ubuntu-4ubuntu0.3, OpenSSL 1.0.2n  7 Dec 2017
ubuntu@endless-scup:~$ ssh -Q cipher
3des-cbc
aes128-cbc
aes192-cbc
aes256-cbc
rijndael-cbc@lysator.liu.se
aes128-ctr
aes192-ctr
aes256-ctr
aes128-gcm@openssh.com
aes256-gcm@openssh.com
chacha20-poly1305@openssh.com
ubuntu@endless-scup:~$ ssh -Q mac
hmac-sha1
hmac-sha1-96
hmac-sha2-256
hmac-sha2-512
hmac-md5
hmac-md5-96
umac-64@openssh.com
umac-128@openssh.com
hmac-sha1-etm@openssh.com
hmac-sha1-96-etm@openssh.com
hmac-sha2-256-etm@openssh.com
hmac-sha2-512-etm@openssh.com
hmac-md5-etm@openssh.com
hmac-md5-96-etm@openssh.com
umac-64-etm@openssh.com
umac-128-etm@openssh.com
ubuntu@endless-scup:~$
```

Figure 5

```
ubuntu@endless-scup:~$ ps -eo pcpu,%mem,pid,rss,start_time,stat,bsdtime,tty,user,vsz,args |
> egrep -v 'ps -eo|jq|egrep|awk' | awk '{for(i=1;i<=10;i++){printf"%s\t",$i};out=$11; for(i=12;i<=NF;i++){out=out
 "$i}; print out}' |
> jq -s --slurp --raw-input --raw-output 'split("\n") | .[1:-1] |
> map(split("\t")) | map({"COMMAND": .[10], "CPU": .[0]|tonumber, "MEM":
> .[1]|tonumber, "PID": .[2]|tonumber, "RSS": .[3]|tonumber, "START":
> .[4], "STAT": .[5], "TIME":  .[6], "TTY": .[7], "USER": .[8], "VSZ":
> .[9]|tonumber})| { "command": { "ps": .}}'
```

Figure 6

```
# Cipher encryption algorithms to be used.
#
# MUST be supplied as a comma-separated string without
# any spaces or newlines.
#
# Use ciphers to limit to more secure algorithms only
# any spaces.
# Supported ciphers:
#
# aes128-ctr
# aes192-ctr
# aes256-ctr
# aes256-cbc
# aes192-cbc
# aes128-cbc
# 3des-cbc
# blowfish-cbc
# cast128-cbc
ciphers = 3des-cbc,aes128-cbc,aes192-cbc,aes256-cbc,aes128-ctr,aes192-ctr,aes256-ctr
```

Figure 7

```
{
  "command": {
    "ps": [
      {
        "COMMAND": "/sbin/init",
        "CPU": 0.1,
        "MEM": 0.4,
        "PID": 1,
        "RSS": 8924,
        "START": "17:34",
        "STAT": "Ss",
        "TIME": "0:04",
        "TTY": "?",
        "USER": "root",
        "VSZ": 77896
      },
      {
        "COMMAND": "[kthreadd]",
        "CPU": 0,
        "MEM": 0,
        "PID": 2,
        "RSS": 0,
        "START": "17:34",
        "STAT": "S",
        "TIME": "0:00",
        "TTY": "?",
        "USER": "root",
        "VSZ": 0
      },
```


Figure 8

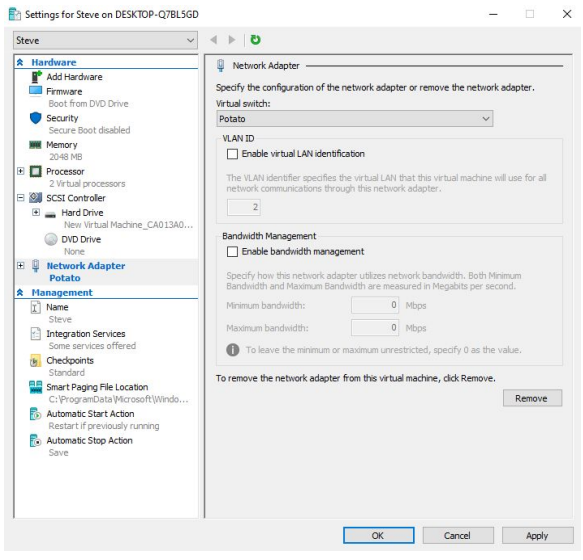

Figure 9

```
# ====================================================================
# Shell Options
# Options around Cowrie's Shell Emulation
# ====================================================================

[shell]

# File in the Python pickle format containing the virtual filesystem.
#
# This includes the filenames, paths, permissions for the Cowrie filesystem,
# but not the file contents. This is created by the bin/createfs utility from
# a real template linux installation.
#
# (default: fs.pickle)
filesystem = ${honeypot:share_path}/ubuntu.pickle
```

Figure 10

```
cowrie@ubuntu:~/cowrie$ bin/cowrie start

Join the Cowrie community at: https://www.cowrie.org/slack/

Using default Python virtual environment "/home/cowrie/cowrie/cowrie-env"
Starting cowrie: [twistd  --umask=0022 --logger cowrie.python.logfile.logger --pidfile=var/run/cowrie.pid cowrie ]...
cowrie@ubuntu:~/cowrie$
```

Figure 11

```
mauricio@mauricio-HP-EliteBook-840-G1:/home/mo$ sudo nmap -sV 192.168.0.9 -Pn
Starting Nmap 7.80 ( https://nmap.org ) at 2020-12-09 14:16 MST
Nmap scan report for 192.168.0.9
Host is up (0.00040s latency).
Not shown: 999 filtered ports
PORT     STATE SERVICE VERSION
2222/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3, OpenSSL 1.0.2n  7 Dec 2017 (Ubuntu Linux; protocol 2.0)
MAC Address: B8:27:EB:57:98:DB (Raspberry Pi Foundation)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 13.74 seconds
mauricio@mauricio-HP-EliteBook-840-G1:/home/mo$
```

Figure 12

```
mauricio@mauricio-HP-EliteBook-840-G1:~$ ssh -p 2222 root@192.168.0.9
The authenticity of host '[192.168.0.9]:2222 ([192.168.0.9]:2222)' can't be established.
RSA key fingerprint is SHA256:FF+IzhyjdmC5SrIVoPjxtUZXc4vUjW8Cy34doqONXsY.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[192.168.0.9]:2222' (RSA) to the list of known hosts.
Ubuntu 18.04.5 LTS
root@192.168.0.9's password:
root@svr04:~#
```

Figure 13

```
root@svr04:~# ls
root@svr04:~# ls -la
drwx------ 1 1002 1002 4096 2020-11-23 17:02 .
d--------- 1 root root    0 1970-01-01 00:00 ..
root@svr04:~# cd ..
root@svr04:/# ls
bin   boot  cdrom  dev   etc   home  lib   lib64  lost+found media
mnt   opt   proc   root  run   sbin  snap  srv    sys         tmp
usr   var
root@svr04:/# cd home
root@svr04:/home# ls
sandstorm
root@svr04:/home# cd sandstorm
root@svr04:/home/sandstorm# ls -la
drwxr-xr-x 1 1002 1002 4096 2020-12-09 18:07 .
drwxr-xr-x 1 1002 1002 4096 2020-11-23 17:02 ..
-rw------- 1 1002 1002 1426 2020-12-09 19:36 .bash_history
-rw-r--r-- 1 1002 1002  220 2018-04-04 18:30 .bash_logout
-rw-r--r-- 1 1002 1002 3771 2018-04-04 18:30 .bashrc
drwx------ 1 1002 1002 4096 2020-11-23 17:07 .cache
drwx------ 1 1002 1002 4096 2020-11-23 17:07 .gnupg
-rw-r--r-- 1 1002 1002  807 2018-04-04 18:30 .profile
drwx------ 1 1002 1002 4096 2020-11-24 20:18 .ssh
-rw-r--r-- 1 1002 1002    0 2020-11-23 17:30 .sudo_as_admin_successful
-rw------- 1 1002 1002 7007 2020-12-09 18:07 .viminfo
root@svr04:/home/sandstorm# Connection to 192.168.0.9 closed by remote host.
Connection to 192.168.0.9 closed.
mauricio@mauricio-HP-EliteBook-840-G1:~$
```

Figure 14

```
cowrie@ubuntu:~/cowrie$ ls
CHANGELOG.rst     LICENSE.rst  README.rst  cowrie-env  honeyfs                requirements.txt  src
CONTRIBUTING.rst  MANIFEST.in  backupfs    docs        requirements-dev.txt   setup.py          tox.ini
INSTALL.rst       Makefile     bin         etc         requirements-output.txt  share           var
cowrie@ubuntu:~/cowrie$ cd var/log/cowrie
cowrie@ubuntu:~/cowrie/var/log/cowrie$ ls
cowrie.json  cowrie.log
cowrie@ubuntu:~/cowrie/var/log/cowrie$ vim cowrie.log
cowrie@ubuntu:~/cowrie/var/log/cowrie$
```

Figure 15

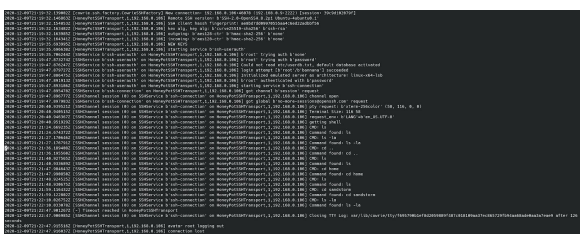

Figure 16