Honeypot Research

Jacob Ingraham
Department of Computer Science
Colorado State University
Fort Collins, Colorado

jacob.ingraham@colostate.edu

***Abstract*— A prominent issue within computer security is the vast array of techniques for attacking systems. The best way to combat this is with data about common attacks. Using a honeypot server, I will collect and analyze data about SSH attacks.**

***Keywords*— Cowrie, cybersecurity, honeypot, SSH**

I. Introduction

For this project I set up a honeypot server using an Amazon Web Services (AWS) virtual machine. Honeypots are systems set up to attract attackers for a variety of reasons. They are often used to pull attackers' attention away from valuable systems or used to gather information about attackers and their common attack patterns. Using the open-source honeypot program, Cowrie, data was logged from all attempted attacks. Cowrie forms connections with potential attackers through Secure Shell (SSH). SSH is a network protocol which creates encrypted connections between systems. It works on top of the Transmission Control Protocol (TCP) and Internet Protocol (IP) which handle the delivery of data packets across internet connections. SSH tunnels traffic to and from designated ports which computers use to communicate. Attackers often use SSH to attack target systems through these ports. Cowrie acts as a go-between to protect the internal system and record all activity.

To host this honeypot server, I used AWS. They offer a large variety of virtual machines (VMs) which are perfect for this project. Using a VM to host a honeypot reduces much of the risk involved. Because honeypots are meant to attract attackers, they can be used as a launch point for attacks if they are improperly configured. This is not a problem when using a virtual machine because it is completely separated from the underlying hardware. This means that even an attack getting past the security measures will not result in permanent damage to the system.

The VM I chose to use was the Amazon Elastic Compute Cloud (EC2). This type of virtual machine specialized in scalability so that it can handle spikes in computation or traffic while still saving space. Since I was using the free tier of AWS this proved to be a great option for my needs. I was able to keep it running long enough to gather sufficient data without exceeding the monthly limits set on my account.

After an instance of this VM was created which simulates a Linux environment, the next step was to install and configure a honeypot server. Using Cowrie this was relatively simple and took little time. The EC2 instance ran this server for one week while it collected attack data. This data included all IP addresses that attempted to connect through SSH. Cowrie also recorded all username and password combinations that were attempted. There were many combinations which would appear to give access to the system so that Cowrie could log any shell commands they attempted to execute.

Using the log files, I was able to deduce much information about the attacks which were attempted on the honeypot system. The timing of the attacks gives clues about the type of entity attempting to gain access. The patterns of username and password combinations can indicate how the attacks are being conducted.

II. Description of Setup

My original plan was to use Kippo as an SSH honeypot. This program seemed to provide the most functionality and was highly recommended by a variety of sources. However, when setting it up on my EC2 instance I was unable to get a successful build. After a lot of time trying to fix broken and missing Python dependencies as well as other errors, I decided my time was better spent trying a different honeypot. Cowrie was designed based on Kippo and provided much of the same functionality, including everything I would need for this project. It did not take much time to get Cowrie functional and it was soon ready for connections from the internet.

At this point more issues arose surrounding the EC2 instance. AWS runs their virtual machines within a set of security groups which filter access. These groups can be set to manage inbound or outbound traffic based on source or destination (for inbound or outbound respectively), port, and protocol. VM instances can also use key pairs generated by asymmetric encryption algorithms as further protection. While setting up the honeypot this security group was configured to only allow connections from my personal computer's IP address when using my RSA key. Once I was ready for Cowrie to receive connections from the outside world this security group blocked access attempts which violated these rules. To fix this there were two settings I needed to change. First, within the security group itself I needed to allow access from any IP on the port that Cowrie was running on. Second, I needed to adjust the SSH settings within my instance to request a username and password on this port instead of using a key pair to authenticate.

Once these changes were made Cowrie was reachable through the internet. However, it was listening on port 2222 so as not to interfere with my remote connection to the VM on the default SSH port 22. This was an easy fix using port forwarding with Authbind. Port forwarding is a way to redirect internet traffic to different ports on a machine. I needed to redirect port 22 to port 2222 and vice versa. This would allow default SSH connections to reach Cowrie at port 22. Non-root users cannot normally bind to ports 0-1023 since they are reserved for privileged processes. Authbind allows non-root users to access privileged network services, so this port forwarding would be possible. It was a simple process to download and configure Authbind, then edit the appropriate port numbers. Within Cowrie's settings I changed the listening port to 22 and within the EC2 instance's SSH settings I changed the SSH service listening port to 2222. Finally, I swapped the port numbers in the security group so that Cowrie was reachable from anywhere on port 22 and accessing the VM was still restricted to my IP address on port 2222.

To make all these changes permanent I created an image from the running EC2 instance. This saved the current state of the VM so that any server reboot or other interruption would not reset the settings that were changed. This also ensured that Cowrie would be restarted any time such an interruption happened. Since AWS may restart their servers and move VMs to different servers at any time this step was very important.

At this point Cowrie was ready and all I had to do was let my instance run for an appropriate amount of time. After a few days I was ready to start pulling logs and begin my analysis.

III. Description of Results

The Cowrie server ran for a total of 7 days. It was configured to reject all username and password combinations except for most login attempts using root as the username. The only passwords which didn't work when using root as the username were 123456 and root.

The following describes all significant attack patterns as well as my observations about them. They are listed in descending order of attack frequency. When discussing username and password combinations the format username:password will be used (e.g. root:pass123).

A. "Bye Bye" Attack Pattern

The first attack to be carried out on the Cowrie server happened a few hours after it was made publicly accessible. This attack pattern was the most common attack during the data collection, with at least a couple hours of consistent attacking every day. It was very likely carried out by a brute force tool because most of the access attempts took less than 3 seconds. It had a very predictable pattern which made it very easy to analyze. The pattern was the same for all appearances of the attack so the following will break down the first attack, which was the most extensive and continued for much of the day.

The most important factor which made this attack very easy to recognize was the custom disconnection message. Whenever an SSH session is terminated by the attacker Cowrie will log "Got remote error, code 11 reason: disconnected by user" or "Connection lost". In the case of this attack the message always read "Got remote error, code 11 reason: Bye Bye". I was not able to find any information about this message, so I am assuming it is caused by the way this specific tool severs the SSH connection. This message made the pattern very easy to analyze despite attacks from many different IPs.

Even without a distinguishing message I could have connected many of the attempts submitted by this tool. It tried a variety of username:password combinations, all of which (from the first attack) are listed in Table 1. The later attacks used some repeating values and a variety of new ones as well.

Because one of the passwords attempted was "#changeme#" and "changeme" also showed up in a later attack, I believe this tool uses a set of username:password combinations as input. This theory is further supported by other passwords attempted which draw patterns across the keyboard. "1q2w3e4r5t" draws a zigzag across the top left of a standard keyboard, "qazwsxedc" draws lines on the left side, and "4rfv4rfv" draws a line from 4 down to v twice. Of course, it is also possible that all of these have been real passwords that the tool encountered. However, a tool creating access attempts from passwords encountered in the past would likely have more diversity in the combinations with usernames. This pattern was very consistent with only guessing certain passwords with certain usernames and vice versa. There also seems to be two different input sets. Exactly halfway through this first attack the attempts stop for about 3 minutes, then continue with the inputs being very different. Before the break (left column of Table 1) most of the attempts involve words and patterns. After the break (right column of Table 1) most of the usernames are short combinations of random letters and the passwords are usually the same as the username or "123456".

TABLE I. "Bye Bye" Username:Password Combinations

Username:Password	
tomcat:tomcat@123	ngrk:123456
usuaria:root	pwim:123456
peer:peer	tkyx:tkyx
root:webadmintest	swpw:123456
345gs566wd34:345gs566wd34	fqyy:fqyy
root:345gs566wd34	jens:jens
marty:marty	duqm:duqm
klog:123456	dev2:dev2
stu1:stu1	ptfn:ptfn
zhangyun:zhangyun	lgga:123456
ex:1111	rbsg:123456
root:Password123456	admin:admins
root:Support2022#	wyyn:wyyn
root:1234%asd	wfjy:123456
cerguest:cerguest	pudh:123456
witchupan:witchupan	fpdg:123456
oracle:123!@#	jtpg:123456
root:Ek8jt?*l?Yrt	cfry:123456
bytenest_dev:bytenest_dev	bmgk:bmgk
root:abcdefghijklmn	ptrn:ptrn
root:opel1	ylog:ylog
admin:a123456789	tuhh:tuhh
root:4rfv4rfv	root:qwert123456789
cer-user:123	yjwg:yjwg
cris:123	root:a123
root:b.321	whgh:123456
root:1q2w3e4r5t	bmpg:123456
junior:junior	jvpe:jvpe
root:mypass1	ptfn:ptfn
root:Huwei!@#	dfgdfg:dfgdfg
cr4zyg0d:tomcat	ynmq:123456
root:terminal12#	yiyf:123456
webdev01:webdev01	root:Abc#123
root:Asd2021	isut:123456
testuser:testuser	rpim:123456
root:#changeme#	root:terminal#123
elizabeth:elizabeth123	test2:123456

test:qazwsxedc	tpb:123456
root:p@$$v0rd1	gsi:123456
sysadmin:sysadmin	jg:123456
cer2:123456	ur:123456
root:nopass123	ansible:123456
root:aa123456	ezm:123456
author:author	xya:123456

Fig. 1. Order is top to bottom (left), then top to bottom (right)

This tool also seems to be a very naive approach to brute forcing. The attempts use a large degree of randomness which only decreases the coverage. For example, many username:password combinations had to be removed from Table 1 because they are exact duplicates of earlier attempts. Some of these identical attempts were even tried multiple times in a row.

The use of differing IP addresses was also random. Many IPs were used only once before a new one attempted access, while others were used up to 12 times in a row. Because this attack pattern never attempted access from two IPs at the same time, I am led to believe that it is using IP address spoofing. However, a botnet is also very likely.

Since many of the attempted logins used root with a password that succeeded, there were many shell sessions logged as well. Whenever an access attempt succeeded the tool would execute the two commands listed below in Table 2. It would then, while the successful connection was still active, attempt to connect using "345gs5662d34:345gs5662d34". After this failed, it would immediately attempt "root:345gs5662d34". Because of the settings on the server this most recent attempt would succeed, but no more commands were input. The tool would simply disconnect both successful connections and continue testing randomly.

TABLE II. "BYE BYE" SHELL COMMANDS

Command Number	**Commands**
1	cd ~; chattr –ia .ssh; lockr –ia .ssh
2	cd ~ && rm -rf .ssh && mkdir .ssh && echo "ssh-rsa AAAAB3NzaC1yc2EAAAABJQAAAQEArDp4cun21hr4KUhBGE7VvAcwdli2a8dbnrTOrbMz1+5073fcBOx8NVbUT0bUanUV9tJ2/9p7+vD0EpZ3Tz/+0kX34uAx1RV/75GVOmNx+9EuWonvNoaJe0QXxzilg9eLBHpgLMuakb5+BgTFB+rKJAw9u9FSTDengvS8hX1kNFS4Mjux0hJOK8rvcEmPecjdySYMb66nylAkQwCEE6WEQHmd1mUPgHwGQ0hWCwsQk13yCGPK5w6hYp5zYkFnvlC8hGmd4Ww+u97k6pfTGTUbJk14ujvcD9iUKQTTWYYjllu5PmUux5bsZ0R4WFwdle6+j7rBLAsPKgASSVKPRK+oRw== mdrfckr:>>.ssh/authorized_keys && chmod -R go= ~/.ssh && cd ~

Reference [1] discusses a honeypot which logged many attackers attempting to execute commands very similar to these. The first thing I noticed when comparing the commands is that the attacks on my server reverse the order of the two commands. The combination of chattr and lockr in [1] was part of the second command, just executed at the end. They seem to only work as intended when executed after the second command. However, in sources such as [3] these commands appear in the exact order and format as this attack pattern. These exact commands being recorded by many other honeypots supports the theory that a popular attack tool is being used.

These commands have the following purposes. The chattr (change attribute) command is used to change the access rights on files, with the –ia options making the affected files append only and the changes irreversible. The file extension (.ssh) is meant to affect the SSH configuration directory. The lockr command is part of a subscription based SSH key management system. Since the Cowrie server had no such system installed it was listed as unrecognized in the log files. The second command attempts to replace the SSH configuration directory with the attacker's RSA key. The format for inserting the attacker's RSA key is: "AAAAB3NzaC1(keydata)...+oRw" [1] which means the long string in the middle is meant to be the attacker's key. If successful, this command will remove the original user's access to the system and allow the attacker unlimited access through their RSA key. If this is followed by the chattr command as described by [1] these changes will be permanent. However, since the chattr command is executed first this is not the case. Instead, the file is adjusted to be append only right before it is deleted and replaced. Note: when discussing the RSA key I used the term "meant to be". This will be discussed more in (C. System Scanning Attack Pattern).

Another interesting detail is the very commonly used "345gs5662d34" string. As mentioned previously, this was always used after a successful connection. It took very little research to discover that this value was a subject of confusion for other researchers working with honeypots [2]. One of the comments on [2] voiced the idea that it could be a translation from a non-English keyboard. They reference a similar string which was appearing very consistently a few years ago, "ji32k7au4a83". This turned out to be a translation of "my password" in Taiwanese. This "345gs5662d34" string is only used as both username and password, or with root as the username. This further confirms the suspicion voiced by another commenter who claims this might be a translation of root in another language [2].

B. Iterative and Dictionary Attack Patterns

- There were many attacks that tried different username:password combinations by holding one value constant and iterating through possible options for the other value. These will be referred to as Iterative attacks. They were often used along with a form of Dictionary attack. A Dictionary attack will typically try words from a dictionary one after the other, hoping to stumble upon a working credential. The Dictionary attacks that hit the Cowrie server were more intuitive than this and tended to guess security related words. These attacks were almost always launched by attack tools because they were able to make many attempts in just a few seconds.

An example attack which used these techniques tried the following passwords, all with the username pi: Raspberry, pi, raspberry, 123, toor, 12345678, ubuntu, oracle, admin. Since pi is the default username on Raspberry Pi devices this attack is likely very successful against many such systems. People do not always change their passwords when setting up a new device and this attack takes advantage of this. Another example used the following usernames: 123456, user, ansible, oracle, postgres, es, esuser, aws, ec2-user, vagrant, jenkins, git. For each of these it tried both 123456 and the username as passwords. Since many of these values are related to AWS and EC2 instances this specific attack might specifically target AWS virtual machines.

These attacks were not often successful, but some of them did try root as a username and gained access. As was expected of such a simplistic algorithm they would only run one command before exiting. This command was uname with a few options. The options varied but were usually 2-4 of the following: -a, -s, -m, -o. The Linux command uname returns information about the underlying architecture. The –a option is short for all and will print all available information. The –s option returns the kernel name, -m returns the architecture type, and –o returns the operating system name. This command is commonly used by scanning tools to pinpoint machines to attack further.

C. System Scanning Attack Pattern

The next attack pattern uses scanning techniques like those in (section B) but employs a more advanced process to gain more information and privileges. The process of this attack has a lot of built in redundancy to ensure that damage to the target system is always maximized. This pattern also seems to be the work of a tool because all commands were executed in less than 2 seconds once access was gained. Successful access attempts usually used a random string of letters, numbers, and special characters as a password. This lack of a distinguishing pattern made it difficult to determine what method of brute force was used. This was also a relatively uncommon attack, with 24 attacks during the 7 days the server was running.

TABLE III. SYSTEM SCANNING SHELL COMMANDS

Command Number	Commands
1	cd ~; chattr –ia .ssh; lockr –ia .ssh
2	cd ~ && rm -rf .ssh && mkdir .ssh && echo "ssh-rsa AAAAB3NzaC1yc2EAAAABJQAAAQEArDp4cun21hr4KUhBGE7VvAcwdli2a8dbnrTOrbMz1+5073fcBOx8NVbUT0bUanUV9tJ2/9p7+vD0EpZ3Tz/+0kX34uAx1RV/75GVOmNx+9EuWonvNoaJe0QXxzilg9eLBHpgLMuakb5+BgTFB+rKJAw9u9FSTDengvS8hX1kNFS4Mjux0hJOK8rvcEmPecjdySYMb66nylAkQwCEE6WEQHmd1mUPgHwGQ0hWCwsQk13yCGPK5w6hYp5zYkFnvlC8hGmd4Ww+u97k6pfTGTUbJk14ujvcD9iUKQTTWYYjllu5PmUux5bsZ0R4WFwdle6+j7rBLAsPKgASSVKPRK+oRw== mdrfckr:>>.ssh/authorized_keys && chmod -R go= ~/.ssh && cd ~
3	cat /proc/cpuinfo | grep name | wc –l
4	echo “root:uUoZve86Mpkz”|chpasswd|bash
5	rm –rf /tmp/secure.sh; rm –rf /tmp/auth.sh; pkill –9 secure.sh; pkill –9 auth.sh; echo > /etc/hosts.deny; pkill –9 sleep;
6	cat /proc/cpuinfo | grep name | head –n 1 | awk ‘{print $4,$5,$6,$7,$8,$9;}’
7	free –m | grep Mem | awk ‘{print $2, $3, $4, $5, $6, $7}’
8	ls –lh $(which ls)
9	crontab –l
10	w
11	uname –m
12	cat /proc/cpuinfo | grep model | grep name | wc –l
13	top
14	uname
15	uname -a
16	whoami
17	lscpu | grep Model
18	df –h | head –n 2 | awk ‘FNR == 2 {print $2;}’

Once access was gained the first two commands were identical to that of the “Bye Bye” attacks. Although these commands are common among SSH attacks there are many interesting similarities which were not expected. Firstly, this attack pattern also executed the commands in the incorrect order. As mentioned before, the shorter command is meant to be appended to, or executed after, the longer command so that it functions as intended. Secondly, the commands being identical means that the RSA keys were a perfect match. Unless these two attacks were carried out by the same entity, one or both attacker(s) is using this command incorrectly. These two attacks coming from the same source is fairly probable, but it is also likely that both attackers are using downloaded attack tools. This is because the rest of the commands this attack executed were also commonly found on honeypots [1, 3, 4]. Failing to edit the keys contained within these tools would be an easy mistake.

Command numbers 4 and 5 in Table 3 serve as another way to create a backdoor into the system while restricting the access of the original user. By sending the output of echo “root:uUoZve86Mpkz” to chpasswd in bash the attacker hopes to change the root password to the string in the command. Next, the rm –rf and pkill commands are executed to delete important script files/directories as well as the processes they spawned. Also included in command 5 is echo > etc/hosts.deny which simply creates a file. This file seems to be used to blacklist certain IP addresses from accessing the system, but no IPs are added to it.

The rest of the commands return a lot of important information about the system. This information can be used by the attacker to determine how useful the system would be in a botnet or similar network. The access to this compromised machine may be sold on the dark web based on how valuable its computing power is [3].

The attacker gets some information about the target system’s CPU(s) by reading from the /proc/cpuinfo file in three ways. The first version of this command (command 3 in Table 3) filters the output of cat with grep name. This will search for lines which contain the string “name”, returning one line for each CPU the system has. The output is filtered one more time using wc –l. This command usually returns the number of words, but the –l option instead returns the number of newlines. Thus, the entire command returns the number of CPUs the system has. Command 12 in Table 3 also calls cat /proc/cpuinfo but searches with grep for the string “model” in addition to “name”. It also uses wc –l to return the number of lines found by grep. This command usually returns the same data as command 3 but is likely included as a form of redundancy. The last command which reads this file (command 6 in Table 3) also uses grep name, but filters with head –n 1. This will return the first row, which will be the first CPU in the file. This output is passed to awk which prints columns 4, 5, 6, 7, 8, and 9. These columns contain information about the model of the CPU.

Command 7 in Table 3 uses free –m to fetch information about the memory usage within the system. This can be a good indication of how useful the system would be as a crypto-currency miner [4]. The ls command (command 8 in Table 3) returns information about the location ls is installed. Such an integral shell process will likely be in a root directory so there are many reasons an attacker would want this information. Cron is the name of a daemon which often handles job scheduling in Linux. Instructions for this daemon are stored in a file called Crontab and the attacker attempts to pull information from this file using the –l option (command 9 in Table 3). The very simple command, w, (command 10 in Table 3) displays all users currently logged into the system, jobs they are running, percentage of CPU those jobs are taking up, and more. The attacker may place less value on a machine whose workload is high, because less CPU and memory will be available. Uname is run a few times (commands 11, 14, and 15 in Table 3), with no options (which is the same as –s) as well as with –m and –a. The top command (command 13 in Table 3) shows a dynamic table of running processes. This table contains process ID, user, percentage of CPU, percentage of memory, command which spawned it, and more. Command 16 in Table 3, whoami, displays the current user. Calling lscpu | grep Model (command 17 in Table 3) is another way to fetch information about the CPU model. The last command is number 18 in Table 3. First, df –h prints information about the filesystem in human readable format and head –n 2 filters output to the first 2 lines. This output is passed to awk which further filters the output down using FNR == 2 before printing the second column. Since FNR == 2 only returns the second row the command likely aims to only fetch the data and remove the header. The second column of this row when I tested the command contained the size of the filesystem, but this may not be consistent between all system types.

The amount of information the attacker gathers with these commands is concerning. Using this data, they can decide how to best use the compromised machine. Almost every command executed also has at least one backup in case of failure. Since the first two commands are not executed correctly, the redundancy of attempting to create another backdoor is very useful to the attack. It is also worth noting that the exact spacing of the commands (specifically commands 6 and 7 in Table 3) is the same as the examples I was able to find in [4]. This is further indication that the attack comes from a popular attacking program.

D. *Modem Cryptocurrency Mining Attack Pattern*

This attack pattern was unique from the others in multiple ways. The main reason is that it seems to target a completely different set of devices than the other attacks, Wi-Fi modems. It also was the only pattern which seemed to check for the existence of a honeypot and behave differently if one was found. This attack only connected to the Cowrie system 5 times during the 7 days and used very simple access attempts. It would try root:root followed by root:admin. Since this access pattern worked so quickly it would have been interesting to see what else it tried. Like all other attack patterns I have analyzed, the speed of access attempts and command executions (less than 3 seconds) indicate another tool being used.

TABLE IV. MODEM CRYPTOCURRENCY MINING SHELL COMMANDS

Command Number	Commands
1	/Ip cloud print
2	ifconfig
3	uname -a
4	cat /proc/cpuinfo
5	ps | grep '[Mm]iner'
6	ps –ef | grep '[Mm]iner'
7	ls -la /dev/ttyGSM* /dev/ttyUSB-mod* /var/spool/sms/* /var/log/smsd.log /etc/smsd.conf* /usr/bin/qmuxd /var/qmux_connect_socket /etc/config/simman /dev/modem* /var/config/sms/*
8	echo Hi | cat –n

Once access to the system was established the first command to be executed was /ip cloud print (command 1 in Table 4). This command is native to a specific operating system used by a brand of modems, MicroTik. This company is based in Latvia, but popular in the US. They develop all types of network equipment and are used by large corporations such as Panasonic and Ebay [5]. This command has since been deprecated so it will only work on outdated versions of MikroTik's RouterOS. If successful, this command will return information about the parameter set being used by RouterOS. The next command (command 2 in Table 4) is ifconfig. This prints a variety of information about the network interface. This pattern next executes uname (command 3 in Table 4) with the –a option to get information about the operating system and hardware. Command 4 in Table 4 was seen in other patterns, cat /proc/cpuinfo, but this attack uses no filtering. Next, commands 5 and 6 in Table 4 use ps with the output being filtered by grep. The ps command returns active processes and is executed with and without the –ef options. Option –e selects all processes (even hidden ones) and –f prints full size listings. Output is sent to grep to search for the strings "Miner" and "miner". This command seems to be searching for crypto-mining processes that are running on the system. Command 7 in Table 4 searches for a specific set of files with ls. The –l option prints long format and –a searches for hidden files. The files this command is trying to find are related to SMS, GSM, SIM, and similar services. Short Message Service (SMS) is used to send short text messages between mobile devices. The Global System for Mobile Devices (GSM) is a set of protocols also used by most mobile devices. Subscriber Identity Module (SIM) cards are used by cell phones to store all the information needed to uniquely identify a device. All these files would be very valuable to an attacker. The last command (number 8 in Table 4) is echo Hi | cat –n. All this does is print "Hi" to standard output within the shell. It is not clear what the attacker hopes to accomplish through this, but a likely theory is that the attacking software is marking the target machine. "The mark could be a flag to prevent the botnet from dropping their malware on research honeypots." [6]. It is very possible that this command only executes when the software detects that the target system is a honeypot.

This attack appears to be looking for vulnerable internet modems to plant cryptocurrency mining malware on. This conclusion is based on the nature of the commands as well as the findings in [6]. This attack seems to largely target MikroTik modems. However, it is likely that many types of modems may be vulnerable to such attacks. MicroTik specifically seems to have had problems with these attacks in the past. Some research done in 2018 warns that "routers were being compromised by Coinhive cryptocurrency malware" [6]. Shortly afterwards the FBI put out a similar announcement warning of the dangers of VPNFilter malware [6]. Since this attack seems to deal with planting malware, the idea that it avoids uploading the malware to honeypots is very plausible. The commands in Table 4 were run on the Cowrie server, but no files were uploaded.

E. *Commonly guessed usernames and passwords*

The following table contains the 5 most guessed usernames and passwords for the entire experiment.

TABLE V. COMMONLY GUESSED USERNAMES AND PASSWORDS

Order	Username (# of attempts)	Password (# of attempts)
1	root (825)	345gs5662d4 (677)
2	345gs5662d4 (344)	123456 (282)
3	admin (87)	123 (47)
4	pi (31)	admin (19)

5	ubuntu (25)	password (18)

As seen in this table the attack that hit the Cowrie server the most often, "Bye Bye", had a large effect on the most guessed usernames and passwords. This attack also attempted the same two username and password combinations after every successful attempt. Below the first few rows in the table the frequency of guesses drops drastically. This is because most of the attack patterns recorded contained a large degree of randomness in their guesses.

IV. Conclusions

Aside from a few roadblocks at the beginning, this project went smoothly. Going into this I had little experience with honeypot systems and was not sure what to expect. However, the data collected by Cowrie led me to many interesting observations.

The fact that most, if not all, of the attacks were carried out by computer programs was not surprising. Brute forcing into password-protected systems is a job very well suited for attack tools. They can submit dozens of access attempts and shell commands faster than it would take a human to submit one. They can also be executed on multiple machines at a time, oftentimes through a form of botnet. This will boost the efficiency of the attack to make it much more dangerous.

It also makes sense that many attacks share common characteristics. Commands like uname were used by all but one of the patterns discussed. Although the reasons for attackers launching these brute force attacks may be different, information about the target machine is nearly always useful.

Other similarities, such as identical RSA keys between the two patterns were more surprising. There are many possible reasons that this happened. The attack patterns could both have been launched by the same entity. This is very probable since multiple tools would increase the overall coverage of brute force attacks, which increases the effectiveness. The entities using one or both attack tools could have made a mistake. This is also very probable because there is a lot of evidence that these are popular attack tools, likely downloaded from the dark web. The attackers using these tools may not fully understand how they work and what parts of the program will need editing. This is further evidenced by the "changeme" values which may not have been intentional. Regardless, this was an interesting detail.

It was also interesting to find the attack which appeared to target Wi-Fi modems. SSH is a service which is used by a wide variety of devices, so brute force attacks using those channels will affect all of them. A modem being compromised can be dangerous for all connected devices as well. Although the attack pattern appeared to be interested in installing malware to mine cryptocurrency, it also looked for files relating to important mobile device systems. Files relating to SMS, GMS, and SIM cards can reveal a wide variety of sensitive information about the mobile devices that have sent packets through the compromised modem.

Overall, this project taught me a lot about the SSH attack space on the internet. Although there are countless systems in place to keep online devices and data protected, there are attackers attempting to break these systems at all hours of the day. As IOT devices become more commonplace, the attack space for these tools is only getting bigger. All devices which send and receive data using the internet need to be properly protected.

References

[1] "Honeypots: Know your Adversary." Cyber Defense and Research Inc. https://www.cyderinc.net/server-administration-ins-and-outs/honeypots-know-your-adversary (Accessed Apr. 30, 2024).

[2] J. La Grew. "Common usernames submitted to honeypots." SANS Technology Institute. https://isc.sans.edu/diary/Common+usernames+submitted+to+honeypots/30188. (Accessed Apr. 30, 2024).

[3] Sanseo. "Threat Actors Installing Linux Backdoor Accounts." ASEC. https://asec.ahnlab.com/en/61185/. (Accessed Apr. 30, 2024).

[4] S. Bell. "Cryptojacking Attacks Continue To Target SSH Servers." Secure Honey. https://securehoney.net/blog/cryptojacking-attacks-continue-to-target-ssh-servers.html. (Accessed Apr. 30, 2024).

[5] "Companies using MikroTik." Enlyft. https://enlyft.com/tech/products/mikrotik. (Accessed Apr. 30, 2024).

[6] Malwaremily "Honeypot Logs: A Botnet's Search for MikroTik Routers." Medium. https://malwaremily.medium.com/honeypot-logs-a-botnets-search-for-mikrotik-routers-48e69e110e52. (Accessed Apr. 30, 2024).