

BPFDoor - Part 2 - The Present

By haxrob

Published: 2025-06-02 · Archived: 2026-04-05 19:07:03 UTC

Despite the venerable **BPFDoor** malware has once again found itself in the [media spotlight](#). [Recent variants](#) avoid existing detections, so we will take a look at samples found in significant telecommunications provider breach in April 2025.

Detection evasion improvements

We will be using two following samples as references

```
cf2d3d9e0246a3220d7c3cc94257447085911b32e1de0aee9d4af7dd6427597d
3f6f108db37d18519f47c5e4182e5e33cc795564f286ae770aa03372133d15c4
```

What's changed

```
$ lsof -p 1488
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
dbus-daem 1488 root cwd DIR 8,5 4096 917506 /home/remnux
dbus-daem 1488 root rtd DIR 8,5 4096 2 /
dbus-daem 1488 root txt REG 0,25 36480 3 /dev/shm/kdmtmpflush (deleted)
dbus-daem 1488 root mem REG 8,5 2029592 3017082 /usr/lib/x86_64-linux-gnu/libc-2.31.so
dbus-daem 1488 root mem REG 8,5 191504 3015536 /usr/lib/x86_64-linux-gnu/ld-2.31.so
dbus-daem 1488 root 0u CHR 136,1 0t0 4 /dev/pts/1
dbus-daem 1488 root 1u CHR 136,1 0t0 4 /dev/pts/1
dbus-daem 1488 root 2u CHR 136,1 0t0 4 /dev/pts/1
dbus-daem 1488 root 3u pack 27663 0t0 IP type=SOCK_RAW
$
$ lsof -p 1554
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
/usr/sbin 1554 root cwd DIR 8,5 4096 2 /
/usr/sbin 1554 root rtd DIR 8,5 4096 2 /
/usr/sbin 1554 root txt REG 8,5 2116536 917530 /usr/sbin/smardt
/usr/sbin 1554 root mem REG 8,5 2029592 3017082 /usr/lib/x86_64-linux-gnu/libc-2.31.so
/usr/sbin 1554 root mem REG 8,5 18848 3017084 /usr/lib/x86_64-linux-gnu/libdl-2.31.so
/usr/sbin 1554 root mem REG 8,5 191504 3015536 /usr/lib/x86_64-linux-gnu/ld-2.31.so
/usr/sbin 1554 root 0u CHR 1,3 0t0 6 /dev/null
/usr/sbin 1554 root 1u CHR 1,3 0t0 6 /dev/null
/usr/sbin 1554 root 2u CHR 1,3 0t0 6 /dev/null
/usr/sbin 1554 root 3u pack 28014 0t0 IP type=SOCK_DGRAM
```

File descriptors for the more recent BPFDoor compared to the prior reported

- No more **SOCK_RAW** appearing. You will only find socket of type **SOCK_UDP** in **BPFDoor** open file descriptors. Despite this, the implant still accepts **ICMP**, **UDP** and **TCP** wakeup packets
- The random selection of custom process names used for masquerading has been replaced with a fixed process name.
 - Mutex locks file paths are adjusted accordingly
- Removal of "fileless" feature, no longer writing to **/dev/shm** and deleting itself from disk
- Global variables / function names stripped - no more looking for binaries with **godpid**
- SSL for transport encryption with embedded certificate
- Updated BPF filter, now coming at a whopping 229 bytes long

Goodbye SOCK_RAW

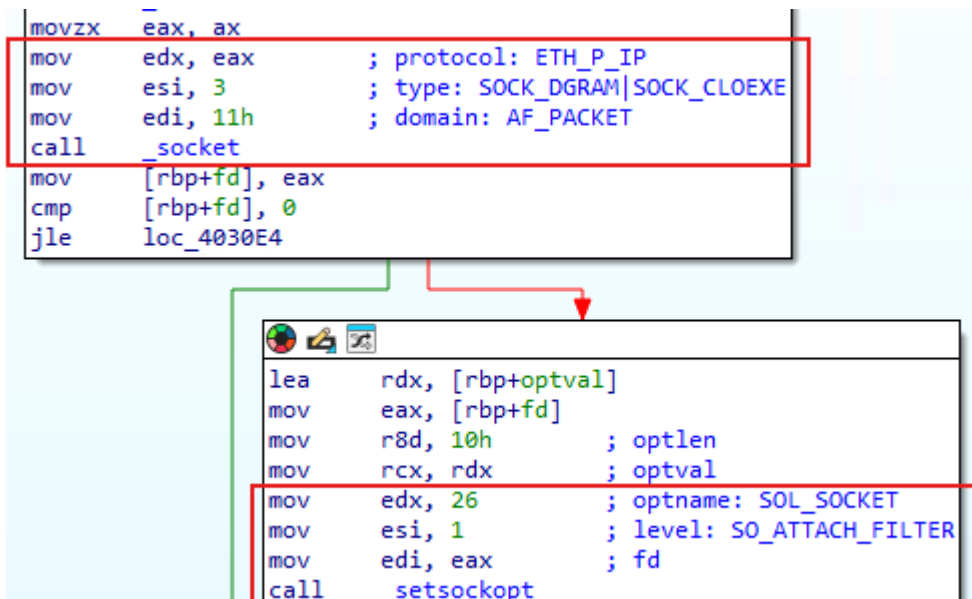
A common and useful detection opportunity is to look for unexpected processes with open raw sockets. Doing a `lsof | grep SOCK_RAW` won't surface `BPFDoor` anymore. This is how it looks now:

```
# lsof -p 4073
..
/usr/sbin 4073 root  0u  CHR 136,3    0t0    6 /dev/pts/3
/usr/sbin 4073 root  1u  CHR 136,3    0t0    6 /dev/pts/3
/usr/sbin 4073 root  2u  CHR 136,3    0t0    6 /dev/pts/3
/usr/sbin 4073 root  3u  pack 37048     0t0    IP type=SOCK_DGRAM
```

Prior, variants would open a raw socket (followed by `setsockopt` to install its `BPF` filter):

```
socket(PF_PACKET, SOCK_RAW, htons(ETH_P_IP))
..
setsockopt(sock, SOL_SOCKET, SO_ATTACH_FILTER, &filter, sizeof(filter))
```

Now we see that `SOCK_RAW` has been replaced with `SOCK_DGRAM|SOCK_CLOEXEC`



In other words we now have:

```
socket(AF_PACKET, SOCK_DGRAM|SOCK_CLOEXEC, htons(ETH_P_IP))
```

One would assume on a surface glance that `AF_PACKET` with `SOCK_DGRAM` would only receive `UDP` packets (`SOCK_DGRAM`), but this is not actually the case.

[packet\(7\)](#) states:

"When protocol is set to `htons(ETH_P_IP)`, then all IPv4 packets are received. All incoming packets of that protocol type will be passed to the packet socket before they are passed to the protocols

implemented in the kernel."

Hence, as per the description, `ETH_P_IP` will take precedence, making the socket 'applicable' for `IP` packets of any protocol (`TCP` , `UDP` , `ICMP`). What is not apparent is that kernel will reports the socket as `SOCK_DGRAM` which is misleading unless the nuance is understood.

The data received is slightly different as well: With `SOCK_RAW` the kernel will pass the whole packet, including the link layer, while `AF_PACKET` strips this out and will everything from the `IP` header onwards. Hence we see in the [original source](#) with `SOCK_RAW` discards the 14 byte ethernet header:

```

520             memset(buff, 0, 512);
521             psize = 0;
522             r_len = recvfrom(sock, buff, 512, 0x0, NULL, NULL);
523
524             ip = (struct sniff_ip *)(buff+14);
525             size_ip = IP_HL(ip)*4;
526             if (size ip < 20) continue;
```

While the newer version using `SOCK_DGRAM` assumes the first byte to be the start of the IP header:

```

58             memset(buff, 0, 1024uLL);
59             v30 = 0;
60             r_len = recvfrom(sock, buff, 1024uLL, 0, 0LL, 0LL);
61             v28 = buff;
62             size_ip = 4 * (buff[0] & 0xF);           // size_ip = IP_HL(ip)*4;
63         }
64         while ( size_ip <= 19 );
```

This very minor modification is a particularly improvement to it's stealth

Socket detection opportunities

`ss -0pb` will still surface the `BPF` filter.

```

# ss -0pb
..
users:(("/usr/sbin/smart",pid=4535,fd=3))
    bpf filter (229): 0x30 0 0 0, 0x54 0 0 240 ..
```

Removing the `b` flag (and trimming out `systemd` related false positives), the socket is listed as `p_dgr` :

```

# ss -0p
p_dgr .... users:(("/usr/sbin/smart",pid=4812,fd=3))
```

If monitoring system calls, there are some opportunities. For example, `auditd` rules for `socket` with `AF_PACKET` . Expect false positives.

```
auditctl -a exit,always -F arch=b64 -S socket -F a0=17
```

and `setsockopt` with `SO_ATTACH_FILTER` :

```
auditctl -a exit,always -F arch=b64 -S setsockopt -F a2=26
```

No more deleted in /dev/shm

As an evasion technique, BPFDoor would copy itself into `/dev/shm` , execute itself then delete itself from disk as an anti-forensic technique.

The kernel marking a processes' executable `inode` as `(deleted)` within a temporary file system such as `/dev/shm` proclaims loudly "this deserves a closer look".

Now `BPFDoor` no longer touches `/dev/shm` nor does it deletes itself. It behaves more like a normal process with the exception of its process name masquerading with the `prctl` system call and overwriting the `envp` on the stack.

Process Names and Mutex Lock

Prior, the malware would randomly select a process name from a fixed list of common process names and a somewhat constant file path for its [mutex lock](#). Both of these have changed - The masqueraded process name and mutex lock path may vary to better match the target environment. For reference, the [original](#) would randomly select from the following list of process names/arguments on initialization:

```
char *self[] = {
    "/sbin/udev -d",
    "/sbin/mingetty /dev/tty7",
    "/usr/sbin/console-kit-daemon --no-daemon",
    "hald-addon-acpi: listening on acpi kernel interface /proc/acpi/event",
    "dbus-daemon --system",
    "hald-runner",
    "pickup -l -t fifo -u",
    "avahi-daemon: chroot helper",
    "/sbin/auditd -n",
    "/usr/lib/systemd/systemd-journald"
};
```

A single process name is now hard coded. For example, the following example disguises itself as `smartd` , a disk monitoring daemon:

```

1 | int64 __fastcall main(int argc, char **argv, char **envp)
2 | {
3 |     // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
4 |
5 |     strcpy(pass, "73b9989bb8dd522b8e172f2e985810eb");
6 |     strcpy(pass2, "d46bf5d43cffd7793665d40fc767ed86");
7 |     src = "/usr/sbin/smardt -n -q never";
8 |     file = "/var/run/hald-smardt.pid";
9 |     if ( !access("/var/run/hald-smardt.pid", 4) )
10 |         exit(0);

```

Other new "single process" masqueraded process names include:

- `lldpad -d , /var/run/lldpad.lock`
- `dbus-daemon --system , /var/run/system.pid`
- `/usr/libexec/hald-addon-volume , /var/run/hald-addon.pid`
- `/usr/sbin/console-kit , /var/run/console-kit.pid`

Notably, when a magic packet is received, the forked process name remains `/usr/libexec/postfix/master` and the familiar `qmgr` .



As with all BPFDoor samples, running 'strings' or similar will miss many strings. For example, in the disassembly, the `md5` password hashes above are littered with `mov` instructions. This is due to the use of the strings being defined as an array, or as a "stack string".

<pre> mov [rbp+src], 2Fh ; '/' mov [rbp+var_C0F], 75h 'u' mov [rbp+var_C0E], 73h 's' mov [rbp+var_C0D], 72h 'r' mov [rbp+var_C0C], 2Fh '/' mov [rbp+var_C0B], 6Ch 'l' mov [rbp+var_C0A], 69h 'i' mov [rbp+var_C09], 62h 'b' mov [rbp+var_C08], 65h 'e' </pre>	<pre> • 114 strcpy(src, "/usr/libexec/postfix/master"); • 115 if (fork()) • 116 exit(0); • 117 setuid(); • 118 signal(1, 0LL); • 119 v4 = strlen(::dest); • 120 memset(::dest, 0, v4); • 121 strcpy(::dest, src); • 122 prctl(15, src); </pre>
---	--

`bpfdoor-dump.py` can be used to extract obfuscated hashes and other strings from samples. Alternatively, check out [this](#) from elastic.co.

SSL Encryption

Vanilla `RC4` has been deprecated in the `getshell` function, replaced `SSL` using `RC4-MD5` for both `bind` mode (listening server) or the reverse connection mode (client).

```

106     v32 = BIO_new_mem_buf(certStr, -1);
107     v36 = sub_48FCD0(v32, 0LL, 0LL, 0LL);
108     SSL_CTX_use_certificate((__int64)sslCtx, v36);
109     v31 = BIO_new_mem_buf(privkeyStr, -1);
110     bio_RSAPrivateKey = PEM_read_bio_RSAPrivateKey((__int64)v31, 0LL, 0LL, 0LL);
111     SSL_CTX_use_RSAPrivateKey((__int64)sslCtx, bio_RSAPrivateKey);
112     if ( !(unsigned int)SSL_CTX_check_private_key((__int64)sslCtx) )
113     {
114         fwrite("Private key does not match the public certificate\n", 1uLL, 0x32uLL, stderr);
115         sub_4710D0(stderr);
116         return 0LL;
117     }
118     SSL_CTX_set_cipher_list((__int64)sslCtx, "RC4-MD5");
119     ssl = (void *)SSL_new((__int64 *)sslCtx);
120     SSL_set_fd(ssl, sock);
121     if ( SSL_accept((SSL *)ssl) <= 0 )
122     {
123         sub_4710D0(stderr);
124         return 0LL;
125     }
126 }
127 else // reverse TCP mode
128 {
129     SSL_library_init();
130     sub_414FA0();
131     SSL_library_init();
132     v37 = (void *)sub_405A80();
133     sslCtx = (void *)SSL_CTX_new((__int64)v37);
134     if ( !sslCtx )
135         return 0LL;
136     SSL_CTX_set_cipher_list((__int64)sslCtx, "RC4-MD5");
137     ssl = (void *)SSL_new((__int64 *)sslCtx);
138     SSL_set_fd(ssl, sock);
139     if ( SSL_connect((SSL *)ssl) <= 0 )
140         return 0LL;
141 }

```

function names, vars have been renamed during decomp.

The hardcoded certificate and private key (used in `bind / server mode`, with `SSL_accept` is extractable with `strings` . In all samples analyzed, 3 unique self-signed certificates were identified.

```

-----BEGIN CERTIFICATE-----
MIIB+zCCAWQCQCtA0agZ+qQ5jANBgkqhkiG9w0BAQsFAADBCMQswCQYDVQQGEwJY
WDEVMBMGA1UEBwwMRGVMYXVsdCBDaXR5MRwwGgYDVQQKBNEZWhdWx0IENvbXBh
bnkgTHRkMB4XD0tIxmTEyMzAyMTc0N0LoXDTMxMTEyMTA5MTc0N0LowQjELMAKGA1UE
BhMCWFgxFATBgNVBACMDERLZmF1bHh0eTEcMBoGA1UECgwTRGVmYXVsdCBDbD
b21wYW55IEEx0ZDZCbnANBgkqhkiG9w0BAQEFAAOBjQAwYkCgYEAx1Jv0+nqr24g
wc8at6x1NtZt7DoDi1/Ge/F70zz4gbxX/Oxh0xXKexYrphsHXBzYVEW0yof9Vnok
ST7GKdRiRg60S90WfdWfOVN2EdxwBN+BdozmrRBG1DAdqAhbeUcUFeZ00Fbuo7fr
FvTfsC31khj6ioKJl0d4kfo2zLk6WhcCAwEAATANBgkqhkiG9w0BAQsFAA0BgQA1
iC/5g+eN3Hq/627tMbLhipNUTC00Edtpq20mbUIMXTRYh4kZPAah1LZqx2h72BV1
i8pYJo34kZ/3HyV6UJtBf/jJv1fprEWvo2Lj8YrCpagXh82i7353GueiKfVr0gx+
4ruTus1m0bX1NZN6XRAbgzar7bfki0HHjWxJB8NRLQ==
-----END CERTIFICATE-----

```

```

openssl x509 --in cert.pem -noout -dates -fingerprint
notBefore=Nov 23 02:17:46 2021 GMT
notAfter=Nov 21 02:17:46 2031 GMT
SHA1 Fingerprint=85:CA:7E:BB:F1:1F:53:45:4E:DA:BB:27:DD:DC:59:DB:52:C2:0E:08

```

This change may explain one reason the newer samples are statically compiled, resulting in a much larger file executable file size - with the benefit of avoiding any linking issues with the SSL library.

Password hashes

Embedded passwords in older samples have been replaced with salted hashes. Where hashing was used, the same constant salt is present: I5*AYbs@LdaWbs0

```
1  _int64 __fastcall auth(const char *incoming_pass)
2  {
3      // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
4
5      result = 0;
6      memset(salt_pass, 0, sizeof(salt_pass));
7      v11 = 0;
8      *salt = 0LL;
9      v6 = 0LL;
10     v7 = 0LL;
11     v8 = 0LL;
12     v9 = 0LL;
13     strcpy(_salt, "I5*AYbs@LdaWbs0");
14     strcpy(salt, _salt);
15     if ( incoming_pass )
16         strcat(salt, incoming_pass, 14uLL);
17     sub_4033BE(salt, salt_pass);
18     v1 = strlen(revshell_pass);
19     result = memcmp(revshell_pass, salt_pass, v1);
20     if ( !result )
21         return 0LL;
22     v3 = strlen(bind_pass);
23     result = memcmp(bind_pass, salt_pass, v3);
24     if ( result )
25         return 2LL;
26     else
27         return 1LL;
28 }
```

The first character of the password is used as an instruction on which mode to invoke. Passwords must start with either a `m`, `s` or `j`. This means the the search space for cracking the password hashes is reduced by one character which can make a huge difference in the time to brute force the key space, reducing the maximum length from 14 to 13 characters.

```

129     switch ( SBYTE2(password) )
130     {
131         case 'm':
132             puts("[+] Monitor packet send.");
133             direct = 2;
134             break;
135         case 's':
136             if ( raw == 2 || raw == 3 )
137             {
138                 puts("[-] Mode error.");
139                 return -1;
140             }
141             if ( raw == 1 && inaddr == -1 )
142             {
143                 puts("[-] missing -1 ");
144                 return -1;
145             }
146             puts("[+] Direct connection mode");
147             direct = 1;
148             break;
149         case 'j':
150             puts("[+] Reserve connection mode");
151             if ( !v25 )
152             {

```

6227cb77cb4ab1d066eebf14e825dbc0a0a7f1e9

We can crack these hashes with [hashcat](#), using mode `20` - `md5($salt.$pass)`

For example:

```
hashcat -a 3 -m 20 hash:I5*AYbs@LdaWbs0 j?1?1?1?1?1?1?1?1?1 -1 ?l?u -i
```

Here are some hashes taken from samples with their corresponding plain-text password. *Cracked hashes that include acronyms which can identify the victim organisation have been omitted.*

```

aa73d4574fd91b9648d73b01ea1920f3:I5*AYbs@LdaWbs0:joinfare
5609e5e3d3e7efd85e219901ab06bb61:I5*AYbs@LdaWbs0:jberemote
215c5b9279d3e462eceb9af3b5028c05:I5*AYbs@LdaWbs0:justgetso
629849fe5277500a777087d78ddc5dde:I5*AYbs@LdaWbs0:jusrbackso
5fb2ce4f90c53071b12e65d52445d33d:I5*AYbs@LdaWbs0:javatelnet
73b9989bb8dd522b8e172f2e985810eb:I5*AYbs@LdaWbs0:justgetdata
05b37b412e1d1bfdc6b8643d3c869b01:I5*AYbs@LdaWbs0:justgetcheck
8528eba01dca94e6b0d7c4c8cc39889f:I5*AYbs@LdaWbs0:justgotowork
4cf71dacf1750e2a9f122fba74b86a5d:I5*AYbs@LdaWbs0:senttome
3de78247e0e1c9ca3c291bc060d9b622:I5*AYbs@LdaWbs0:setdefault
d46bf5d43cffd7793665d40fc767ed86:I5*AYbs@LdaWbs0:sentandconn
3d45acc78e9d6de380b3cbdccf38af0a:I5*AYbs@LdaWbs0:setopenview

```

[bpfdoor-dump.py](#) can be used to extract the hashes and generate the respective hashcat commands.

YARA Rules

Testing against both Elastic.co's set of BPFDoor [yara rules](#) from 2022 does identify the newer statically compiled, stripped samples due to the rule `Linux_Trojan_BPFDoor_5` including:

```
{ D0 48 89 45 F8 48 8B 45 F8 0F B6 40 0C C0 E8 04 0F B6 C0 C1 }
```

This corresponds to the instructions in the `packet_loop` function which calculates the offset of the TCP header for extracting the magic bytes in the TCP wakeup packets. But the ruleset does miss quite a few samples. For maximum coverage, use with Florian Roth's ruleset.

[signature-base/yara/mal_lnx_implant_may22.yar at 391a990859091dbc4c21d15db335b371090f606e · Neo23x0/signature-base](#)

[YARA signature and IOC database for my scanners and tools - Neo23x0/signature-base](#)



[GitHubNeo23x0](#)

Neo23x0/ signature-base



YARA signature and IOC database for my scanners and tools

48

Contributors

5

Issues

3k

Stars

631

Forks



For additional coverage on the latest variants, including the early `NotBPDDoor`, the following `YARA` rules can be used:

```
rule bpfdoor_cert_variant
{
  meta:
    description = "Detects BPFDoor SSL versions"
    reference = ""
    date = "2025-04-31"
    hash1 = "3f6f108db37d18519f47c5e4182e5e33cc795564f286ae770aa03372133d15c4"
    hash2 = "724bd9163641666e035cef81701856fc9ff2dada2509d55dec14588fd1b5e801"
    hash3 = "7804f1dfb5d80a80830829c06ae65b410073748038f965f688dbd84d02eb0008"
    hash4 = "28bfb3f2067c77b83898ef4e41c9fc573e6aaa8581da9b59bddb782205a0b091"
    hash5 = "29564c19a15b06dd5be2a73d7543288f5b4e9e6668bbd5e48d3093fb6ddf1fdb"
    author = "@haxrob"
  strings:
    $s1 = "ttcompat" fullword ascii
    $s2 = "Private key does not match the public certificate" fullword ascii
    $s3 = "HISTFILE=/dev/null" fullword ascii
  condition:
    uint16(0) == 0x457f and (all of them)
}

rule notbpfdoor
{
  meta:
    description = "Detects early (2015/2016) variant"
    reference = ""
    date = "2025-04-31"
    hash1 = "ebffd115918f6d181da6d8f5592dfffb3e4f08cd4e93dcf7b7f1a2397af0580d9"
    hash2 = "b2d3c212e71ddbaf015d8793d30317e764131c9beda7971901620d90e6887b30"
```

```
author = "@haxrob"  
strings:  
  $s1 = "ttcompat" fullword ascii  
  $s2 = "auto install failed, plz manual install it!" fullword ascii  
  $s3 = "unset LC_TIME" fullword ascii  
condition:  
  uint16(0) == 0x457f and (all of them)  
}
```

Source: <https://haxrob.net/bpfdoor-past-and-present-part-2/>