Netlab 360.com

Lightweight emulation based IoC extraction for Gafgyt botnets

Ya Liu

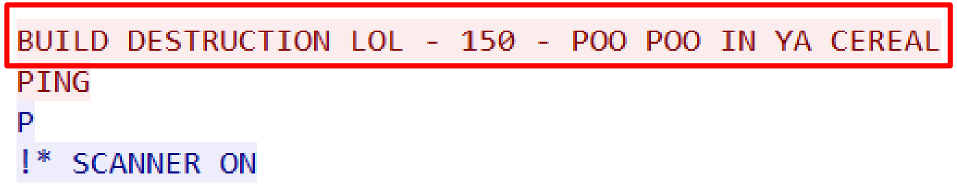
VB2020 localhost

Outline

- Background
- Gafgyt C2 loop
- Emulating initConnection() to get C2
- Analyzing Mirai code in Gafgyt

About Gafgyt

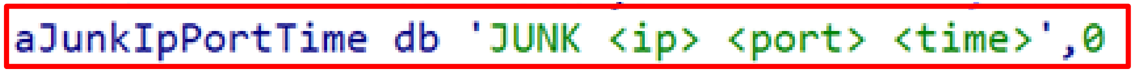
- firstly emerged in 2014
- also known as Qbot, BASHLITE, LizardStresser
- an earlier IoT botnets than Mirai
- A IRC like C2 protocol is used



```
BUILD DESTRUCTION LOL - 150 - POO POO IN YA CEREAL
PING
P
!* SCANNER ON
```

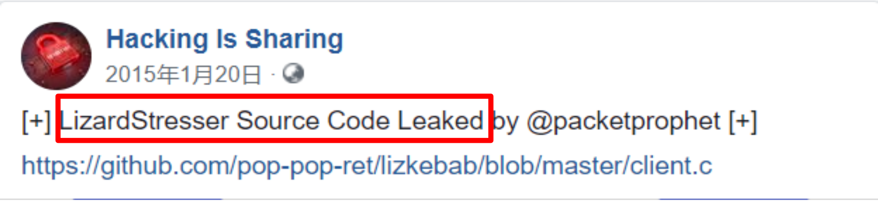
register

command



```
aJunkIpPortTime db 'JUNK <ip> <port> <time>',0
```

Why so many Gafgyt variants?

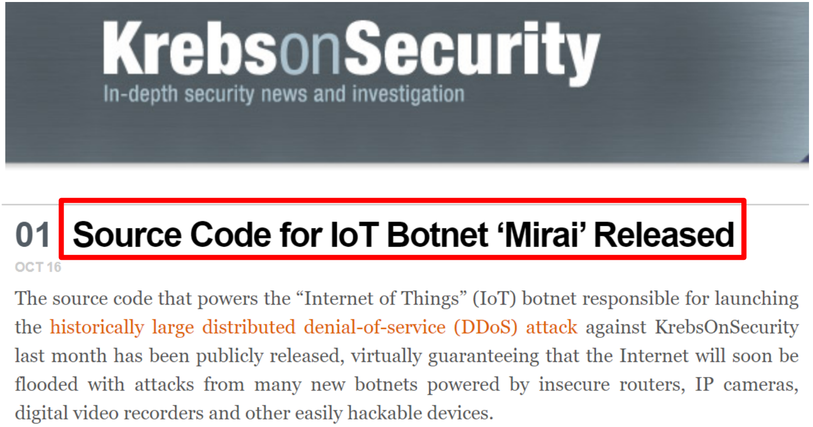


Hacking Is Sharing

2015年1月20日 ·

[+] LizardStresser Source Code Leaked by @packetprophet [+]

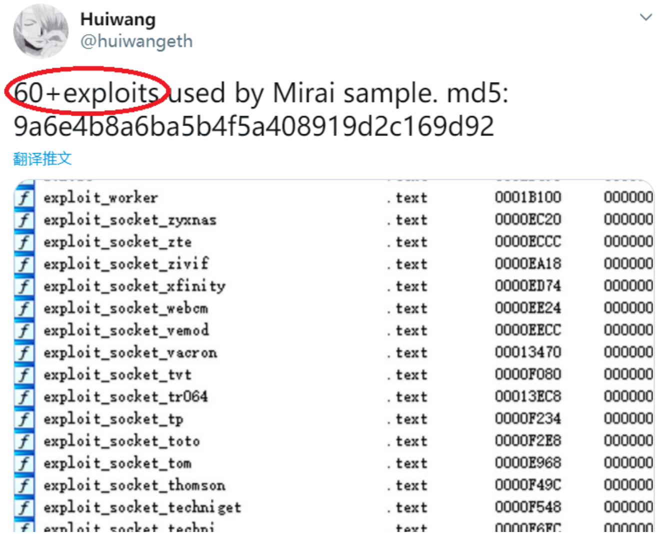
https://github.com/pop-pop-ret/lizkebab/blob/master/client.c



01 Source Code for IoT Botnet 'Mirai' Released

OCT 16

The source code that powers the "Internet of Things" (IoT) botnet responsible for launching the historically large distributed denial-of-service (DDoS) attack against KrebsOnSecurity last month has been publicly released, virtually guaranteeing that the Internet will soon be flooded with attacks from many new botnets powered by insecure routers, IP cameras, digital video recorders and other easily hackable devices.



Huiwang
@huiwangeth

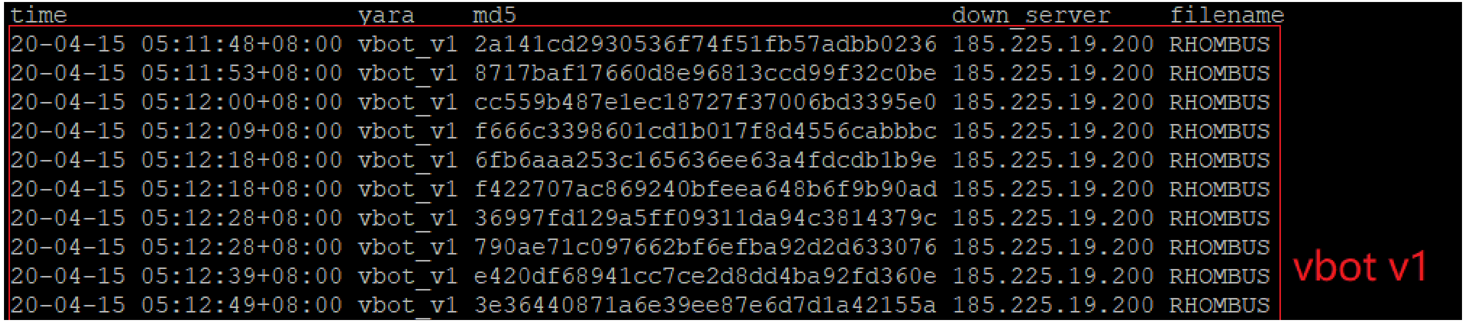
60+exploits used by Mirai sample. md5: 9a6e4b8a6ba5b4f5a408919d2c169d92

翻译推文

```
exploit_worker              .text    0001B100    000000
exploit_socket_zyxnas       .text    0000EC20    000000
exploit_socket_zte          .text    0000ECCC    000000
exploit_socket_zivif        .text    0000EA18    000000
exploit_socket_xfinity      .text    0000ED74    000000
exploit_socket_webcm        .text    0000EE24    000000
exploit_socket_vemod        .text    0000EECC    000000
exploit_socket_vacron       .text    00013470    000000
exploit_socket_tvt          .text    0000F080    000000
exploit_socket_tr064        .text    00013EC8    000000
exploit_socket_tp           .text    0000F234    000000
exploit_socket_toto         .text    0000F2E8    000000
exploit_socket_tom          .text    0000E968    000000
exploit_socket_thomson      .text    0000F49C    000000
exploit_socket_techniget    .text    0000F548    000000
exploit_socket_techni       .text    0000F6FC    000000
```

Developing a new Gafgyt variant is just a process of "Ctrl+c" and "Ctrl+v".

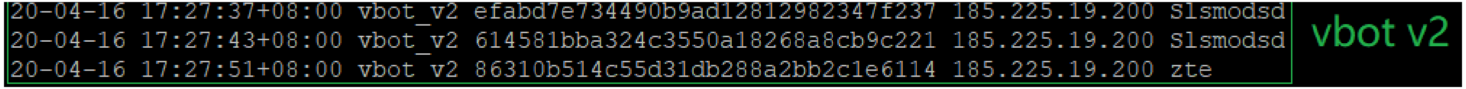
Fast emerging while short living



```
time                      yara     md5                              down_server    filename
20-04-15 05:11:48+08:00 vbot_v1 2a141cd2930536f74f51fb57adbb0236 185.225.19.200 RHOMBUS
20-04-15 05:11:53+08:00 vbot_v1 8717baf17660d8e96813ccd99f32c0be 185.225.19.200 RHOMBUS
20-04-15 05:12:00+08:00 vbot_v1 cc559b487e1ec18727f37006bd3395e0 185.225.19.200 RHOMBUS
20-04-15 05:12:09+08:00 vbot_v1 f666c3398601cd1b017f8d4556cabbbc 185.225.19.200 RHOMBUS
20-04-15 05:12:18+08:00 vbot_v1 6fb6aaa253c165636ee63a4fdcdb1b9e 185.225.19.200 RHOMBUS
20-04-15 05:12:18+08:00 vbot_v1 f422707ac869240bfeea648b6f9b90ad 185.225.19.200 RHOMBUS
20-04-15 05:12:28+08:00 vbot_v1 36997fd129a5ff09311da94c3814379c 185.225.19.200 RHOMBUS
20-04-15 05:12:28+08:00 vbot_v1 790ae71c097662bf6efba92d2d633076 185.225.19.200 RHOMBUS
20-04-15 05:12:39+08:00 vbot_v1 e420df68941cc7ce2d8dd4ba92fd360e 185.225.19.200 RHOMBUS
20-04-15 05:12:49+08:00 vbot_v1 3e36440871a6e39ee87e6d7d1a42155a 185.225.19.200 RHOMBUS
```

vbot v1

- It kept active from mid-April to mid-June
- 2 versions have been found
- **31** campaigns were detected, with **572** samples captured from **12** download servers
- **13** C2 servers were found



```
20-04-16 17:27:37+08:00 vbot_v2 efabd7e734490b9ad12812982347f237 185.225.19.200 Slsmodsd
20-04-16 17:27:43+08:00 vbot_v2 614581bba324c3550a18268a8cb9c221 185.225.19.200 Slsmodsd
20-04-16 17:27:51+08:00 vbot v2 86310b514c55d31db288a2bb2c1e6114 185.225.19.200 zte
```

vbot v2

IoC extraction

- Quick IoC extraction would play an important role in fighting Gafgyt like fast emerging while short living threats
- Current solutions include sandbox based and static analysis based
- Issues of sandbox based IoC extraction:
  - deploying sandboxes of multiple CPU architectures
  - needing to know fixed patterns of C2 messages in advance
  - potentially impacting other systems due to scans initialized by samples
- Static analysis based solution has the issue of signature explosion

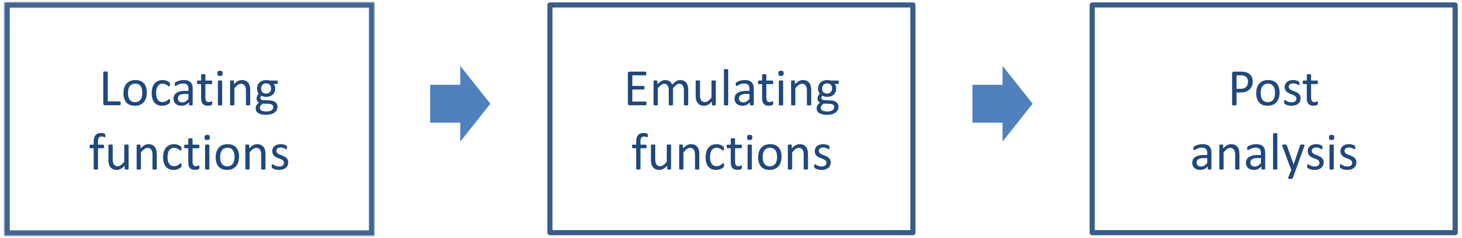
About lightweight emulation

- LWE: Lightweight Emulation

	Sandbox	LWE
Is dynamic analysis?	yes	yes
Targets	PE、ELF、DOC、…	code snippet
Instruction-level instrumentation	Not necessary	MUST
Are syscalls provided?	MUST	No, or partially provided
Time	a few minutes	a few seconds
Output	behavior reports, PCAPs	logs of executed instructions, CPU registers and memory snapshots

2 prerequisites for LWE based extractions

1. Fixed behavior patterns can be concluded from interested code
   - C2 communication code
2. The target code can be located in an automatic manner
   - Such locating must be independent of static patterns
   - CFG patterns are recommended

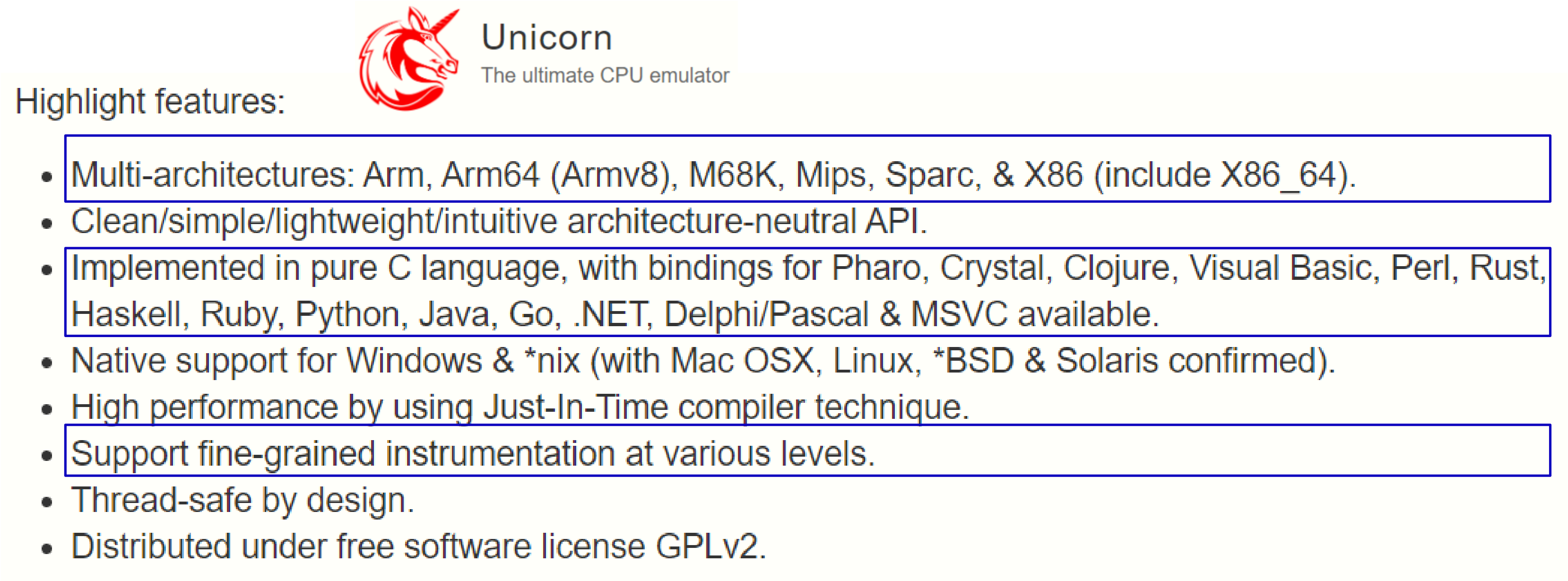


Locating functions

Emulating functions

Post analysis

LWE engine



Unicorn

The ultimate CPU emulator

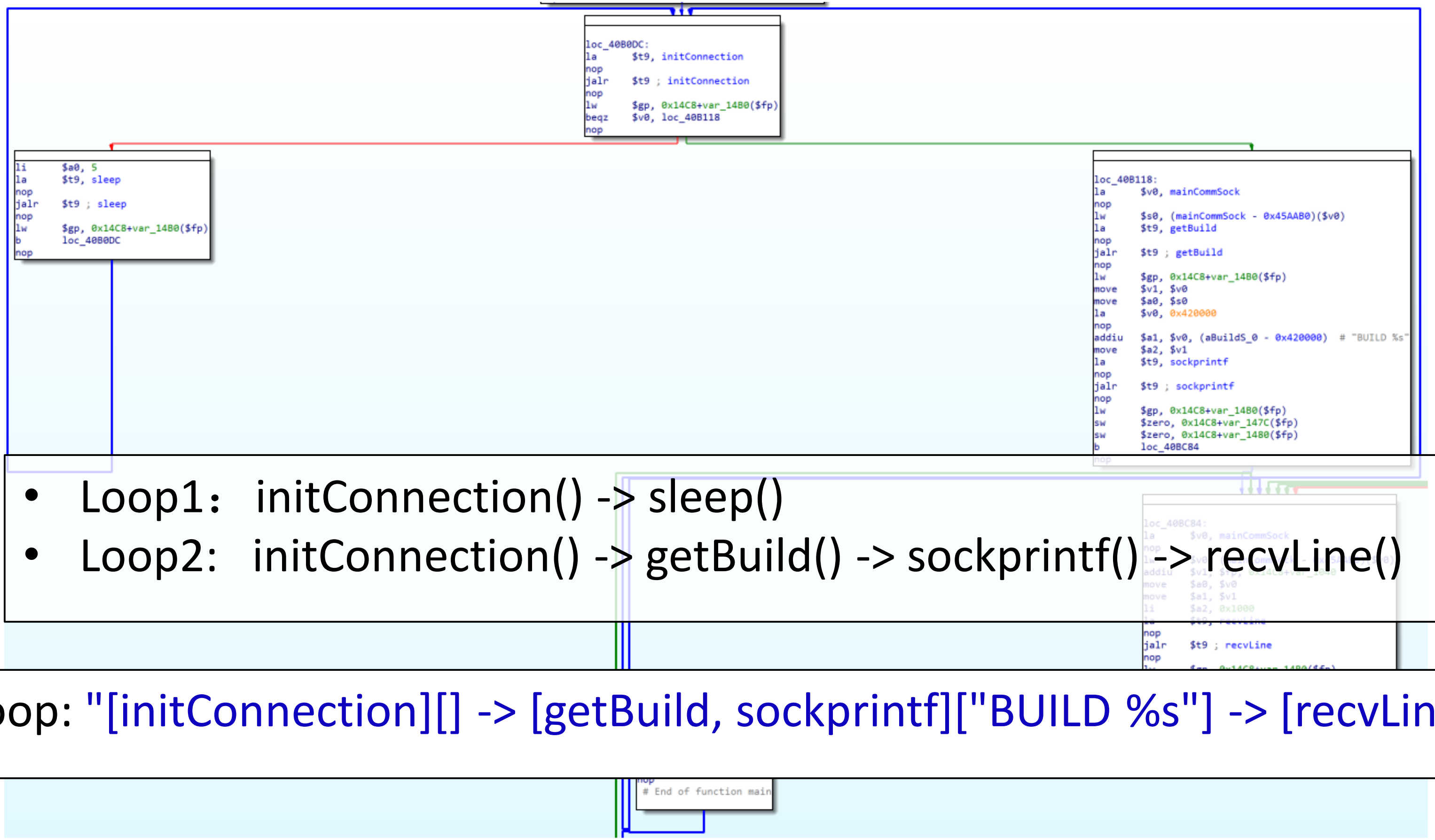
Highlight features:

- Multi-architectures: Arm, Arm64 (Armv8), M68K, Mips, Sparc, & X86 (include X86_64).
- Clean/simple/lightweight/intuitive architecture-neutral API.
- Implemented in pure C language, with bindings for Pharo, Crystal, Clojure, Visual Basic, Perl, Rust, Haskell, Ruby, Python, Java, Go, .NET, Delphi/Pascal & MSVC available.
- Native support for Windows & *nix (with Mac OSX, Linux, *BSD & Solaris confirmed).
- High performance by using Just-In-Time compiler technique.
- Support fine-grained instrumentation at various levels.
- Thread-safe by design.
- Distributed under free software license GPLv2.

https://www.unicorn-engine.org/

Gafgyt C2 loop

C2 loop in main()



```
loc_40B0DC:
la      $t9, initConnection
nop
jalr    $t9 ; initConnection
nop
lw      $gp, 0x14C8+var_14B0($fp)
beqz    $v0, loc_40B118
nop
```

```
li      $a0, 5
la      $t9, sleep
nop
jalr    $t9 ; sleep
nop
lw      $gp, 0x14C8+var_14B0($fp)
b       loc_40B0DC
nop
```

```
loc_40B118:
la      $v0, mainCommSock
nop
lw      $s0, (mainCommSock - 0x45AAB0)($v0)
la      $t9, getBuild
nop
jalr    $t9 ; getBuild
nop
lw      $gp, 0x14C8+var_14B0($fp)
move    $v1, $v0
move    $a0, $s0
la      $v0, 0x420000
nop
addiu   $a1, $v0, (aBuildS_0 - 0x420000)  # "BUILD %s"
move    $a2, $v1
la      $t9, sockprintf
nop
jalr    $t9 ; sockprintf
nop
lw      $gp, 0x14C8+var_14B0($fp)
sw      $zero, 0x14C8+var_147C($fp)
sw      $zero, 0x14C8+var_1480($fp)
b       loc_40BC84
nop
```

```
loc_40BC84:
la      $v0, mainCommSock
nop
lw      $v0, (mainCommSock - 0x45AAB0)($v0)
addiu   $v1, $fp, 0x14C8+var_1040
move    $a0, $v0
move    $a1, $v1
li      $a2, 0x1000
la      $t9, recvLine
nop
jalr    $t9 ; recvLine
nop
```

```
nop
 # End of function main
```

- Loop1： initConnection() -> sleep()
- Loop2: initConnection() -> getBuild() -> sockprintf() -> recvLine()

C2 loop: "[initConnection][] -> [getBuild, sockprintf]["BUILD %s"] -> [recvLine][] -> [][]"

A summary of C2 loop

- It’ s characteristic enough to be used to distinguish Gafgyt from other families, e.g., Mirai
- With C2 loop, we can:
  - directly get the register message template string
  - find the **initConnection()** function for further emulation to get C2
    - This function is responsible for establishing C2 connection
- It can be found by traversing control flow graph (CFG) of the main() function with IDA Python or radare2
  - graph algorithms, e.g., depth-first-search, are used

C2 loops vs variants

- C2 loops also vary across variants

```
"[initConnection][] --> [jprintf]["arch %s", "unknown"] --> [recvLine][] --> [][] "
"[initConnection][] --> [][] --> [recvLine][] --> [][] "
"[ec hoconnection][] --> [][] --> [recvLine][] --> [][] "
"[initConnection][] --> [sprintf, sockprintf]["fftt:%s"] --> [recvLine][] --> [][] “
"[Connection, botkiller, recv_buf][] "
```

- Common C2 loops can be summarized into 6 types according to their CFG patterns
  - block number
  - called functions
  - referenced strings

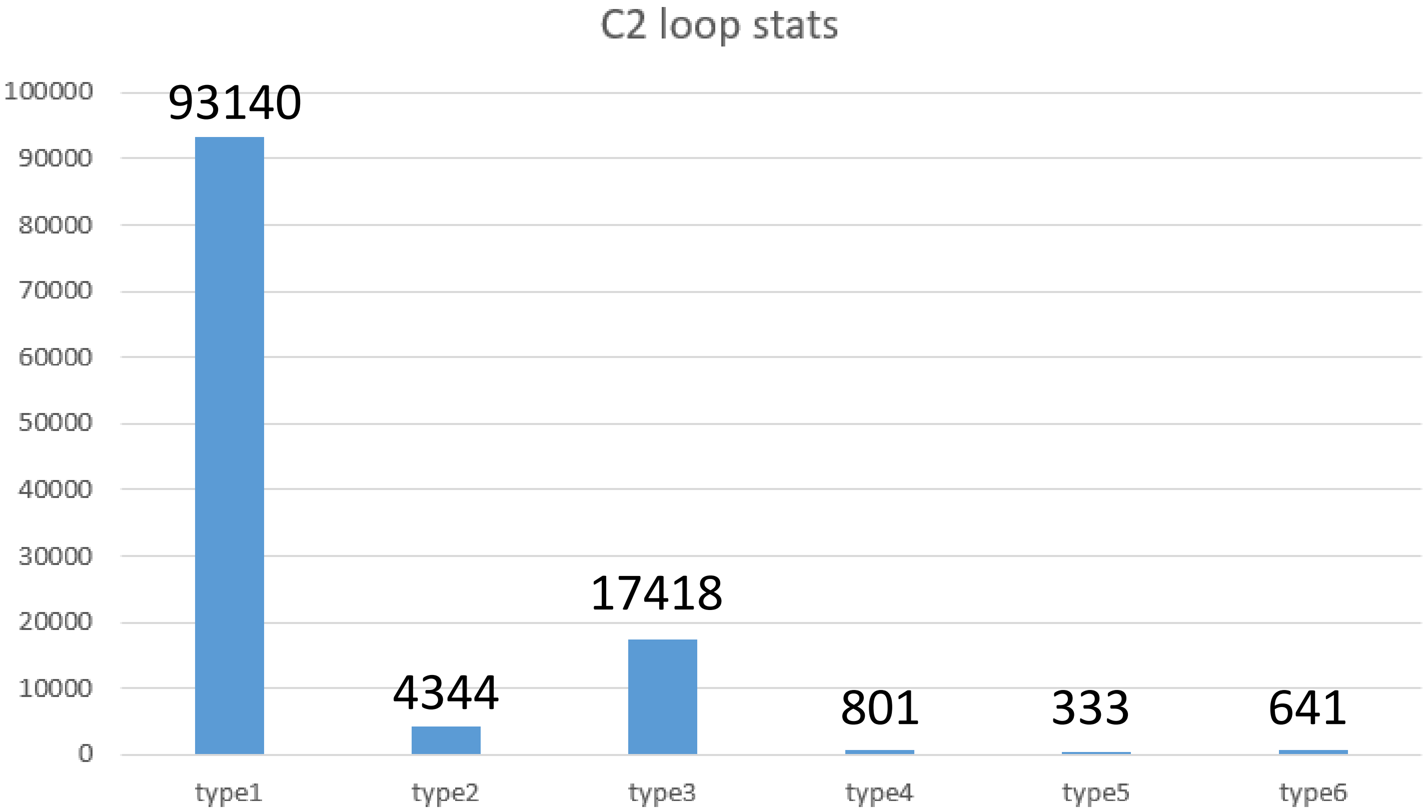
Examples of type 1~3

```
# type 1
"[initConnection][] -> [sockprintf]["3", "BUILD %s"] -> [recvLine][] -> [][]"
"[initConnection][] -> [getBuild, sockprintf][" 0i&", "BUILD %s"] -> [recvLine][]"

# type 2
"[echoconnection][] -> [][] -> [recvLine][] -> [puts]["UPDATING", "ECHOBOT"]"
"[echoconnection][] -> [][] -> [recvLine][] -> [][]"

# type 3
"[recvLine][] -> [initConnection][] -> [sockprintf]["BUILD %s", "DONGS"]"
"[viron][] -> [initConnection][] -> [sleep][]"
```

C2 loop stats on 116,677 samples



About register message template

- It’ s used to generate the register message with sprintf()
- Hundreds of template strings have been found

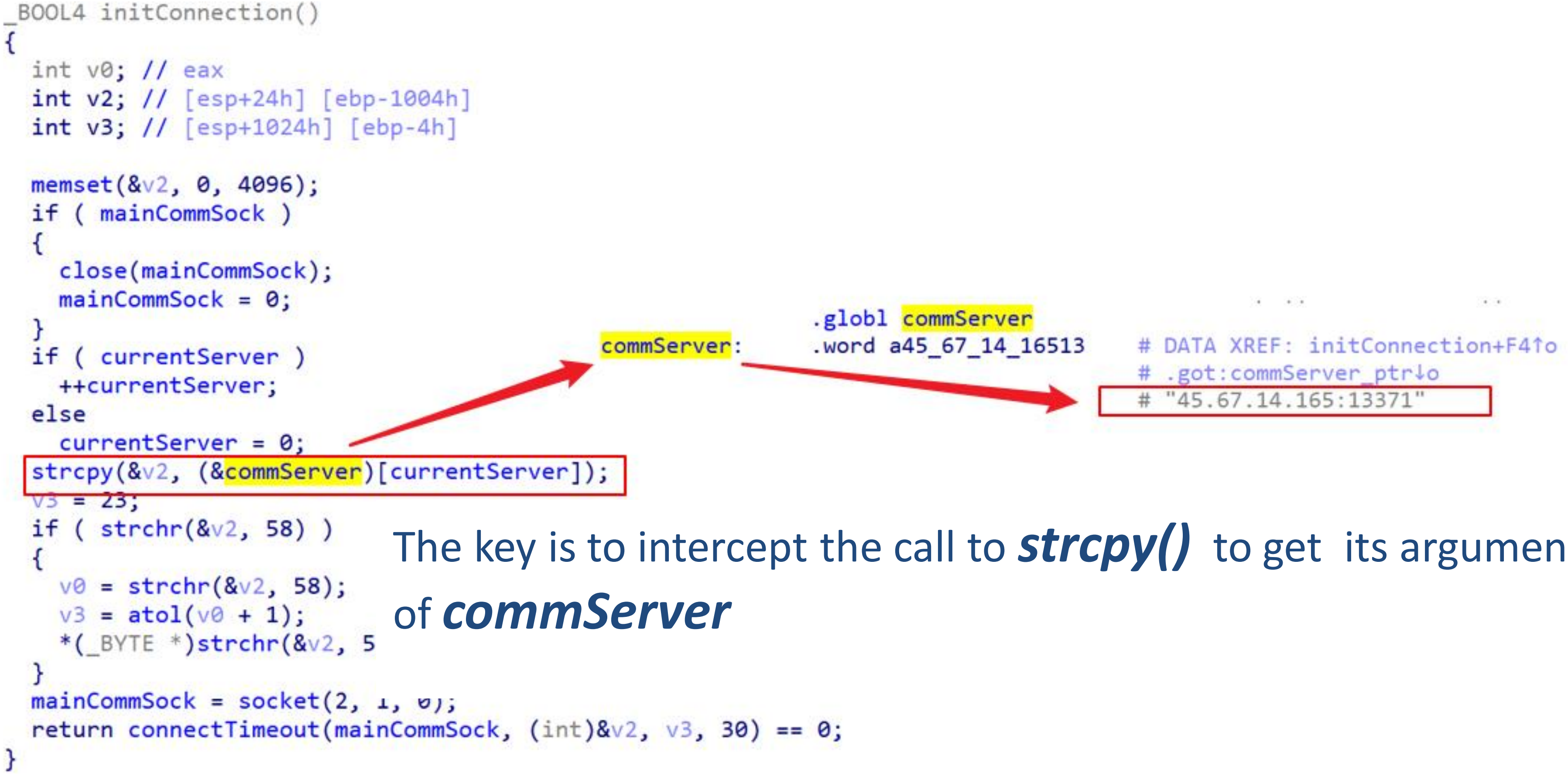
```
BUILD ART OF WAR
BUILD BLACKCULT %s
BUILD BOT : %s : %s
BUILD HERAV1 %s
BUILD Pussy Destroyer v911
BUILD [%s:%s:%d]
BUILD [[35m%s[37m] [[31m%s[37m]
```

```
[ %s CONNECTED ] [ %s: ] [ Arch Type: %s ]
[ BOT ] [ KETASHI ] ---> Bot Joined
[ SUPREME ]-->[%s]-->[%s]-->[%s]-->[%s]-->[%s]
[!] KATURA [!]  ~> [%s] ~> [%s] ~> [%s] ~> [%s] ~> [%s] ~> [%s]
[!] device connected [%s:%s:%s]
[+] Bot Connected - %s - Architecture %s
[+] Joined Hacker!: %s
```

- They are useful to detect Gafgyt C2 communication from network traffic
- Therefore template string is one of the extraction goals

Lightweight emulating initConnection()

Extracting C2 from initConnection()



```
_BOOL4 initConnection()
{
  int v0; // eax
  int v2; // [esp+24h] [ebp-1004h]
  int v3; // [esp+1024h] [ebp-4h]

  memset(&v2, 0, 4096);
  if ( mainCommSock )
  {
    close(mainCommSock);
    mainCommSock = 0;
  }
  if ( currentServer )
    ++currentServer;
  else
    currentServer = 0;
  strcpy(&v2, (&commServer)[currentServer]);
  v3 = 23;
  if ( strchr(&v2, 58) )
  {
    v0 = strchr(&v2, 58);
    v3 = atol(v0 + 1);
    *(_BYTE *)strchr(&v2, 5
  }
  mainCommSock = socket(2, 1, 0);
  return connectTimeout(mainCommSock, (int)&v2, v3, 30) == 0;
}
```

```
                .globl commServer
commServer:     .word a45_67_14_16513    # DATA XREF: initConnection+F4↑o
                                         # .got:commServer_ptr↓o
                                         # "45.67.14.165:13371"
```

The key is to intercept the call to ***strcpy()*** to get its argument of ***commServer***

Emulation steps

1. Initialization
   - Setting CPU arch and initializing registers
   - Mapping ELF file including code and data
   - Installing hooks of UC_HOOK_CODE and UC_HOOK_MEM_WRITE
2. Emulating initConnection() from its starting address
3. Post analysis
   - Parsing logged events: call and memory write
   - Reading C2 from global memory with parsed address

Instruction level instrumentation

- It is done through unicorn hook of UC_HOOK_CODE
- When encountering a call instruction, it will:
  - log the PC together with its arguments for post analysis
  - set return value, e.g., EAX in x86 CPU, to zero or a valid memory addr
  - skip to next instruction

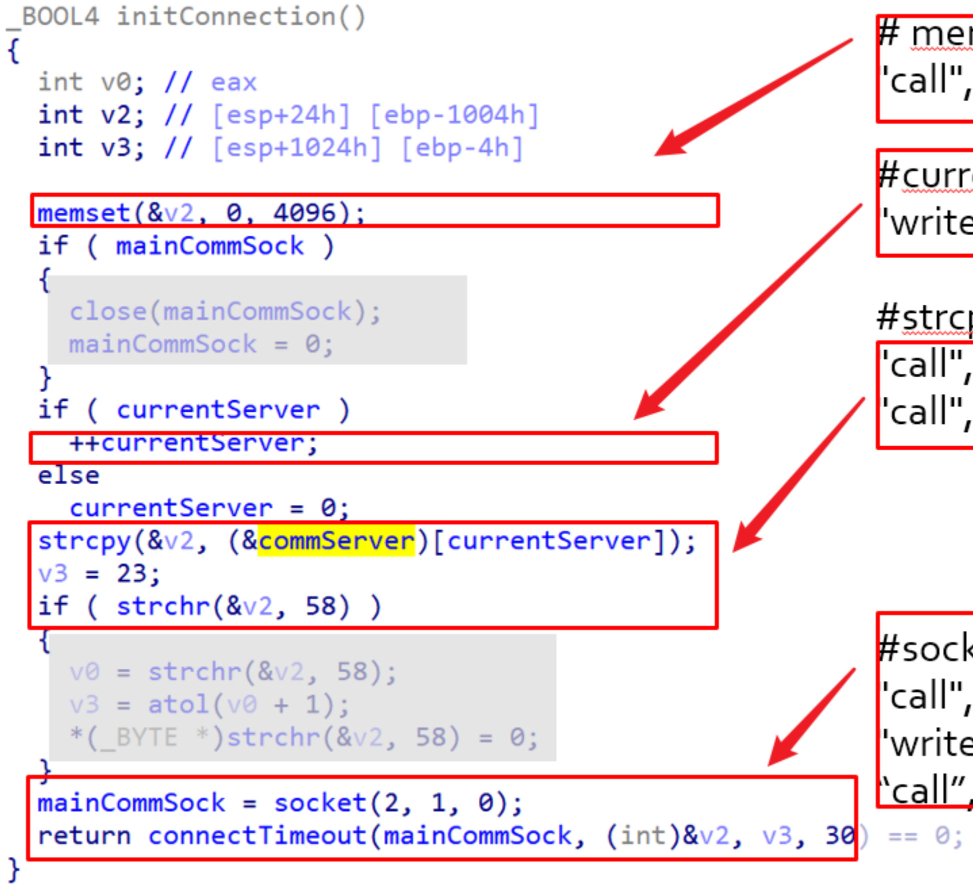
    "call", pc=0x0804dc4f, (0x0124eff8, 0, 0x1000, 0)
- When detecting ending address or an address beyond emulation range, it will stop the emulation

Hooking memory writes

- It’ s done through unicorn hook of UC_HOOK_MEM_WRITE
- Only writes to global memory are logged
  - to ignore writes to stack memory
- For each event, the logged information includes PC, write address, size, and value

```
"write", pc=0x0804dc8a, (0x080591b0, 0x00000004, 0x00000000)
```

Post analysis



```
_BOOL4 initConnection()
{
  int v0; // eax
  int v2; // [esp+24h] [ebp-1004h]
  int v3; // [esp+1024h] [ebp-4h]

  memset(&v2, 0, 4096);
  if ( mainCommSock )
  {
    close(mainCommSock);
    mainCommSock = 0;
  }
  if ( currentServer )
    ++currentServer;
  else
    currentServer = 0;
  strcpy(&v2, (&commServer)[currentServer]);
  v3 = 23;
  if ( strchr(&v2, 58) )
  {
    v0 = strchr(&v2, 58);
    v3 = atol(v0 + 1);
    *(_BYTE *)strchr(&v2, 58) = 0;
  }
  mainCommSock = socket(2, 1, 0);
  return connectTimeout(mainCommSock, (int)&v2, v3, 30) == 0;
}
```

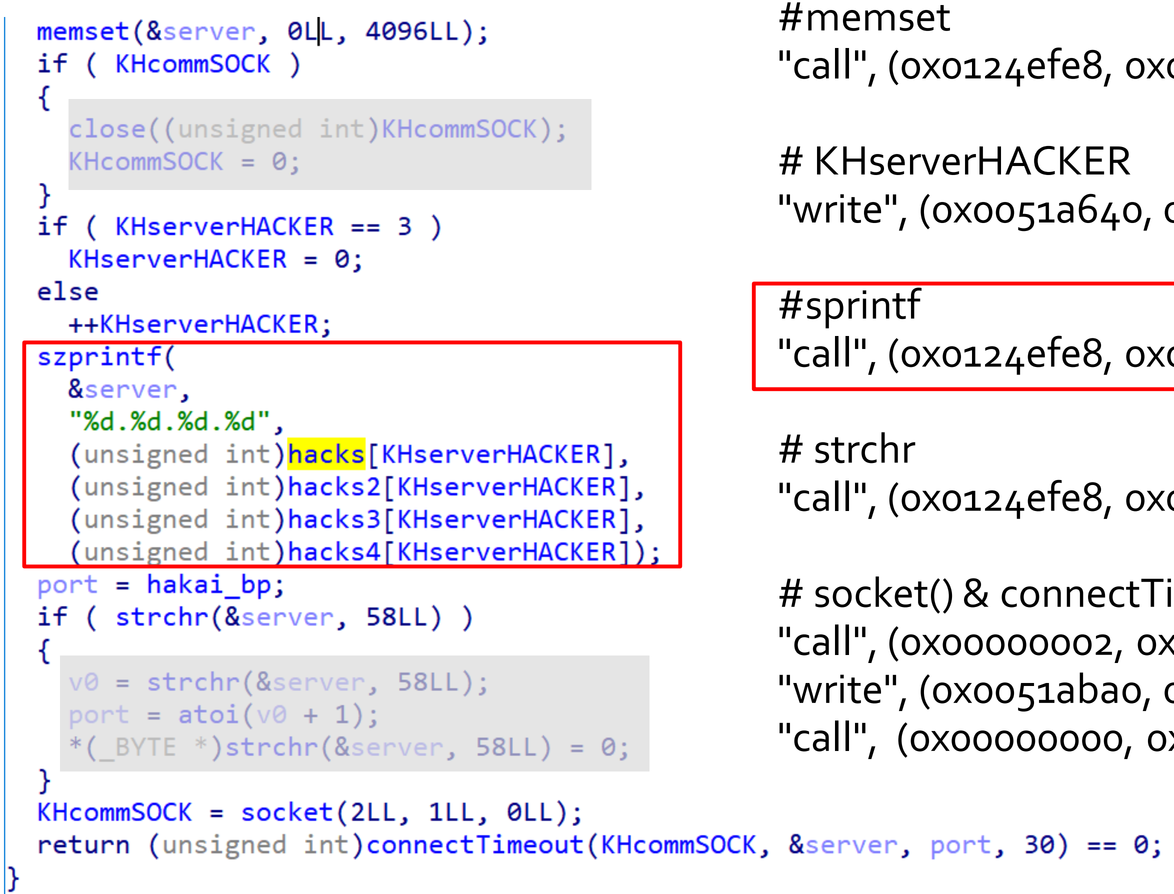
memset()
'call", (0x0124eff8, 0, 0x1000, 0)

#currentserver
'write", (0x080591b0, 0x00000004, 0x00000000)

#strcpy() & strchr()
'call", (0x0124eff8, **0x0805544d**)
'call", (0x0124eff8, 0x3a)

#socket () & connectTimeout()
'call", (2, 1, 0)
'write", (0x0805940, 0x00000004, 0x0000000)
“call”, (0x00000000, 0x0124eff8, 0x17, 0x1e)

Another version of initConnection()



```
  memset(&server, 0LL, 4096LL);
  if ( KHcommSOCK )
  {
    close((unsigned int)KHcommSOCK);
    KHcommSOCK = 0;
  }
  if ( KHserverHACKER == 3 )
    KHserverHACKER = 0;
  else
    ++KHserverHACKER;
  szprintf(
    &server,
    "%d.%d.%d.%d",
    (unsigned int)hacks[KHserverHACKER],
    (unsigned int)hacks2[KHserverHACKER],
    (unsigned int)hacks3[KHserverHACKER],
    (unsigned int)hacks4[KHserverHACKER]);
  port = hakai_bp;
  if ( strchr(&server, 58LL) )
  {
    v0 = strchr(&server, 58LL);
    port = atoi(v0 + 1);
    *(_BYTE *)strchr(&server, 58LL) = 0;
  }
  KHcommSOCK = socket(2LL, 1LL, 0LL);
  return (unsigned int)connectTimeout(KHcommSOCK, &server, port, 30) == 0;
}
```

```
#memset
"call", (0x0124efe8, 0x00000000, 0x00001000, 0x10101010)

# KHserverHACKER
"write", (0x0051a640, 0x00000004, 0x00000000)

#sprintf
"call", (0x0124efe8, 0x00417f10, 0xc6, 0x90, 0xb5, 0x11)

# strchr
"call", (0x0124efe8, 0x0000003a, 0x10101010, 0x00000090)

# socket() & connectTimeout()
"call", (0x00000002, 0x00000001, 0x00000000, 0x00000090)
"write", (0x0051aba0, 0x00000004, 0x00000000)
"call",  (0x00000000, 0x0124efe8, 0x00000e4f, 0x0000001e)
```

Behavior patterns and extraction rules

- IoC extraction is actually done in post analysis stage
  - applying specific behavior patterns on logged events
  - if matched, the extraction rules will be used to get the C2s
- In total, 6 types of initConnection() are concluded
- For each type a extraction rule is defined
  - Simplified pattern: for quick matching
  - Behavior pattern: for detailed matching
  - Extraction rules: actions to execute if matched successfully

An example extraction rule

Type 1

c: call

w4: 4-byte-write

w2: 2-byte-write

MD5= 00432f33fb3f5cc5377266a5439567bf, x86

Simplified pattern: “cw4cccw4c”

Behavior pattern: “call_memset, w4, call_strcpy, call_strchr, call_socket, w4, call_connectTimeout”

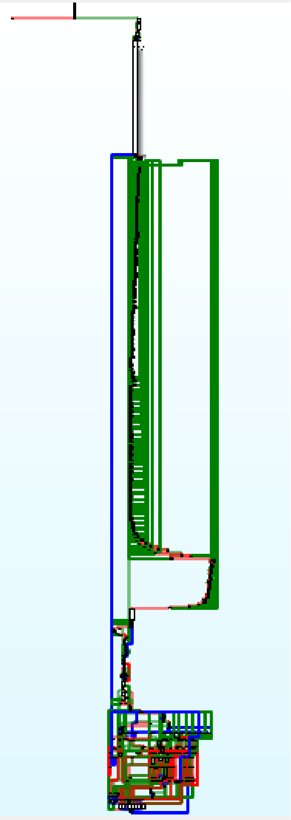
Static pattern: blocs=11, edges=14, called_functions=7, strs=["198.134.120.150:23"]

Extraction rules:

Reading global memory pointed by arg2 of strcpy() to get the string format of “C2:port”

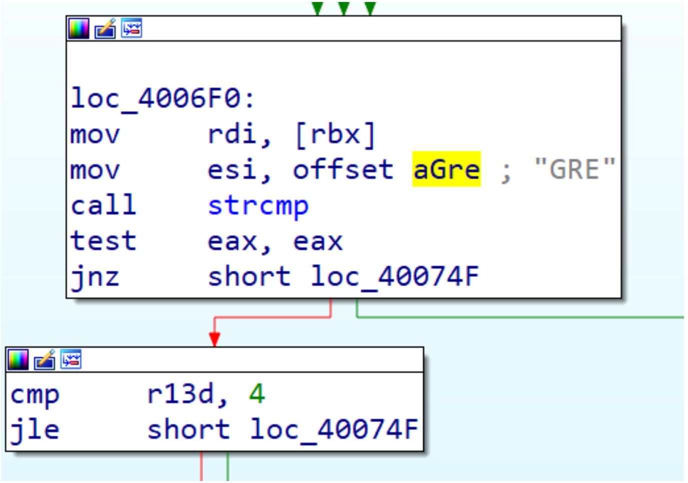
Mirai code in Gafgyt

Variants of Gafgyt + Mirai



scanner_init()

MD5= 00c183e4661881402f3dc90fd9f99c57



```
loc_4006F0:
mov     rdi, [rbx]
mov     esi, offset aGre ; "GRE"
call    strcmp
test    eax, eax
jnz     short loc_40074F
```

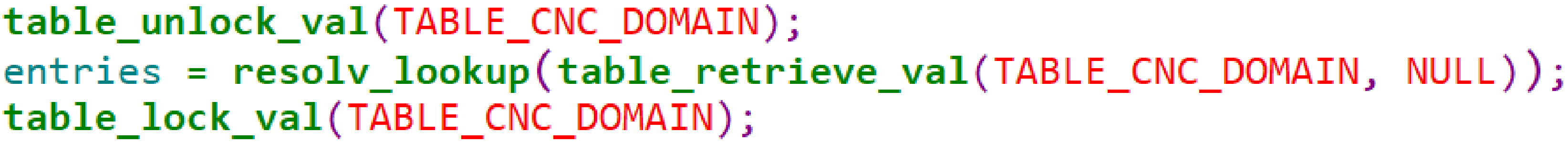
```
cmp     r13d, 4
jle     short loc_40074F
```

processCmd()

MD5= 05af2f5a37f3840ef7441c0f607a390a

Mirai’s Achilles heel

- A custom encrypted configuration db is heavily used in Mirai

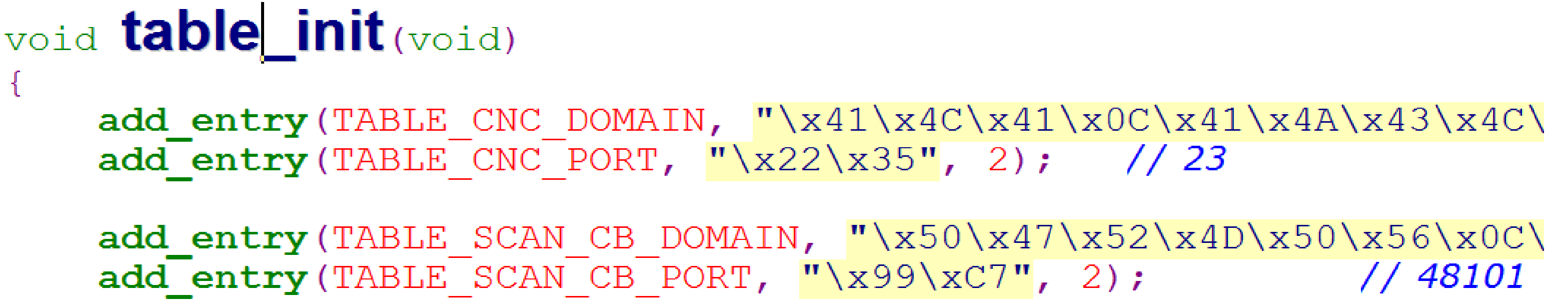


```
table_unlock_val(TABLE_CNC_DOMAIN);
entries = resolv_lookup(table_retrieve_val(TABLE_CNC_DOMAIN, NULL));
table_lock_val(TABLE_CNC_DOMAIN);
```

- It’ s usually copied together with the borrowed code
  – The original design is not bad
  – Its connections with other modules are too tight to be easily cut
  – The authors are lazy, or just don’ t know how to cut it
- Therefore it’ s possible to analyze Gafgyt+Mirai variants by analyzing their configurations

About configuration extraction

- It ’s also based on LWE, and was presented on VB2018
  - https://www.virusbulletin.com/virusbulletin/2018/12/vb2018-paper-tracking-mirai-variants/

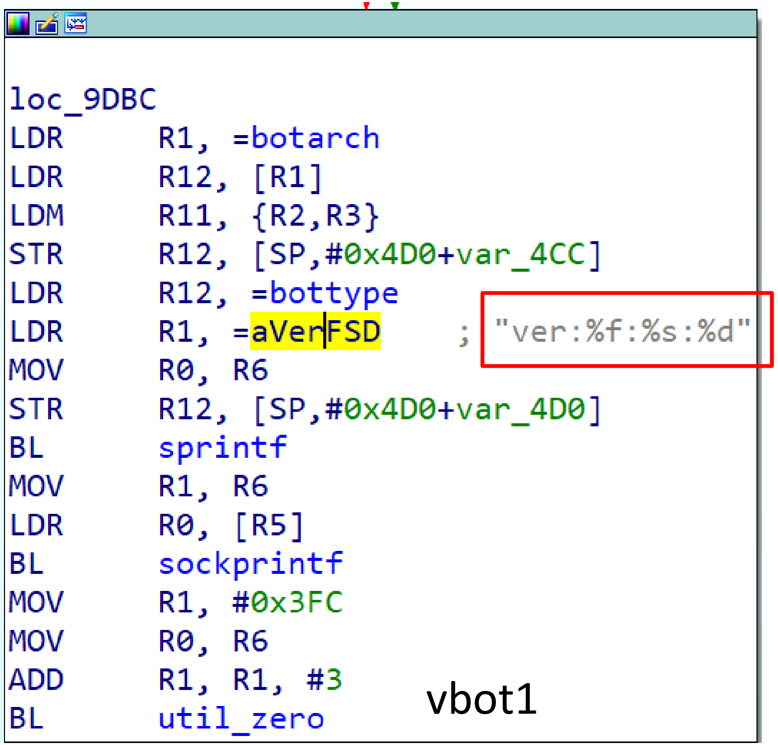


```
void table_init(void)
{
    add_entry(TABLE_CNC_DOMAIN, "\x41\x4C\x41\x0C\x41\x4A\x43\x4C\
    add_entry(TABLE_CNC_PORT, "\x22\x35", 2);   // 23

    add_entry(TABLE_SCAN_CB_DOMAIN, "\x50\x47\x52\x4D\x50\x56\x0C\
    add_entry(TABLE_SCAN_CB_PORT, "\x99\xC7", 2);          // 48101
```

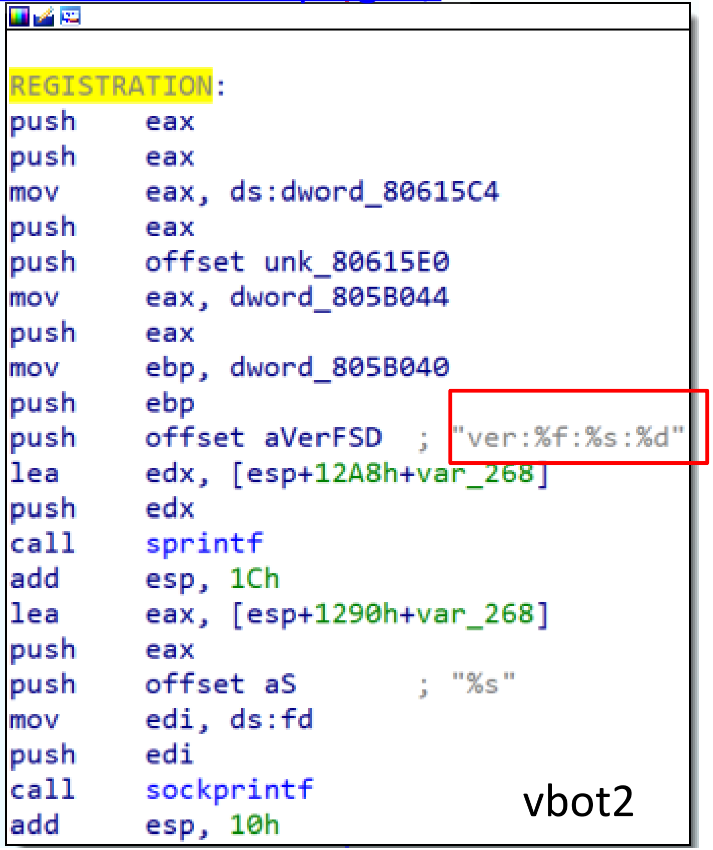
Gafgyt variant of vbot

https://blog.netlab.360.com/the-gafgyt-variant-vbot-and-its-31-campaigns/



```
loc_9DBC
LDR     R1, =botarch
LDR     R12, [R1]
LDM     R11, {R2,R3}
STR     R12, [SP,#0x4D0+var_4CC]
LDR     R12, =bottype
LDR     R1, =aVerFSD     ; "ver:%f:%s:%d"
MOV     R0, R6
STR     R12, [SP,#0x4D0+var_4D0]
BL      sprintf
MOV     R1, R6
LDR     R0, [R5]
BL      sockprintf
MOV     R1, #0x3FC
MOV     R0, R6
ADD     R1, R1, #3
BL      util_zero
```

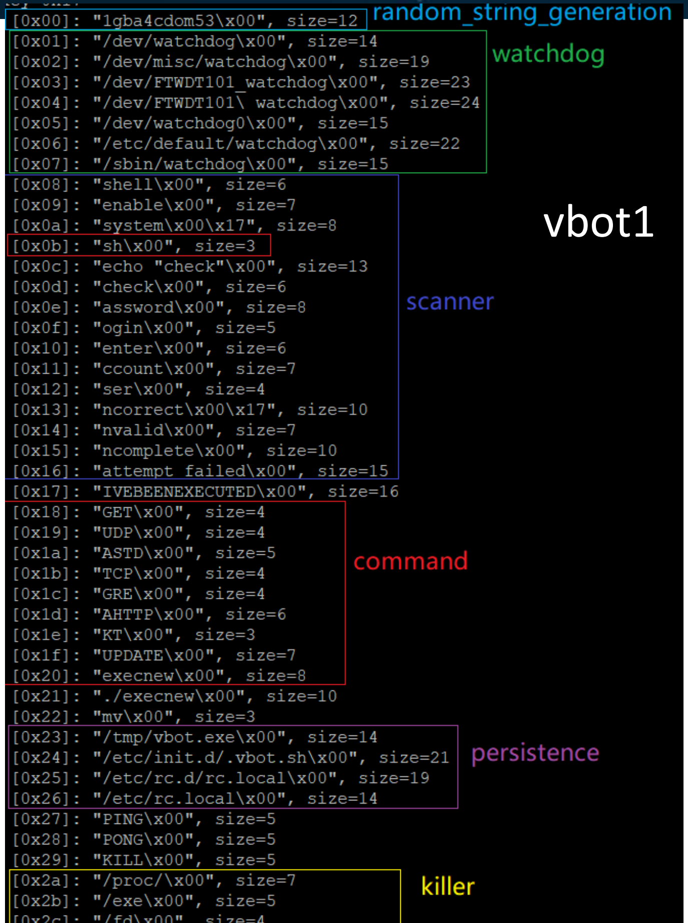
vbot1



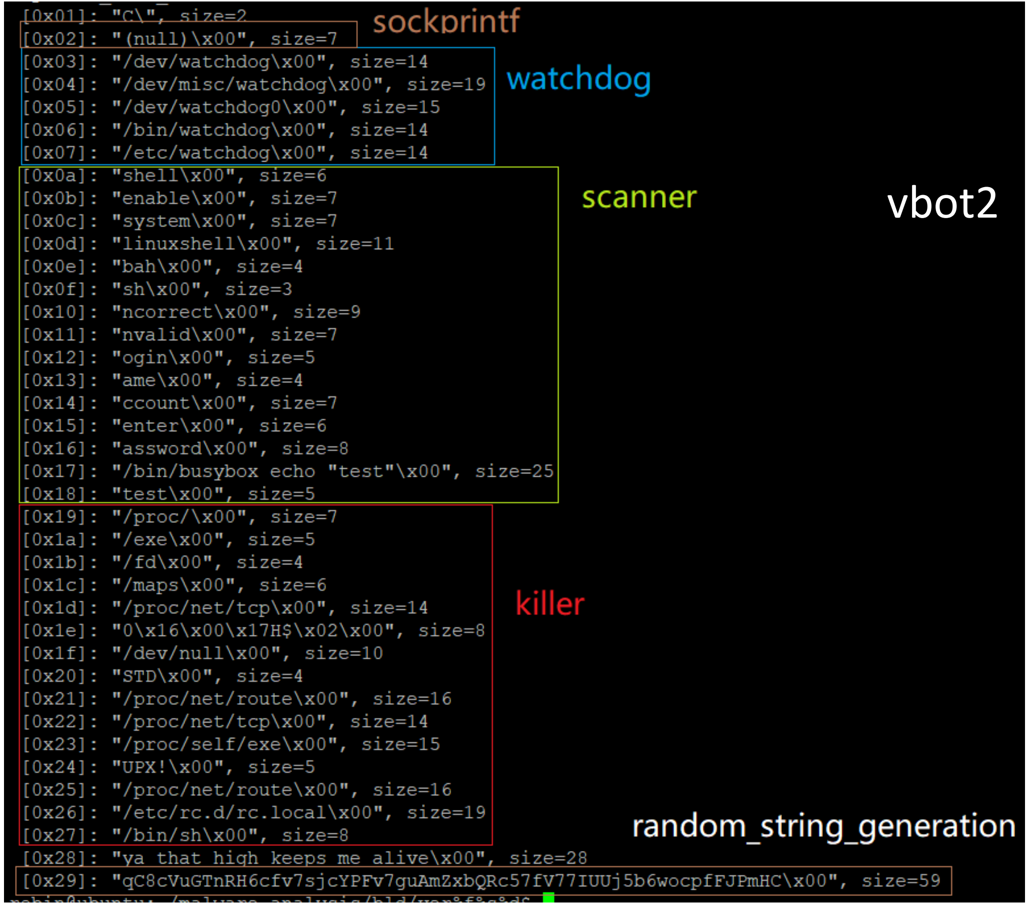
```
REGISTRATION:
push    eax
push    eax
mov     eax, ds:dword_80615C4
push    eax
push    offset unk_80615E0
mov     eax, dword_805B044
push    eax
mov     ebp, dword_805B040
push    ebp
push    offset aVerFSD  ; "ver:%f:%s:%d"
lea     edx, [esp+12A8h+var_268]
push    edx
call    sprintf
add     esp, 1Ch
lea     eax, [esp+1290h+var_268]
push    eax
push    offset aS          ; "%s"
mov     edi, ds:fd
push    edi
call    sockprintf
add     esp, 10h
```

vbot2

Configuration comparison



```
[0x00]: "1gba4cdom53\x00", size=12
[0x01]: "/dev/watchdog\x00", size=14
[0x02]: "/dev/misc/watchdog\x00", size=19
[0x03]: "/dev/FTWDT101_watchdog\x00", size=23
[0x04]: "/dev/FTWDT101\ watchdog\x00", size=24
[0x05]: "/dev/watchdog0\x00", size=15
[0x06]: "/etc/default/watchdog\x00", size=22
[0x07]: "/sbin/watchdog\x00", size=15
[0x08]: "shell\x00", size=6
[0x09]: "enable\x00", size=7
[0x0a]: "system\x00\x17", size=8
[0x0b]: "sh\x00", size=3
[0x0c]: "echo "check"\x00", size=13
[0x0d]: "check\x00", size=6
[0x0e]: "assword\x00", size=8
[0x0f]: "ogin\x00", size=5
[0x10]: "enter\x00", size=6
[0x11]: "ccount\x00", size=7
[0x12]: "ser\x00", size=4
[0x13]: "ncorrect\x00\x17", size=10
[0x14]: "nvalid\x00", size=7
[0x15]: "ncomplete\x00", size=10
[0x16]: "attempt failed\x00", size=15
[0x17]: "IVEBEENEXECUTED\x00", size=16
[0x18]: "GET\x00", size=4
[0x19]: "UDP\x00", size=4
[0x1a]: "ASTD\x00", size=5
[0x1b]: "TCP\x00", size=4
[0x1c]: "GRE\x00", size=4
[0x1d]: "AHTTP\x00", size=6
[0x1e]: "KT\x00", size=3
[0x1f]: "UPDATE\x00", size=7
[0x20]: "execnew\x00", size=8
[0x21]: "./execnew\x00", size=10
[0x22]: "mv\x00", size=3
[0x23]: "/tmp/vbot.exe\x00", size=14
[0x24]: "/etc/init.d/.vbot.sh\x00", size=21
[0x25]: "/etc/rc.d/rc.local\x00", size=19
[0x26]: "/etc/rc.local\x00", size=14
[0x27]: "PING\x00", size=5
[0x28]: "PONG\x00", size=5
[0x29]: "KILL\x00", size=5
[0x2a]: "/proc/\x00", size=7
[0x2b]: "/exe\x00", size=5
[0x2c]: "/fd\x00", size=4
```



```
[0x01]: "C\", size=2
[0x02]: "(null)\x00", size=7
[0x03]: "/dev/watchdog\x00", size=14
[0x04]: "/dev/misc/watchdog\x00", size=19
[0x05]: "/dev/watchdog0\x00", size=15
[0x06]: "/bin/watchdog\x00", size=14
[0x07]: "/etc/watchdog\x00", size=14
[0x0a]: "shell\x00", size=6
[0x0b]: "enable\x00", size=7
[0x0c]: "system\x00", size=7
[0x0d]: "linuxshell\x00", size=11
[0x0e]: "bah\x00", size=4
[0x0f]: "sh\x00", size=3
[0x10]: "ncorrect\x00", size=9
[0x11]: "nvalid\x00", size=7
[0x12]: "ogin\x00", size=5
[0x13]: "ame\x00", size=4
[0x14]: "ccount\x00", size=7
[0x15]: "enter\x00", size=6
[0x16]: "assword\x00", size=8
[0x17]: "/bin/busybox echo "test"\x00", size=25
[0x18]: "test\x00", size=5
[0x19]: "/proc/\x00", size=7
[0x1a]: "/exe\x00", size=5
[0x1b]: "/fd\x00", size=4
[0x1c]: "/maps\x00", size=6
[0x1d]: "/proc/net/tcp\x00", size=14
[0x1e]: "0\x16\x00\x17H$\x02\x00", size=8
[0x1f]: "/dev/null\x00", size=10
[0x20]: "STD\x00", size=4
[0x21]: "/proc/net/route\x00", size=16
[0x22]: "/proc/net/tcp\x00", size=14
[0x23]: "/proc/self/exe\x00", size=15
[0x24]: "UPX!\x00", size=5
[0x25]: "/proc/net/route\x00", size=16
[0x26]: "/etc/rc.d/rc.local\x00", size=19
[0x27]: "/bin/sh\x00", size=8
[0x28]: "ya that high keeps me alive\x00", size=28
[0x29]: "qC8cVuGTnRH6cfv7sjcYPFv7guAmZxbQRc57fV77IUUj5b6wocpfFJPmHC\x00", size=59
```

Some conclusions on vbot

- Although they shared the same registration code, they were obviously derived from different code bases

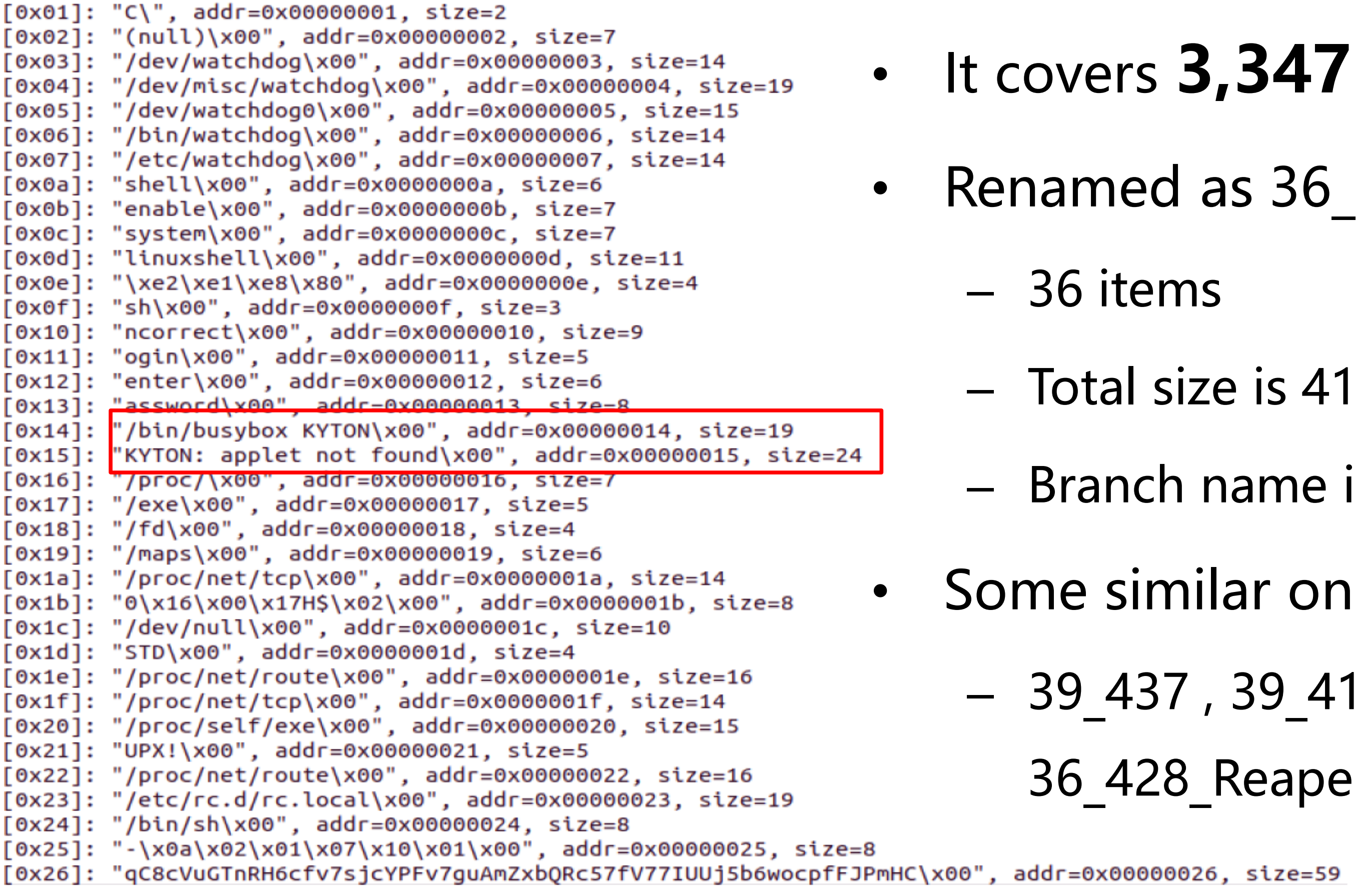
- Since the registration code is characteristic enough, both versions should have come from the same authors

- The authors have multiple sets of code bases

A summary of extracted configurations

- In total Mirai **16** configurations have been successfully extracted from **3,700** Gafgyt samples
- With the extracted configurations those samples can be well grouped
  - Each group of sample share the same configuration usage patterns
  - In most cases they can be classified as the same variant
- Similar configurations hint potential code sharing

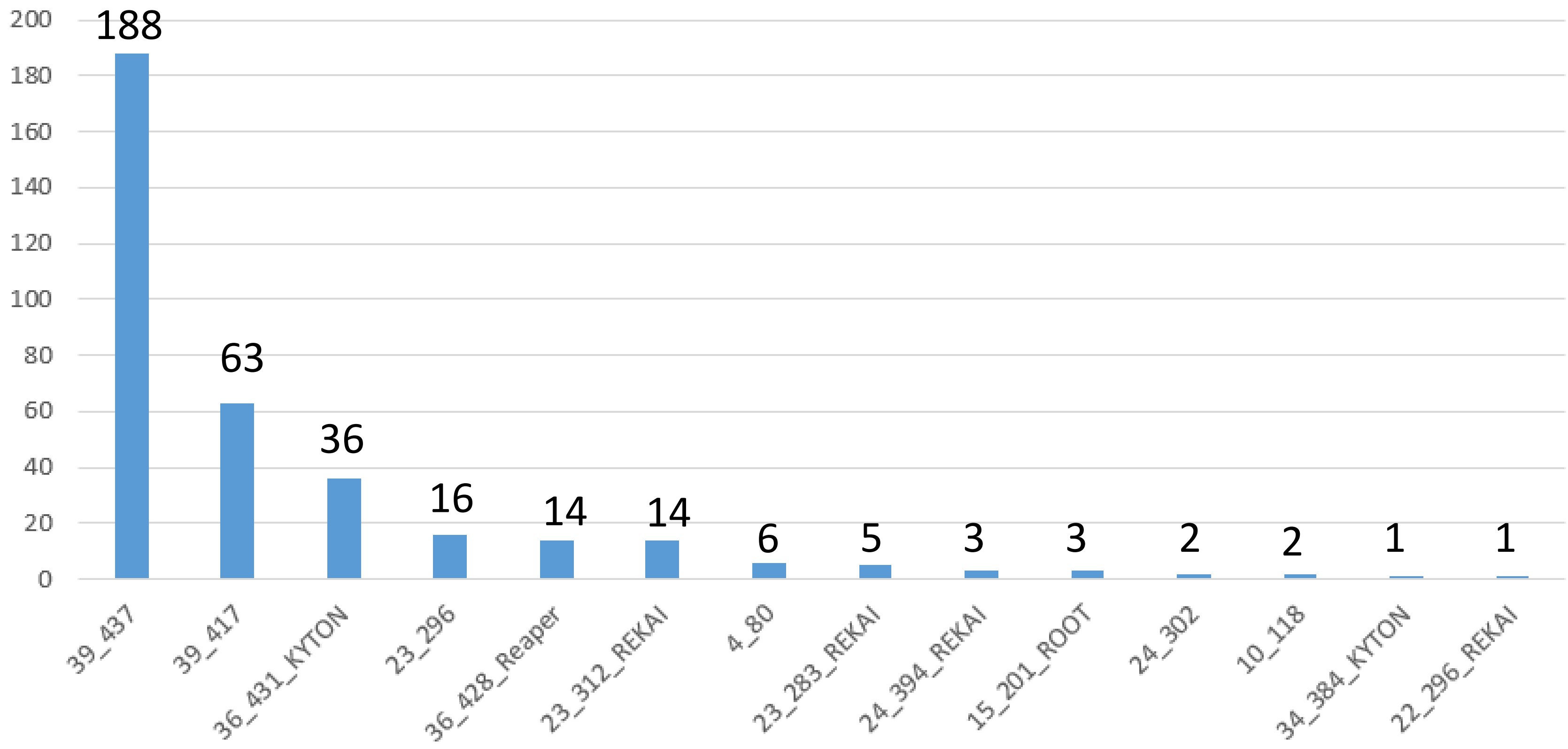
The most common configuration



```
[0x01]: "C\", addr=0x00000001, size=2
[0x02]: "(null)\x00", addr=0x00000002, size=7
[0x03]: "/dev/watchdog\x00", addr=0x00000003, size=14
[0x04]: "/dev/misc/watchdog\x00", addr=0x00000004, size=19
[0x05]: "/dev/watchdog0\x00", addr=0x00000005, size=15
[0x06]: "/bin/watchdog\x00", addr=0x00000006, size=14
[0x07]: "/etc/watchdog\x00", addr=0x00000007, size=14
[0x0a]: "shell\x00", addr=0x0000000a, size=6
[0x0b]: "enable\x00", addr=0x0000000b, size=7
[0x0c]: "system\x00", addr=0x0000000c, size=7
[0x0d]: "linuxshell\x00", addr=0x0000000d, size=11
[0x0e]: "\xe2\xe1\xe8\x80", addr=0x0000000e, size=4
[0x0f]: "sh\x00", addr=0x0000000f, size=3
[0x10]: "ncorrect\x00", addr=0x00000010, size=9
[0x11]: "ogin\x00", addr=0x00000011, size=5
[0x12]: "enter\x00", addr=0x00000012, size=6
[0x13]: "assword\x00", addr=0x00000013, size=8
[0x14]: "/bin/busybox KYTON\x00", addr=0x00000014, size=19
[0x15]: "KYTON: applet not found\x00", addr=0x00000015, size=24
[0x16]: "/proc/\x00", addr=0x00000016, size=7
[0x17]: "/exe\x00", addr=0x00000017, size=5
[0x18]: "/fd\x00", addr=0x00000018, size=4
[0x19]: "/maps\x00", addr=0x00000019, size=6
[0x1a]: "/proc/net/tcp\x00", addr=0x0000001a, size=14
[0x1b]: "0\x16\x00\x17H$\x02\x00", addr=0x0000001b, size=8
[0x1c]: "/dev/null\x00", addr=0x0000001c, size=10
[0x1d]: "STD\x00", addr=0x0000001d, size=4
[0x1e]: "/proc/net/route\x00", addr=0x0000001e, size=16
[0x1f]: "/proc/net/tcp\x00", addr=0x0000001f, size=14
[0x20]: "/proc/self/exe\x00", addr=0x00000020, size=15
[0x21]: "UPX!\x00", addr=0x00000021, size=5
[0x22]: "/proc/net/route\x00", addr=0x00000022, size=16
[0x23]: "/etc/rc.d/rc.local\x00", addr=0x00000023, size=19
[0x24]: "/bin/sh\x00", addr=0x00000024, size=8
[0x25]: "-\x0a\x02\x01\x07\x10\x01\x00", addr=0x00000025, size=8
[0x26]: "qC8cVuGTnRH6cfv7sjcYPFv7guAmZxbQRc57fV77IUUj5b6wocpfFJPmHC\x00", addr=0x00000026, size=59
```

- It covers **3,347** samples
- Renamed as 36_412_KYTON
  - 36 items
  - Total size is 412
  - Branch name is KYTON
- Some similar ones
  - 39_437 , 39_417, 36_431_KYTON, 36_428_Reaper

Stats on other 15 configurations



Conclusions

- Gafgtyt variants can be recognized with their characteristic C2 loops
- With C2 loops, both register message template and initConnection() function can be obtained
- C2 information can be got by lightweight emulating initConnection() together with the concluded behavior patterns
- Gafgyt + Mirai variants can be well analyzed with Mirai characteristic configurations

Thank you

twitter: @liuya0904
email: liuya@360.cn