基于 SPIKE 的网络协议 Fuzzing 技术

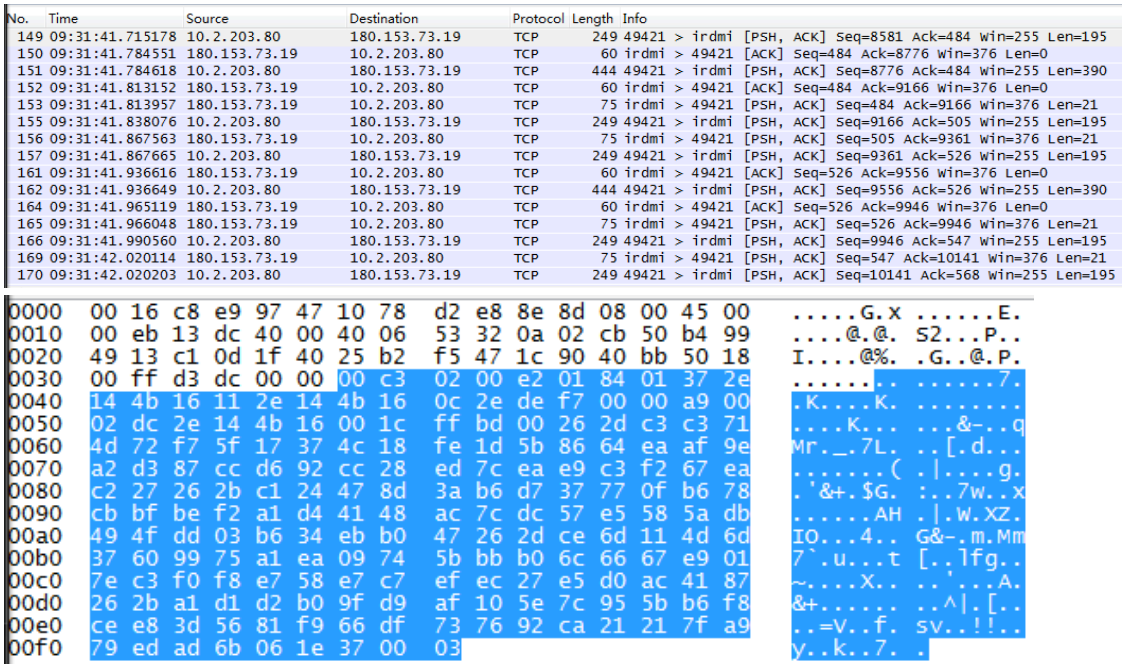
riusksk（泉哥）
腾讯安全应急响应中心
2011 年 12 月 30 日星期五

前言

SPIKE 是由 ImmunitySec 公司创始人 Dave Aitel 编写的黑盒测试工具，作者也是《the Hacker's Handbook》（中文版：《黑客防范手册》）和《the Shellcoder's Handbook》（中文版：《黑客攻防技术宝典：系统实战篇》）的作者之一。业界开源的 Fuzzing 工具很多，但对于网络协议的开源 Fuzzing 工具，还是以 spike 为主流，它主要是基于数据块的 Fuzz 测试，目前只支持 Linux 系统，其它 Fuzzing 工具还有 peach,、taof、sulley、beSTORM 等等。本文主要包括 Fuzzing 测试方法，spike 脚本编写并结合实例进行实战运用。

协议格式分析

以往采取 blind fuzzing 对未知协议进行模糊测试时，虽可减少工作量，但对成功率较低，而且发送的数据包容易因错误而被抛弃，导致很难进入到更深的处理逻辑，测试的广度就会受到限制。为了提高成功率，我们需要先对协议格式进行分析。协议分析主要是分析数据包的结构和格式、发送和接收的内容的顺序以及不同控制字段所代表的意义，包括发送方和接收方对各个字段的处理过程。目前主流协议分析工具当以 WireShark 为主，下面是针对 QTalk 语音聊天时捕获的数据包：



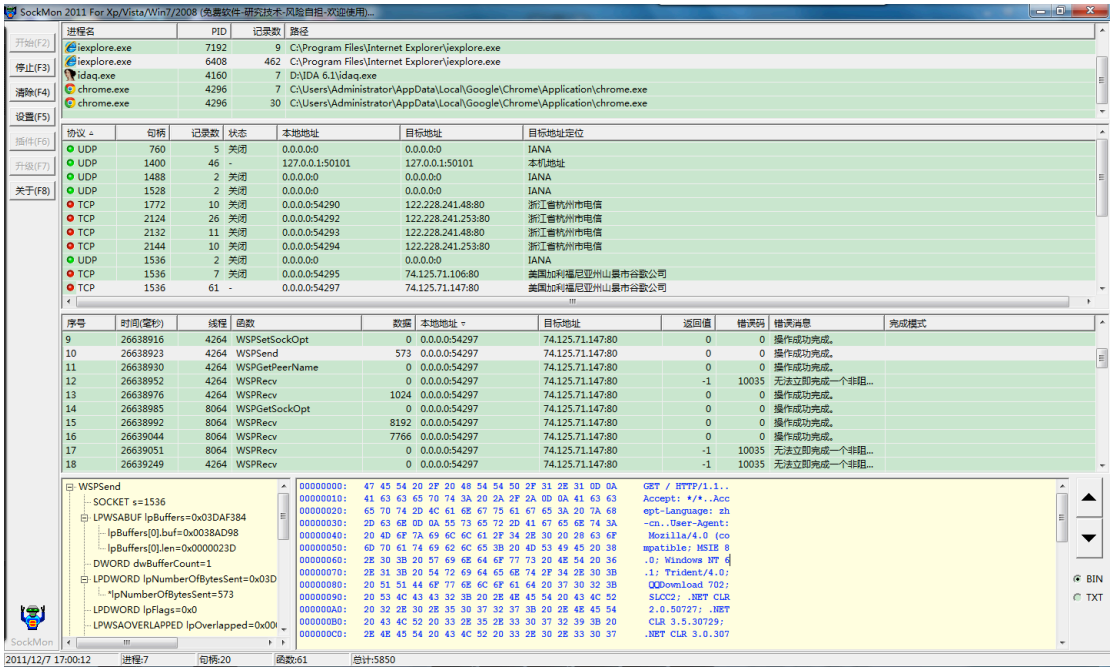
No.	Time	Source	Destination	Protocol	Length	Info
149	09:31:41.715178	10.2.203.80	180.153.73.19	TCP	249	49421 > irdmi [PSH, ACK] Seq=8581 Ack=484 Win=255 Len=195
150	09:31:41.784551	180.153.73.19	10.2.203.80	TCP	60	irdmi > 49421 [ACK] Seq=484 Ack=8776 Win=376 Len=0
151	09:31:41.784618	10.2.203.80	180.153.73.19	TCP	444	49421 > irdmi [PSH, ACK] Seq=8776 Ack=484 Win=255 Len=390
152	09:31:41.813152	180.153.73.19	10.2.203.80	TCP	60	irdmi > 49421 [ACK] Seq=484 Ack=9166 Win=376 Len=0
153	09:31:41.813957	180.153.73.19	10.2.203.80	TCP	75	irdmi > 49421 [PSH, ACK] Seq=484 Ack=9166 Win=376 Len=21
155	09:31:41.838076	10.2.203.80	180.153.73.19	TCP	249	49421 > irdmi [PSH, ACK] Seq=9166 Ack=505 Win=255 Len=195
156	09:31:41.867563	180.153.73.19	10.2.203.80	TCP	75	irdmi > 49421 [PSH, ACK] Seq=505 Ack=9361 Win=376 Len=21
157	09:31:41.867665	10.2.203.80	180.153.73.19	TCP	249	49421 > irdmi [PSH, ACK] Seq=9361 Ack=526 Win=255 Len=195
161	09:31:41.936616	180.153.73.19	10.2.203.80	TCP	60	irdmi > 49421 [ACK] Seq=526 Ack=9556 Win=376 Len=0
162	09:31:41.936649	10.2.203.80	180.153.73.19	TCP	444	49421 > irdmi [PSH, ACK] Seq=9556 Ack=526 Win=255 Len=390
164	09:31:41.965119	180.153.73.19	10.2.203.80	TCP	60	irdmi > 49421 [ACK] Seq=526 Ack=9946 Win=376 Len=0
165	09:31:41.966048	180.153.73.19	10.2.203.80	TCP	75	irdmi > 49421 [PSH, ACK] Seq=526 Ack=9946 Win=376 Len=21
166	09:31:41.990560	10.2.203.80	180.153.73.19	TCP	249	49421 > irdmi [PSH, ACK] Seq=9946 Ack=547 Win=255 Len=195
169	09:31:42.020114	180.153.73.19	10.2.203.80	TCP	75	irdmi > 49421 [PSH, ACK] Seq=547 Ack=10141 Win=376 Len=21
170	09:31:42.020203	10.2.203.80	180.153.73.19	TCP	249	49421 > irdmi [PSH, ACK] Seq=10141 Ack=568 Win=255 Len=195

```
0000  00 16 c8 e9 97 47 10 78  d2 e8 8e 8d 08 00 45 00   .....G.x ......E.
0010  00 eb 13 dc 40 00 40 06  53 32 0a 02 cb 50 b4 99   ....@.@. S2...P..
0020  49 13 c1 0d 1f 40 25 b2  f5 47 1c 90 40 bb 50 18   I....@%. .G..@.P.
0030  00 ff d3 dc 00 00 00 c3  02 00 e2 01 84 01 37 2e   ........ ......7.
0040  14 4b 16 11 2e 14 4b 16  0c 2e de f7 00 00 a9 00   .K....K. ........
0050  02 dc 2e 14 4b 16 00 1c  ff bd 00 26 2d c3 c3 71   ....K... ...&-..q
0060  4d 72 f7 5f 17 37 4c 18  fe 1d 5b 86 64 ea af 9e   Mr._.7L. ..[.d...
0070  a2 d3 87 cc d6 92 cc 28  ed 7c ea e9 c3 f2 67 ea   .......( .|....g.
0080  c2 27 26 2b c1 24 47 8d  3a b6 d7 37 77 0f b6 78   .'&+.$G. :..7w..x
0090  cb bf be f2 a1 d4 41 48  ac 7c dc 57 e5 58 5a db   ......AH .|.W.XZ.
00a0  49 4f dd 03 b6 34 eb b0  47 26 2d ce 6d 11 4d 6d   IO...4.. G&-.m.Mm
00b0  37 60 99 75 a1 ea 09 74  5b bb b0 6c 66 67 e9 01   7`.u...t [..lfg..
00c0  7e c3 f0 f8 e7 58 e7 c7  ef ec 27 e5 d0 ac 41 87   ~....X.. ..'...A.
00d0  26 2b a1 d1 d2 b0 9f d9  af 10 5e 7c 95 5b b6 f8   &+...... ..^|.[..
00e0  ce e8 3d 56 81 f9 66 df  73 76 92 ca 21 21 7f a9   ..=V..f. sv..!!..
00f0  79 ed ad 6b 06 1e 37 00  03                       y..k..7. .
```

另外也可再结合 IDA、OD 等逆向工具进行分析，跟踪数据包的处理过程。比如针对某 FTP 服务端软件，借助 IDA 查看程序携带的字符串，通常可到找到一些该软件支持的 FTP 命令，然后我们再以“Command [arg]”这样的形式进行 Fuzzing，比如 Free Float FtpServer 软件：

Address	Length	Type	String
.data:0040A2F4	00000006	C	DELE
.data:0040A2FC	00000018	C	RMD command successful.
.data:0040A314	00000006	C	XRMD
.data:0040A31C	00000035	C	: can't create a file when that file already exists.
.data:0040A354	00000018	C	MKD command successful.
.data:0040A36C	00000006	C	XMKD
.data:0040A374	0000000E	C	Type set to A
.data:0040A384	00000007	C	TYPE A
.data:0040A38C	0000000E	C	Type set to I
.data:0040A39C	00000007	C	TYPE I
.data:0040A3A4	00000006	C	STOR
.data:0040A3AC	00000006	C	RETR
.data:0040A3B4	00000005	C	NLST
.data:0040A3BC	00000005	C	LIST
.data:0040A3C4	0000001C	C	: can't find the directory.
.data:0040A3E0	00000018	C	CWD command successful.
.data:0040A3F8	00000005	C	CWD
.data:0040A400	00000005	C	XPWD
.data:0040A40C	00000019	C	PORT command successful.
.data:0040A428	00000006	C	PORT
.data:0040A430	0000000C	C	logged in.
.data:0040A43C	00000006	C	User
.data:0040A444	00000006	C	PASS
.data:0040A450	00000017	C	Password required for
.data:0040A468	00000006	C	USER
.data:0040A470	00000008	C	Goodbye
.data:0040A478	00000005	C	QUIT
.data:0040CAEC	00000006	C	`停

另外我们还可通过对常用函数如 recv、recvfrom、send 等 API 函数进行下断，然后再结合抓包工具所捕获的数据来跟踪调试，从而定位到特定数据包的处理函数，以了解程序对数据包的处理逻辑。由于当前许多协议支持加密功能，特别是一些 IM 即时通讯软件，为了保护用户隐私都对传输数据进行加密处理，因此跟进数据包处理函数时可能就遇到一些加密算法，对此我们可先用 PEID 上的插件检测程序拥有的加密算法，以便在逆向分析时更快地识别出相应算法。同时还应注意发包的时序性，每个数据包都是按照一定的时序来发送接收的，也就是说发送一个数据包后得到的数据包也是特定的。抓包工具除了 WireShark 之外，这里推荐一款应用层抓包工具 SockMon，它可针对特定进程、协议、IP、网络函数进行抓包，而且还提供调用栈情况，可帮助我们作进一步的逆向分析。SockMon 界面如下：



构造 Fuzzer

在对协议格式进行分析之后，我们就需要根据网络数据包格式来构造 spk 脚本。基于 SPIKE 框架的 Fuzzer 主要可分为两部分：一是主控程序，即测试软件，主要用于实现数据包发送与接收；二是 SPK 测试脚本，用于构造数据包，并在其中插入各种畸形数据。Spike 提供有一套 API 函数用于实现自动化测试，可以帮助你针对不同的协议，灵活构造合适的数据块进行 Fuzzing 测试，以便将数据包传送到更深的处理逻辑，覆盖更广的测试面，以提供成功率。Spike 主要是在命令行下操作，本文以 BackTrack 系统作为操作环境。目前官方并没有提供完善的 spike 使用文档，只附有几篇安全大会上文章而已，因此只能通过阅读 spike 源码及其测试脚本代码来学习如何使用。下面是整理出的一些常用 API 函数：

字符串：

```
s_cstring("abc") // 添加 C 类型（以 NULL 结尾）的字符串
s_unistring("str")// 添加 unicode 字符串
s_string("str")          // 添加固定字符串，在测试时永不改变
s_xdr_string("str")    // 添加 xdr 类型的字符串，即包含 4 字节长度的标签，并用 0 来扩展 4 倍长度，在测试时永不改变
s_unistring_variable(unsigned char *variable)        // 添加 unicode 字符串变量
s_string_variable(“abc”);  // 添加字符串变量
s_string_repeat(“string”,200); // 替换“string” 200 次
s_add_fuzzstring(unsigned char * newfuzzstring)  // 添加自定义畸形字符
s_init_fuzzing() // 使用 spike 自带的畸形数据库
```

二进制：

```
s_binary("A0 E4")          // 添加二进制数据
s_binary_repeat(“\\x41″, 200); // 替换 0x41 200 次
```

整数：

```
s_int_variable(int defaultvalue, int type)    // 添加整数变量
s_add_fuzzint(unsigned long fuzzint)  // 添加自定义的畸形整数值
```

块：

```
s_block_start("block1"); // 定义块“block1″的起始处
s_block_end("block1"); // 定义块“block1″的结尾处
s_blocksize_string("block1", 2);  // 添加 2 字符长度来表示块"block1"的大小
s_binary_block_size_intel_word();  // 获取得 block 的大小
s_binary_block_size_byte("block1");  // 添加 1 字节值来表示块"block1"的大小
```

网络传输：

```
s_read_packet（）; // 读取从服务器收到的数据并打印到屏幕上
s_readline（）;  // 读取一行数据
spike_send_udp(char * host, int port)  // 向 udp 端口并发送 spike 数据
spike_send_tcp(char * host, int port)  // 向 tcp 端口并发送 spike 数据
spike_connect_udp(char * host, int port)  // 连接到 udp 端口
spike_listen_udp(int port) // 监听 udp 端口
spike_connect_tcp(char * host, int port)  // 连接到 tcp 端口
```

```
s_tcp_accept(int listenfd)   // 接受 tcp 连接
s_read_packet()        // 读取数据包
s_close_udp()    // 关闭 udp 连接
spike_close_tcp()      // 关闭 tcp 连接
```

下面以发送 POST 请求为例来构造 Fuzzing 脚本：

```
s_string("POST /test.php HTTP/1.1\r\n");
s_string("Host: server.example.com\r\n");
s_string("Content-Length: ");
s_blocksize_string("block1″, 5);
s_string("\r\nConnection: close\r\n\r\n");
s_block_start("block1″);
s_string("inputvar=");
s_string_variable("inputval");
s_block_end("block1″);
```

上面的脚本相当于发送以下 POST 请求：

```
POST /test.php HTTP/1.1
Host: server.example.com
Content-Length: [size_of_data]
Connection: close
inputvar=[fuzz_string]
```

其中主要针对 inputvar 变量进行 fuzzing 测试，而 Content-Length 的值是随着 block1（也就是 inputvar=[fuzz_string]）大小的变化而变化的，以满足动态变化的需求。我们将以上脚本保存为 spk 文件，然后用 generic_send_tcp 来执行：

```
./generic_send_tcp IP PORT SPKFILE SKIPVAR SKIPSTR
```

上面的 spkfile 就是上面创建的 spk 文件，skipvar 就是忽略的变量个数，skipstr 就是忽略的字符串个数，比如在测试某变量时已经崩溃了，在下次测试时我们就可以直接跳过该变量。例如：

```
./generic_send_tcp 127.0.0.1 21 test.spk 0 0
```

除了编写 spk 脚本之外，你可以直接用 C 语言来编写自己的 Fuzzer，可借助 spike 提供的 API 函数来实现，现在将上面的脚本转换成 C 代码：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
```

```
#include <sys/socket.h>

#include "spike.h"
#include "hdebug.h"
#include "tcpstuff.h"

void usage()
{
      fprintf(stderr,"Usage: ./post_spike target port\r\n");
      exit(-1);
}

int main (int argc, char ** argv)
{
char * target;
char buffer[1500000];
int port;

struct spike * our_spike;
unsigned long retval;
int i;

if (argc!=3)
      {
         usage();
      }

target=argv[1];
printf("Target is %s\r\n",argv[1]);

port=atoi(argv[2]);

our_spike=new_spike();

if (our_spike==NULL)
      {
         fprintf(stderr,"Malloc failed trying to allocate a spike.\r\n");
         exit(-1);
      }

setspike(our_spike);

memset(buffer,0x41,sizeof(buffer));
buffer[sizeof(buffer)]=0;
```

```
for (i=0; i<500; i+=4)
  {
     memcpy(buffer+i,"%25s",4);
  }

buffer[140000]=0;
printf("Buffer size = %d\r\n",strlen(buffer));

s_string(“POST /test.php HTTP/1.1\r\n”);
s_string(“Host: server.example.com\r\n”);
s_string(“Content-Length: “);
s_blocksize_string(“block1″, 5);
s_string(“\r\nConnection: close\r\n\r\n”);
s_block_start(“block1″);
s_string(“inputvar=”);
s_string_variable(“inputval”);
s_block_end(“block1″);

printf("Sending to %s on port %d\r\n",target,port);
if (spike_send_tcp(target,port)<0)
  {
     printf("Couldn't connect to host or send data!\r\n");
     exit(-1);
  }

printf("reading\r\n");
memset(buffer,0x00,sizeof(buffer));
retval=1;
while (retval!=-1)
  {
     s_fd_wait();
     retval=read(our_spike->fd,buffer,1500);
     printf("%s",buffer);
  }
spike_close_tcp()
return 0;
}
```

实例应用

1、FTP 协议测试

文件传输协议（FileTransferProtocol,FTP）是一个已公开的协议，它属于网络协议组的应用层，主要是用于在网络上进行文件传输，以实现 Internet 上的控制文件的双向传输。网上有不少 FTP 服务端软

件，可用于在本地搭建 FTP 服务，这些服务端软件都提供一些命令给用户操作，比如 USER、PASS、LIST 等常见命令，为了获取完整的一套指令，可通过逆向工程获取到，或者通过输入 HELP 命令获取，或者通过阅读软件的帮助文档，然后我们再对这套指令进行 Fuzzing 测试。下面针对 FTP 服务端软件的 spk 脚本代码：

```
s_readline();
s_string_variable("USER ");
s_string_variable("Anonymous");
s_string("\r\n");
s_string("PASS ");
s_string_variable(" ");
s_string("\r\n");
s_string("HOST ");
s_string_variable("10.2.203.86");
s_string("\r\n");
s_string("ABOR ");
s_string_variable("..?");
s_string("\r\n");
s_string("AUTH ");
s_string_variable("SSL");
s_string("\r\n");
s_string("CDUP ");
s_string_variable("dir");
s_string("\r\n");
s_string("FEAT ");
s_string_variable("F");
s_string("\r\n");
s_string("MDTM ");
s_string_variable("/");
s_string("\r\n");
s_string("NOOP ");
s_string_variable("CRLF");
s_string("\r\n");
s_string("OPTS ");
s_string_variable("O");
s_string("\r\n");
s_string("PASV ");
s_string_variable("P");
s_string("\r\n");
s_string("PROT ");
s_string_variable("p");
s_string("\r\n");
s_string("PWD ");
s_string_variable(".");
```

```
s_string("\r\n");
s_string("SYST ");
s_string_variable(".");
s_string("\r\n");
s_string("XCUP ");
s_string_variable("x");
s_string("\r\n");
s_string("XCRC ");
s_string_variable("132");
s_string("\r\n");
s_string("XCWD ");
s_string_variable(".");
s_string("\r\n");
s_string("XMKD ");
s_string_variable(".");
s_string("\r\n");
s_string("XPWD ");
s_string_variable(".");
s_string("\r\n");
s_string("XRMD ");
s_string_variable(".");
s_string("\r\n");
s_string("SITE ");
s_string_variable("SEDV");
s_string("\r\n");
s_string("ACCT ");
s_string_variable("bob");
s_string("\r\n");
s_string("CWD ");
s_string_variable(".");
s_string("\r\n");
s_string("SMNT ");
s_string_variable(".");
s_string("\r\n");
s_string("PORT ");
s_string_variables(",","1,2,3,4,5,6");
s_string("\r\n");
s_string("TYPE ");
s_string_variable("I");
s_string("\r\n");
s_string("STRU ");
s_string_variable("P");
s_string("\r\n");
s_string("MODE ");
```

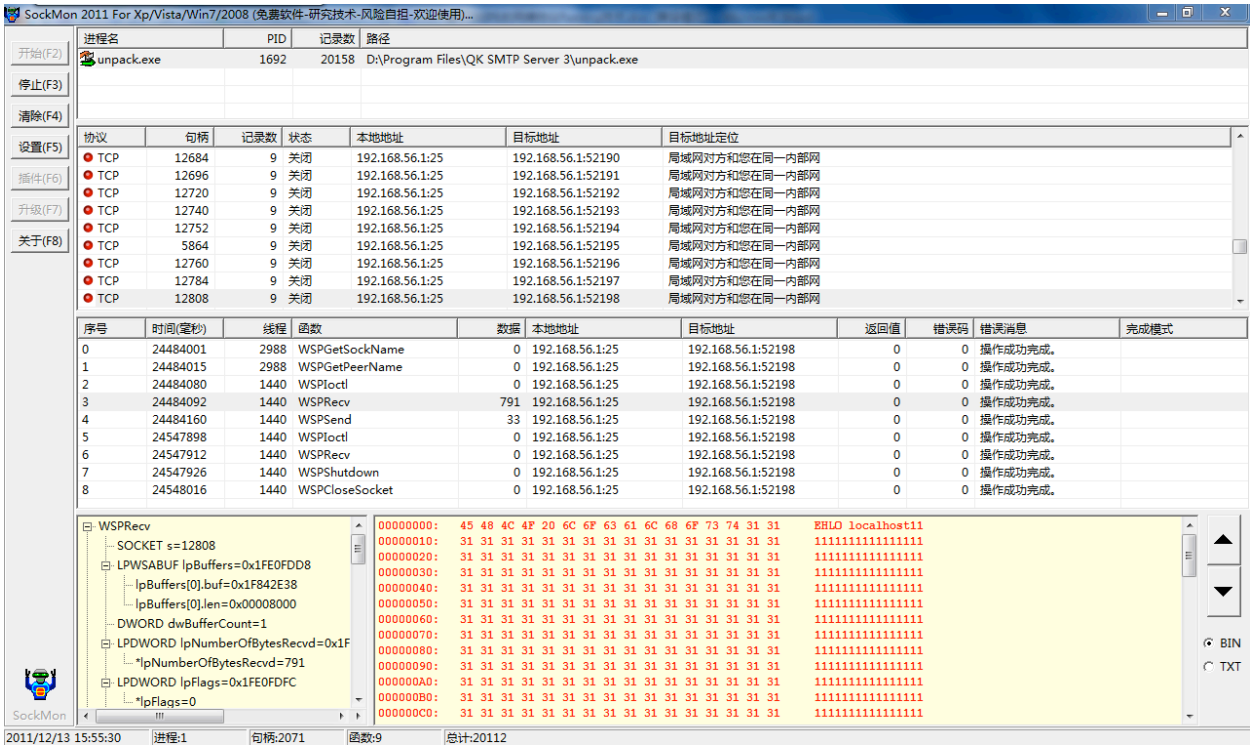
```
s_string_variable("C");
s_string("\r\n");
s_string("RETR ");
s_string_variable("filename");
s_string("\r\n");
s_string("STOR ");
s_string_variable("filename");
s_string("\r\n");
s_string("STOU ");
s_string_variable("filename");
s_string("\r\n");
s_string("ALLO ");
s_string_variables(" ","1 R 1");
s_string("\r\n");
s_string("APPE ");
s_string_variable("filename");
s_string("\r\n");
s_string("REST ");
s_string_variable("1");
s_string("\r\n");
s_string("RNFR ");
s_string_variable("filename");
s_string("\r\n");
s_string("RNTO ");
s_string_variable("newfilename");
s_string("\r\n");
s_string("DELE ");
s_string_variable("filename");
s_string("\r\n");
s_string("MKD ");
s_string_variable("dir");
s_string("\r\n");
s_string("RMD ");
s_string_variable("dir");
s_string("\r\n");
s_string("LIST ");
s_string_variable("dir");
s_string("\r\n");
s_string("NLST ");
s_string_variable("dir");
s_string("\r\n");
s_string("STAT ");
s_string_variable(".");
s_string("\r\n");
```

```
s_string("HELP ");
s_string_variable("SITE");
s_string("\r\n");
s_string("MLST ");
s_string_variable("dir");
s_string("\r\n");
s_string("LANG ");
s_string_variable("A");
s_string("\r\n");
s_string("SIZE ");
s_string_variable("A");
s_string("\r\n");
s_string("UTF8 ");
s_string_variable("A");
s_string("\r\n");
```

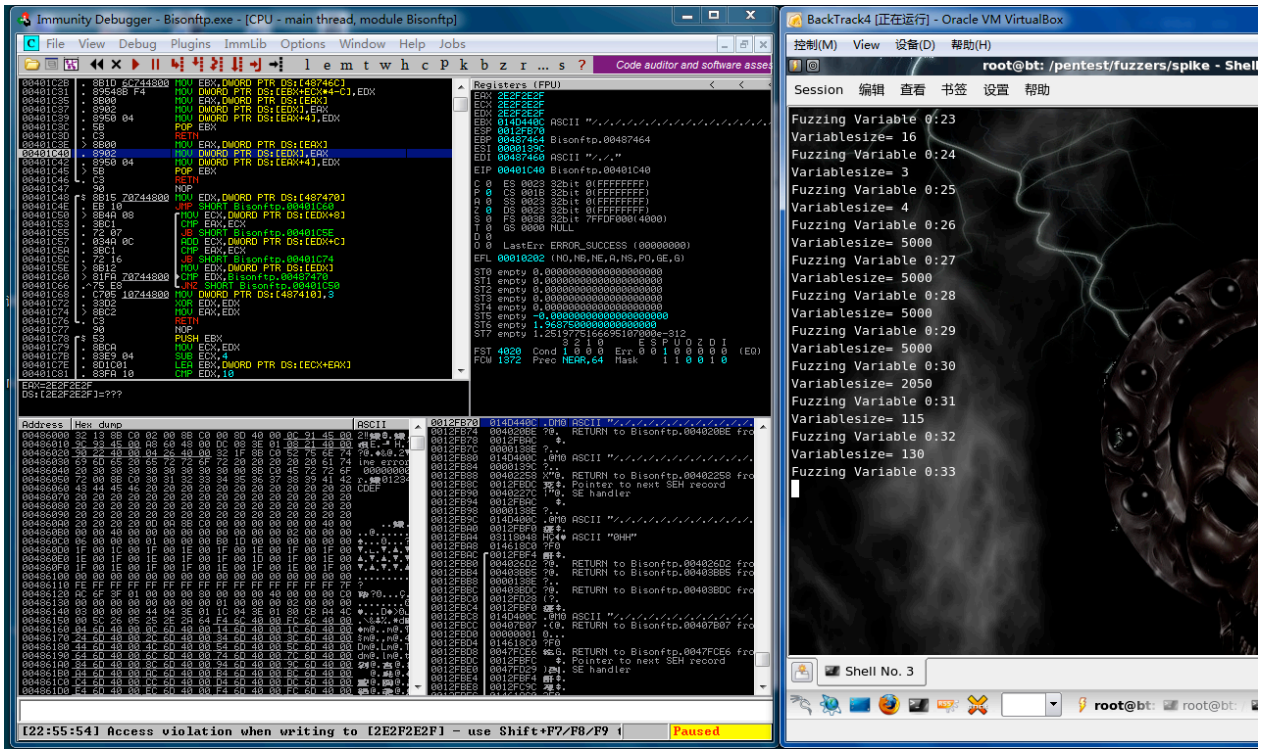
在测试过程中，我们可以同时抓包，以确定发送的数据包内容，在 BT 上面可以直接挂载 windows 上的硬盘，然后用 tcpdump 抓包保存在上面的挂载的位置，然后在 win 上打开 cap：

```
root@bt:~# mkdir /mnt/win7
root@bt:~# mount -t vboxsf win7 /mnt/win7
root@bt: ~# tcpdump -vv -s 0 -p -w /mnt/win7/test.cap
```

除以上方法外，也可使用 sockmon2011 进行抓包，这里强烈推荐：



下面以 BisonWare FTP Server V3.5 为例进行测试，该软件存在任意代码执行漏洞，以下是 Fuzzing 结果：



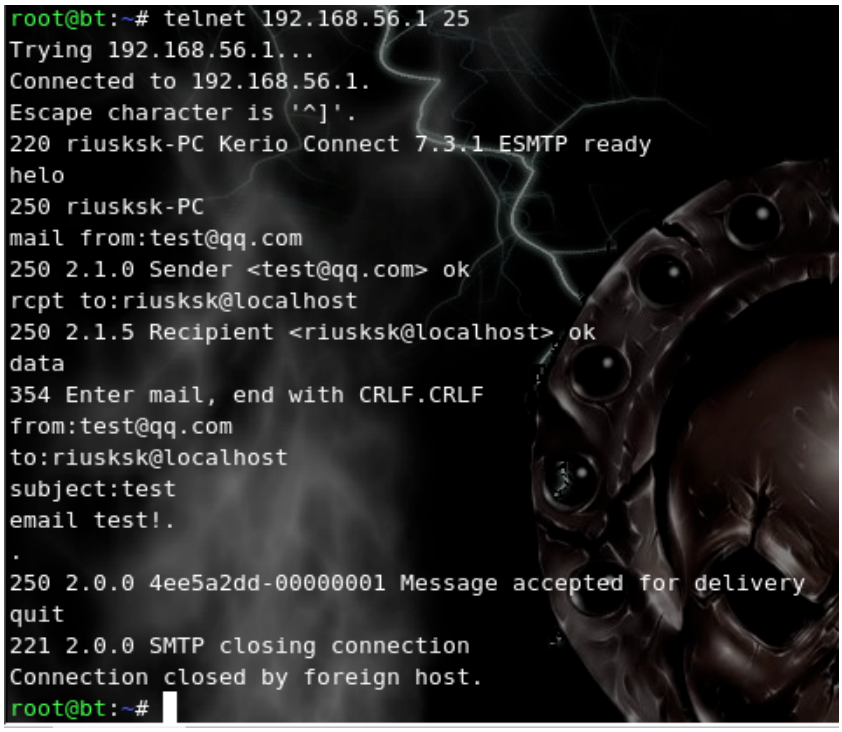
2、SMTP 协议测试

SMTP（Simple Mail Transfer Protocol）即简单邮件传输协议，它是一组用于由源地址到目的地址传送邮件的规则，由它来控制信件的中转方式。SMTP 协议属于 TCP/IP 协议族，它帮助每台计算机在发送或中转信件时找到下一个目的地，即“发信”协议，与其相对应的 POP3 协议即“收信”协议。SMTP 主要命令如下表所示：

命令	作用
HELO	使用标准的 SMTP，向服务器标识用户身份。发送者能进行欺骗，但一般情况下服务器都能检测到
EHLO	使用 ESMTP，向服务器标识用户身份。发送者能进行欺骗，但一般情况下服务器都能检测到。
STARTTLS	启用 TLS
MAIL FROM	命令中指定的地址是发件人地址
SEND FROM	命令中指定的地址是发件人地址
SOML FROM	命令中指定的地址是发件人地址
SAML FROM	命令中指定的地址是发件人地址
RCPT TO	标识单个的邮件接收人；可有多个 RCPT TO；常在 MAIL 命令后面
DATA	在单个或多个 RCPT 命令后，表示所有的邮件接收人已标识，并初始化数据传输，以 CRLF.CRLF 结束
VRFY	用于验证指定的用户/邮箱是否存在；由于安全方面的原因，服务器常禁止此命令
EXPN	验证给定的邮箱列表是否存在，扩充邮箱列表，也常被禁用
HELP	查询服务器支持什么命令
NOOP	无操作，服务器响应 250 OK

RSET	重置会话，当前传输被取消，服务器响应 250 OK
QUIT	结束会话

下面我们以 Kerio Connect 邮件服务器为例使用上述命令进行简单操作：



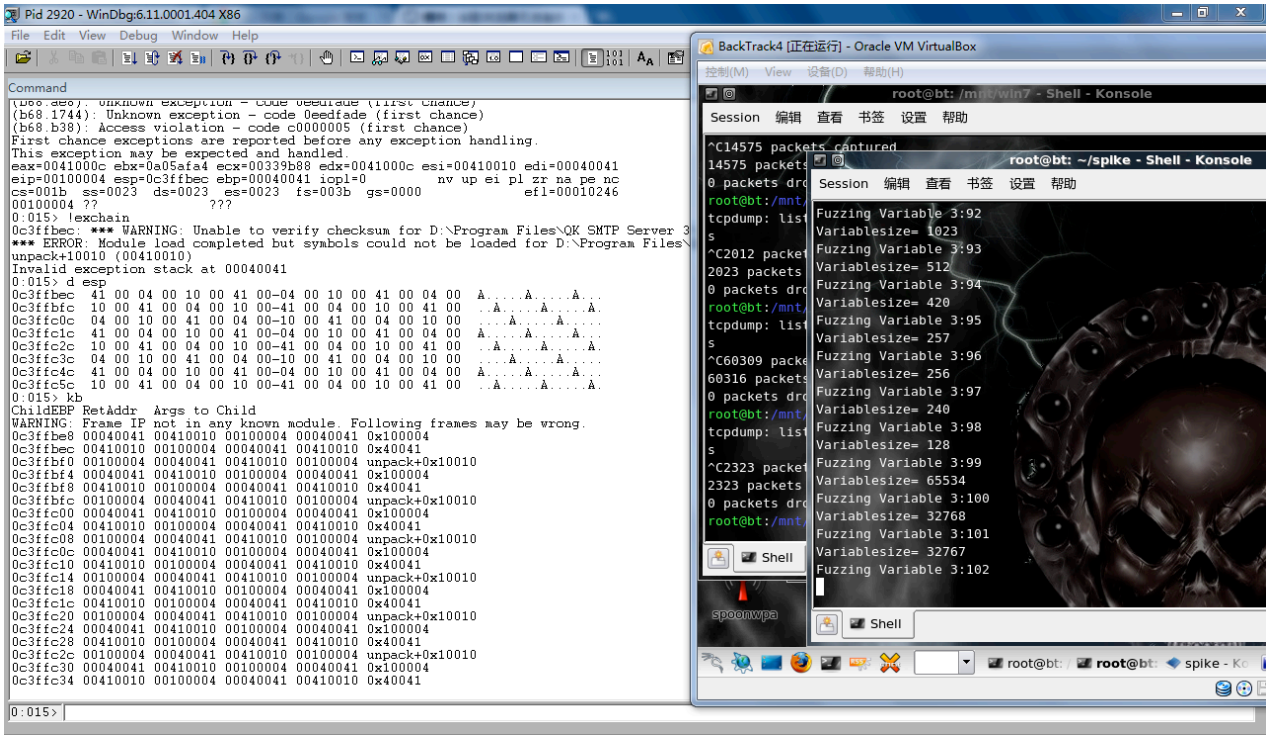
针对 SMTP 的测试与 FTP 有点类似，主要就是针对这些命令进行 Fuzzing 测试，下面就是此次用于测试的 spk 脚本：

```
s_readline();
s_string("EHLO ");
s_string_variable("localhost");
s_string("\r\n");
s_string("HELO ");
s_string_variable("localhost");
s_string("\r\n");
s_string("AUTH ");
s_string_variable("login");
s_string("\r\n");
s_string("VRFY ");
s_string_variable("riusksk");
s_string("@");
s_string_variable("qq.com");
s_string("\r\n");
s_string("EXPN ");
s_string_variable("riusksk");
```

```
s_string("\r\n");
s_string("SEND FROM: ");
s_string_variable("riusksk@qq.com");
s_string("\r\n");
s_string("SOML FROM: ");
s_string_variable("riusksk@qq.com");
s_string("\r\n");
s_string("SAML FROM: ");
s_string_variable("riusksk@qq.com");
s_string("\r\n");
s_string("STARTTLS ");
s_string_variable("AAAA");
s_string("\r\n");
s_string("HELP ");
s_string_variable("AAAA");
s_string("\r\n");
s_string("NOOP ");
s_string_variable("AAAA");
s_string("\r\n");
s_string("RSET ");
s_string_variable("AAAA");
s_string("\r\n");
s_string("MAIL FROM: ");
s_string_variable("riusksk");
s_string("@");
s_string_variable("qq");
s_string(".");
s_string_variable("com");
s_string("\r\n");
s_string("RCPT TO: ");
s_string_variable("riusksk");
s_string("@");
s_string_variable("localhost");
s_string(".");
s_string("com");
s_string("\r\n");
s_string("DATA\r\n");
s_string_variable("Message-ID");
s_string(":");
s_string_variable("123");
s_string("\r\n");
s_string("from:");
s_string_variable("riusksk@qq.com");
s_string("\r\n");
```

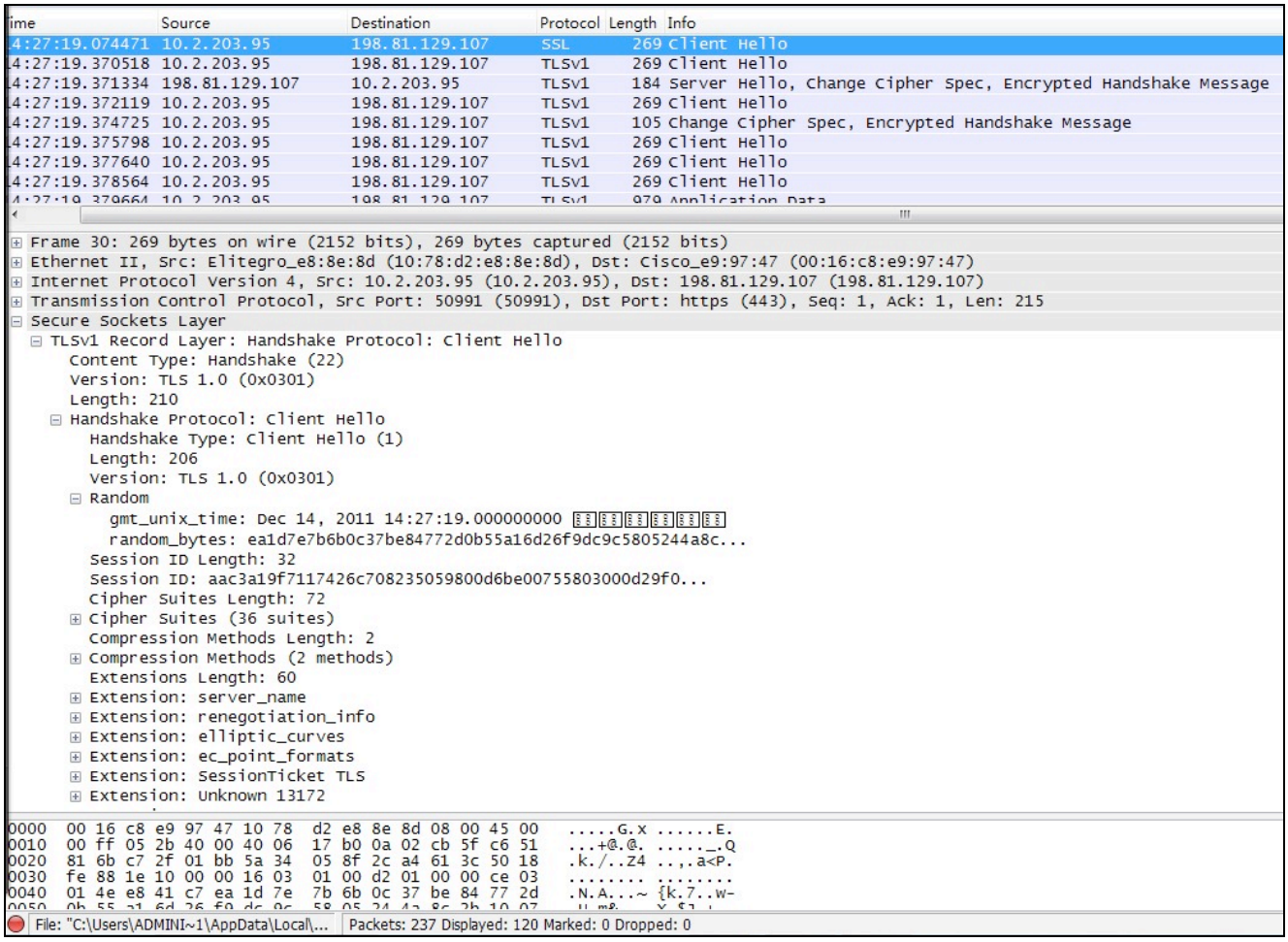
```
s_string("to:");
s_string_variable("riusksk@localhost");
s_string("\r\n");
s_string("subject:");
s_string_variable("test");
s_string("\r\n");
s_string_variable("This is a email test");
s_string("\r\n.\r\n");
```

下面是针对 QK SMTP Server V3.01 邮件服务器的 Fuzzing 结果，该软件存在栈溢出漏洞：

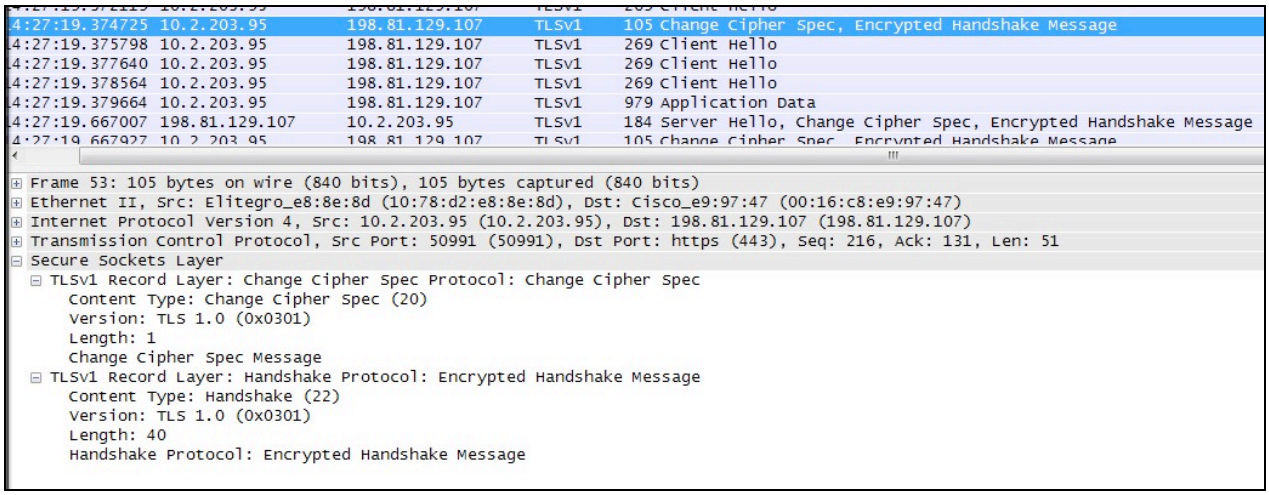


3、SSL 协议测试

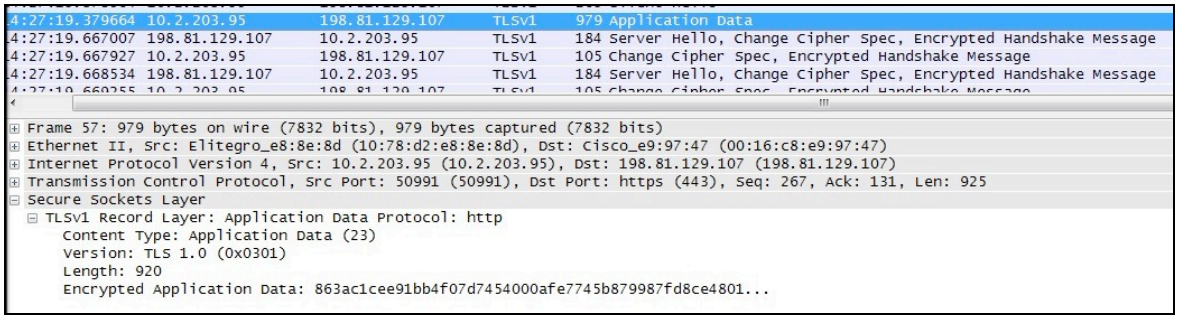
安全套接层协议（SSL，Security Socket Layer）是网景（Netscape）公司提出的基于 WEB 应用的安全协议，它包括：服务器认证、客户认证（可选）、SSL 链路上的数据完整性和 SSL 链路上的数据保密性。在 Fuzzing 之前我们需要先对 SSL 数据包进行分析，可以登陆一个使用 SSL 协议的网站，比如银行、网店之类的站点，然后用 WireShark 抓包。如下所示：



```
ime          Source          Destination     Protocol Length Info
4:27:19.074471 10.2.203.95     198.81.129.107  SSL      269 Client Hello
4:27:19.370518 10.2.203.95     198.81.129.107  TLSv1    269 Client Hello
4:27:19.371334 198.81.129.107  10.2.203.95     TLSv1    184 Server Hello, Change Cipher Spec, Encrypted Handshake Message
4:27:19.372119 10.2.203.95     198.81.129.107  TLSv1    269 Client Hello
4:27:19.374725 10.2.203.95     198.81.129.107  TLSv1    105 Change Cipher Spec, Encrypted Handshake Message
4:27:19.375798 10.2.203.95     198.81.129.107  TLSv1    269 Client Hello
4:27:19.377640 10.2.203.95     198.81.129.107  TLSv1    269 Client Hello
4:27:19.378564 10.2.203.95     198.81.129.107  TLSv1    269 Client Hello
4:27:19.379664 10.2.203.95     198.81.129.107  TLSv1    979 Application Data

⊞ Frame 30: 269 bytes on wire (2152 bits), 269 bytes captured (2152 bits)
⊞ Ethernet II, Src: Elitegro_e8:8e:8d (10:78:d2:e8:8e:8d), Dst: Cisco_e9:97:47 (00:16:c8:e9:97:47)
⊞ Internet Protocol Version 4, Src: 10.2.203.95 (10.2.203.95), Dst: 198.81.129.107 (198.81.129.107)
⊞ Transmission Control Protocol, Src Port: 50991 (50991), Dst Port: https (443), Seq: 1, Ack: 1, Len: 215
⊟ Secure Sockets Layer
  ⊟ TLSv1 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 210
    ⊟ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 206
        Version: TLS 1.0 (0x0301)
      ⊟ Random
          gmt_unix_time: Dec 14, 2011 14:27:19.000000000 [redacted]
          random_bytes: ea1d7e7b6b0c37be84772d0b55a16d26f9dc9c5805244a8c...
        Session ID Length: 32
        Session ID: aac3a19f7117426c708235059800d6be00755803000d29f0...
        Cipher Suites Length: 72
      ⊞ Cipher Suites (36 suites)
        Compression Methods Length: 2
      ⊞ Compression Methods (2 methods)
        Extensions Length: 60
      ⊞ Extension: server_name
      ⊞ Extension: renegotiation_info
      ⊞ Extension: elliptic_curves
      ⊞ Extension: ec_point_formats
      ⊞ Extension: SessionTicket TLS
      ⊞ Extension: Unknown 13172

0000  00 16 c8 e9 97 47 10 78  d2 e8 8e 8d 08 00 45 00   .....G.x ......E.
0010  00 ff 05 2b 40 00 40 06  17 b0 0a 02 cb 5f c6 51   ...+@.@. ....._.Q
0020  81 6b c7 2f 01 bb 5a 34  05 8f 2c a4 61 3c 50 18   .k./..Z4 ..,.a<P.
0030  fe 88 1e 10 00 00 16 03  01 00 d2 01 00 00 ce 03   ........ ........
0040  01 4e e8 41 c7 ea 1d 7e  7b 6b 0c 37 be 84 77 2d   .N.A...~ {k.7..w-
0050  0b 55 a1 6d 26 f0 dc 9c  58 05 24 4a 8c 2b 10 07   .U.m&... X.$J.+..

File: "C:\Users\ADMINI~1\AppData\Local\... Packets: 237 Displayed: 120 Marked: 0 Dropped: 0
```



```
4:27:19.374725 10.2.203.95     198.81.129.107  TLSv1    105 Change Cipher Spec, Encrypted Handshake Message
4:27:19.375798 10.2.203.95     198.81.129.107  TLSv1    269 Client Hello
4:27:19.377640 10.2.203.95     198.81.129.107  TLSv1    269 Client Hello
4:27:19.378564 10.2.203.95     198.81.129.107  TLSv1    269 Client Hello
4:27:19.379664 10.2.203.95     198.81.129.107  TLSv1    979 Application Data
4:27:19.667007 198.81.129.107  10.2.203.95     TLSv1    184 Server Hello, Change Cipher Spec, Encrypted Handshake Message

⊞ Frame 53: 105 bytes on wire (840 bits), 105 bytes captured (840 bits)
⊞ Ethernet II, Src: Elitegro_e8:8e:8d (10:78:d2:e8:8e:8d), Dst: Cisco_e9:97:47 (00:16:c8:e9:97:47)
⊞ Internet Protocol Version 4, Src: 10.2.203.95 (10.2.203.95), Dst: 198.81.129.107 (198.81.129.107)
⊞ Transmission Control Protocol, Src Port: 50991 (50991), Dst Port: https (443), Seq: 216, Ack: 131, Len: 51
⊟ Secure Sockets Layer
  ⊟ TLSv1 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.0 (0x0301)
      Length: 1
      Change Cipher Spec Message
  ⊟ TLSv1 Record Layer: Handshake Protocol: Encrypted Handshake Message
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 40
      Handshake Protocol: Encrypted Handshake Message
```



```
4:27:19.379664 10.2.203.95     198.81.129.107  TLSv1    979 Application Data
4:27:19.667007 198.81.129.107  10.2.203.95     TLSv1    184 Server Hello, Change Cipher Spec, Encrypted Handshake Message
4:27:19.667927 10.2.203.95     198.81.129.107  TLSv1    105 Change Cipher Spec, Encrypted Handshake Message
4:27:19.668534 198.81.129.107  10.2.203.95     TLSv1    184 Server Hello, Change Cipher Spec, Encrypted Handshake Message

⊞ Frame 57: 979 bytes on wire (7832 bits), 979 bytes captured (7832 bits)
⊞ Ethernet II, Src: Elitegro_e8:8e:8d (10:78:d2:e8:8e:8d), Dst: Cisco_e9:97:47 (00:16:c8:e9:97:47)
⊞ Internet Protocol Version 4, Src: 10.2.203.95 (10.2.203.95), Dst: 198.81.129.107 (198.81.129.107)
⊞ Transmission Control Protocol, Src Port: 50991 (50991), Dst Port: https (443), Seq: 267, Ack: 131, Len: 925
⊟ Secure Sockets Layer
  ⊟ TLSv1 Record Layer: Application Data Protocol: http
      Content Type: Application Data (23)
      Version: TLS 1.0 (0x0301)
      Length: 920
      Encrypted Application Data: 863ac1cee91bb4f07d7454000afe7745b879987fd8ce4801...
```

WireShark 已经帮你分析出各个字段的含义了，现在我们针对它编写 spk 脚本：

```
s_readline();
s_binary("22");          // Content Type
s_binary("03 01");      // Version
s_binary_block_size_halfword_bigendian_variable("RecordLength");
s_block_start("RecordLength");
s_binary("01");          // Handshake Type
s_binary("00");          // pad
s_binary_block_size_halfword_bigendian_variable("HandshakeLength");
s_block_start("HandshakeLength");
s_string_variable("03 01"); // Version
s_string_variable("4e e8 41 c7");              // gmt_unix_time
s_string_variable("random_bytes");
s_binary_block_size_halfword_bigendian_variable("SessionIDLength");             // Session ID length
s_block_start("SessionIDLength");
s_string_variable("12345678901234567890123456789012");
s_block_end("SessionIDLength");
s_binary_block_size_halfword_bigendian_variable("CipherSuitesLength");          // Cipher Suites Length
s_block_start("CipherSuitesLength");
s_string_variable("c0 0a c0 14");       // Cipher Suites
s_block_end("CipherSuitesLength");
s_binary("02 01 00 00 3c");       // Compression Method
s_binary("00 00");      // Type:server_name
s_binary_block_size_halfword_bigendian_variable("ServerNameLength");
s_block_start("ServerNameLength");
s_binary("00 0e 00 00 0b");
s_string_variable("www.test.com");
s_block_end("ServerNameLength");
s_string_variable("abc");
s_block_end("HandshakeLength");
s_block_end("RecordLength");

s_binary("14");          // Change Cipher Spec
s_binary("03 01");      // Version
s_binary_block_size_halfword_bigendian_variable("CipherLength");
s_block_start("CipherLength");
s_binary("01");
s_string_variable("");
s_block_end("CipherLength");
s_binary("16");          // Encrypted Handshake
s_binary("03 01");      // Version
s_binary_block_size_halfword_bigendian_variable("EncryLength");
```

```
s_block_start("EncryLength");
s_string_variable("cdee11d1");
s_block_end("EncryLength");

s_int_variable(0x17,3);      // Application Data - onebyte
s_int_variable(0x103,5);     // Version – BinaryBigendianHalfWord
s_int_variable(0x0,3);       // Length
s_int_variable(0xc8,3);
s_string_variable("riusksk");       // Data
```

编写 Fuzzer 时无需完全按照协议格式来编写，只要大体一致即可，其它的交由 Fuzzer 处理。

4、HTTP 协议测试

超文本传输协议（HTTP，HyperText Transfer Protocol)是一个客户端和服务器端请求和应答的标准（TCP）。客户端是终端用户，服务器端是网站。这里以 Kolibri WebServer V2.0 为例，通过抓包得到下面这个字段：

```
HTTP Headers
http://localhost:8080/index.htm

GET /index.htm HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:8.0) Gecko/20100101 Firefox/8.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-cn,zh;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: GB2312,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
```

下面构造 spk 脚本：

```
s_string("GET ");
s_string ("/");
s_string_variable("index.htm ");
s_string_variable("HTTP");
s_string("/1.1\r\n");
s_string("Host:");
s_string_variable("localhost");
s_string(":");
s_string_variable("8080");
s_string("\r\n");
s_string("User-Agent:");
s_string_variable("Mozilla/5.0(Windows NT 6.1;rv:8.0)Gecko/20100101 Firefox/8.0");
s_string("\r\n");
s_string("Accept:");
s_string_variable("text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8");
```

```
s_string("\r\n");
s_string("Accept-Language: ");
s_string_variable("zh-cn,zh;q=0.5");
s_string("\r\n");
s_string("Accept-Encoding: ");
s_string_variable("gzip, deflate");
s_string("\r\n");
s_string("Accept-Charset: ");
s_string_variable("GB2312,utf-8;q=0.7,*;q=0.7");
s_string("Proxy-Connection:");
s_string_variable("keep-alive");
s_string("\r\n");
```

测试结果：