

中南大學

CENTRAL SOUTH UNIVERSITY

《网络安全》 课外实验报告

学生姓名 蒋殿臣

班级学号 0906140112

指导教师 王伟平

设计时间 2016 年 12 月

心脏滴血实验报告

1.实验目的

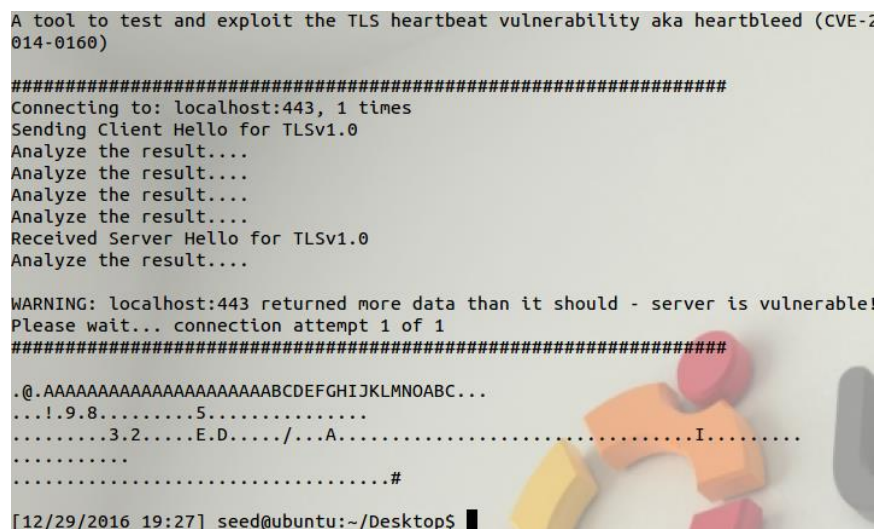
分析 heartbeat 协议具体内容，模拟演示心脏滴血漏洞的攻击流程，分析漏洞原理，并探究发现过程。

2.实验环境

Seedlab 专用虚拟机环境两台(其中包含含有心脏滴血漏洞的 Web 应用程序)，心脏滴血 exploit。

3.实验任务

1> 进行心脏滴血攻击



```
A tool to test and exploit the TLS heartbeat vulnerability aka heartbleed (CVE-2014-0160)

#####
Connecting to: localhost:443, 1 times
Sending Client Hello for TLSv1.0
Analyze the result....
Analyze the result....
Analyze the result....
Analyze the result....
Received Server Hello for TLSv1.0
Analyze the result....

WARNING: localhost:443 returned more data than it should - server is vulnerable!
Please wait... connection attempt 1 of 1
#####

.@.AAAAAAAAAAAAAAAAAAAAABCDEFGHJKLMNOABC...
...!.9.8.....5.....
.....3.2.....E.D..../.A.....I.....
.....
.....#

[12/29/2016 19:27] seed@ubuntu:~/Desktop$
```

2> 心脏滴血原因探究

先解释心跳功能，它是指客户端没间隔一段时间就周期性地向服务器发送一段简短的数据包，以表示自己任然在线，应为这种周期性地请求，才形象的比喻为心跳。

请看 ssl/dl_both.c, [漏洞的补丁](#)从这行语句开始:

```
#!/cpp
int dtls1_process_heartbeat(SSL *s)
{
```

```

unsigned char *p = &s->s3->rrec.data[0], *pl;
unsigned short hbtype;
unsigned int payload;
unsigned int padding = 16; /* Use minimum padding */

```

一上来我们就拿到了一个指向一条 SSLv3 记录中数据的指针。结构体 SSL3_RECORD 的定义如下(译者注: 结构体 SSL3_RECORD 不是 SSLv3 记录的实际存储格式。一条 SSLv3 记录所遵循的存储格式请参见下文分析):

```

#ifndef __cplusplus
typedef struct ssl3_record_st
{
    int type; /* type of record */
    unsigned int length; /* How many bytes available */
    unsigned int off; /* read/write offset into 'buf' */
    unsigned char *data; /* pointer to the record data */
    unsigned char *input; /* where the decode bytes are */
    unsigned char *comp; /* only used with decompression - malloc()ed */
    unsigned long epoch; /* epoch number, needed by DTLS1 */
    unsigned char seq_num[8]; /* sequence number, needed by DTLS1 */
} SSL3_RECORD;
#endif

```

每条 SSLv3 记录中包含一个类型域 (type)、一个长度域 (length) 和一个指向记录数据的指针 (data)。我们回头去看 dtls1_process_heartbeat:

```

#ifndef __cplusplus
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;

```

SSLv3 记录的第一个字节标明了心跳包的类型。宏 n2s 从指针 p 指向的数组中取出前两个字节, 并把它存入变量 payload 中——这实际上是心跳包载荷的长度域 (length)。注意程序并没有检查这条 SSLv3 记录的实际长度。变量 pl 则指向由访问者提供的心跳包数据。

这个函数的后面进行了以下工作:

```

#ifndef __cplusplus
unsigned char *buffer, *bp;
int r;

/* Allocate memory for the response, size is 1 byte
 * message type, plus 2 bytes payload length, plus
 * payload, plus padding
 */

```

```
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;
```

所以程序将分配一段由访问者指定大小的内存区域，这段内存区域最大为 $(65535 + 1 + 2 + 16)$ 个字节。变量 `bp` 是用来访问这段内存区域的指针。

```
#!/cpp
/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
```

宏 `s2n` 与宏 `n2s` 干的事情正好相反：`s2n` 读入一个 16 bit 长的值，然后将它存成双字节值，所以 `s2n` 会将与请求的心跳包载荷长度相同的长度值存入变量 `payload`。然后程序从 `pl` 处开始复制 `payload` 个字节到新分配的 `bp` 数组中——`pl` 指向了用户提供的心跳包数据。最后，程序将所有数据发回给用户。那么 Bug 在哪里呢？用户可以控制变量 `payload` 和 `pl`

如果用户并没有在心跳包中提供足够多的数据，会导致什么问题？比如 `pl` 指向的数据实际上只有一个字节，那么 `memcpy` 会把这条 SSLv3 记录之后的数据——无论那些数据是什么——都复制出来。

3> 漏洞修补

修复代码中最重要的一部分如下：

```
#!/cpp
/* Read type and payload length first */
if (1 + 2 + 16 > s->s3->rrec.length)
    return 0; /* silently discard */
hbtype = *p++;
n2s(p, payload);
if (1 + 2 + payload + 16 > s->s3->rrec.length)
    return 0; /* silently discard per RFC 6520 sec. 4 */
pl = p;
}
```

这段代码干了两件事情：首先第一行语句抛弃了长度为 0 的心跳包，然后第二步检查确保了心跳包足够长。就这么简单。