

网络安全实验报告

Seed 实验

学生姓名 李煜东

指导教师 王伟平

学 院 信息科学与工程学院

专业班级 信安 1402

学 号 0906140213

二〇一六年十二月三十日

实验一 本地DNS欺骗

一、问题描述

DNS (Domain Name System, 域名系统), 因特网上作为域名和IP地址相互映射的一个分布式数据库, 能够使用户更方便的访问互联网, 而不用去记住能够被机器直接读取的IP数串。通过主机名, 最终得到该主机名对应的IP地址的过程叫做域名解析(或主机名解析)。DNS协议运行在UDP协议之上, 使用端口号53。主机名到IP地址的映射有两种方式:

<http://baike.baidu.com/pic/dns/427444/235787/3853ad1b82883b7c8718bfe4?fr=lemma&ct=cover>

- 1) 静态映射, 每台设备上配置主机到 IP 地址的映射, 各设备独立维护自己的映射表, 而且只供本设备使用;
- 2) 动态映射, 建立一套域名解析系统 (DNS), 只在专门的 DNS 服务器上配置主机到 IP 地址的映射, 网络上需要使用主机名通信的设备, 首先需要到 DNS 服务器查询主机所对应的 IP 地址。

通过主机名, 最终得到该主机名对应的 IP 地址的过程叫做域名解析(或主机名解析)。在解析域名时, 可以首先采用静态域名解析的方法, 如果静态域名解析不成功, 再采用动态域名解析的方法。可以将一些常用的域名放入静态域名解析表中, 这样可以大大提高域名解析效率。

一般来说, 浏览器都是通过将两种方式结合的办法来提高用户体验的, 所以DNS攻击其实可以分为两种, 一种是针对静态映射的攻击, 即直接修改用户本地的hosts文件, 这样在浏览器解析地址的时候就不会查询DNS服务器而是直接根据文件中存取的对应的IP地址来进行访问, 达到了攻击的目的。而另外一种方式则是直接攻击用户机器或者服务器来达到目的, 主要是通过发送假的response包来进行欺骗。

所以在日常生活中, 如果可以冒充域名服务器, 然后把查询的IP地址设为攻击者的IP地址, 这样的话, 用户上网就只能看到攻击者的主页, 而不是用户想要取得的网站的主页了, 这就是DNS欺骗的基本原理。DNS欺骗其实并

不是真的“黑掉”了对方的网站，而是冒名顶替、招摇撞骗罢了。

现在的Internet上存在的DNS服务器有绝大多数都是用bind来架设的，使用的bind版本主要为bind 4.9.5+P1以前版本和bind 8.2.2-P5以前版本。这些bind有个共同的特点，就是BIND会缓存(Cache)所有已经查询过的结果，这个问题就引起了下面的问题的存在：在DNS的缓存还没有过期之前，如果在DNS的缓存中已经存在的记录，一旦有客户查询，DNS服务器将会直接返回缓存中的记录。当然，还有更多可能引发的问题，在后面将会继续描述。

二、实验概述

1. 环境配置

为了简化实验环境我们将服务器，攻击者，受攻击者的计算机都放在一台物理机上，但是用不同的虚拟机。本实验中用的网站可以是任意的。我们的配置是基于Ubuntu的。你可以从图1看出，我们建立的三台虚拟机在同一个局域网中，用户的IP地址为192.168.0.100，DNS服务器的IP地址为192.168.0.10，攻击者的IP为192.168.0.200

a. 配置DNS服务器

DNS服务器使用的是bind9软件，在ubuntu14.04上已经下载好了。

b. 创建配置文件

DNS服务器需要读取/etc/bind/named.conf文件来启动DNS服务，而named.conf.options就是一个被包含的文件，而我们需要将这句话加入到named.conf.options文件中：

```
/var/cache/bind/dump.db
```

这是为了让DNS服务器使用dump.db作为它的缓存。

c. 创建zone文件

把ZONE 文件拿出来简单说明一下。ZONE 文件是DNS 上保存域名配置的文件，对

BIND 来说一个域名对应一个ZONE 文件，现以abc.com 的ZONE 文件为例展开。

(罗嗦一句，ZONE 存在于权威DNS 上)

=====+=====+=====+=====

=====

```
$TTL 6h //第1 行
$ORIGIN abc.com. //第2 行
@ 3600 IN SOA ns1.ddd.com. root.ddd.com. ( //第3 行
929142851 ; Serial //第4 行
1800 ; Refresh //第5 行
600 ; Retry //第6 行
2w ; Expire //第7 行
300 ; Minimum //第8 行)
@ 2d IN NS ns1.ddd.com. //第9 行
@ 2d IN NS ns2.ddd.com. //第10 行
@ 2d IN NS ns3.ddd.com. //第11 行
@ 3600 IN A 120.172.234.27 //第12 行
a 3600 IN A 120.172.234.27 //第13 行
b 3600 IN CNAME a.abc.com. //第14 行
@ 3600 IN MX a.abc.com. //第15 行
@ 3600 IN TXT "TXT" //第16 行
```

=====+=====+=====+=====

=====

第1 行，这行内容给出了该域名(abc.com)各种记录的默认TTL 值，这里为6 小时。即如

果该域名的记录没有特别定义TTL，则默认TTL 为有效值。

第2 行，这行内容标识出该ZONE 文件是隶属那个域名的，这里为abc.com。

第3 行，从这行开始到第8 行为该域名的SOA 记录部分，这里的@代表域名本身。

ns1.ddd.com 表示该域名的主权威DNS。root.ddd.com 表示该主权威DNS 管理员邮箱，等价于root@ddd.com。

第4 行, Serial 部分, 这部分用来标记ZONE 文件更新, 如果发生更新则Serial 要单增, 否则MASTER 不会通知SLAVE 进行更新。

第5 行, Refresh 部分, 这个标记SLAVE 服务器多长时间主动(忽略MASTER 的更新通知)向MASTER 复核Serial 是否有变, 如有变则更新之。

第6 行, Retry 部分, 如Refresh 不能完成, 重试的时间间隔。

第7 行, Expire 部分, 如SLAVE 无法与MASTER 取得联系, SLAVE 继续提供DNS 服务的时间, 这里为2W(两周时间)。Expire 时间到期后SLAVE 仍然无法联系MASTER 则停止工作, 拒绝继续提供服务。Expire 的实际意义在于它决定了MASTER 服务器的最长下线时间(如MASTER 迁移, DOWN 机等)。

第8 行, Minimum 部分, 这个部分定义了DNS 对否定回答(NXDOMAIN 即访问的记录在权威DNS 上不存在)的缓存时间。

第9-11 行, 定义了该域名的3 个权威DNS 服务器。通常NS 记录的TTL 大些为宜, 这里为2天。设置过小只会增加服务器无谓的负担, 同时解析稳定性会受影响。

第12-16 行, 比较简单, 是两个A, CNAME, MX 记录, 不再讨论了。

附录FAQ

SOA 记录: 权威记录从这里开始, 它定义了3-8 行这些重要的参数。

A 记录: 域名到IP 之间的关联。

CNAME 记录: 让张三住到李四家里, 这时张三李四是同一个地址。

MX 记录: 定义了发往XXX@ABC.COM 邮箱的邮件服务器地址。

TXT 记录: 这个记录的内容是文本格式如163.COM 的TXT 为“v=spf1 include:spf.163.com -all”, TXT 通常用于邮件服务器来标识自己的身份避免被认为是垃圾邮件服务器

d. 配置zone文件

DNS的核心就存在于zone文件中, zone文件中各个字段的含义在上面都有解释。我们直接从Seed Project上下载对应的zone文件即可。

e. 启动DNS服务器

通过service start bind9或者service restart bind9即可启动。

f. 配置用户机

用户机的配置主要是将DNS服务器配置成192.168.0.10，这里的方式是修改/etc/resolv.conf，将nameserver修改为192.168.0.10.但是这里有个问题就是这个文件有可能被DHCP协议修改成默认的网关。所以我们还需要在System Setting里的Network中设置IPv4，把DHCP协议改为固定的DNS服务器，但是这里有个问题就是无法联网。

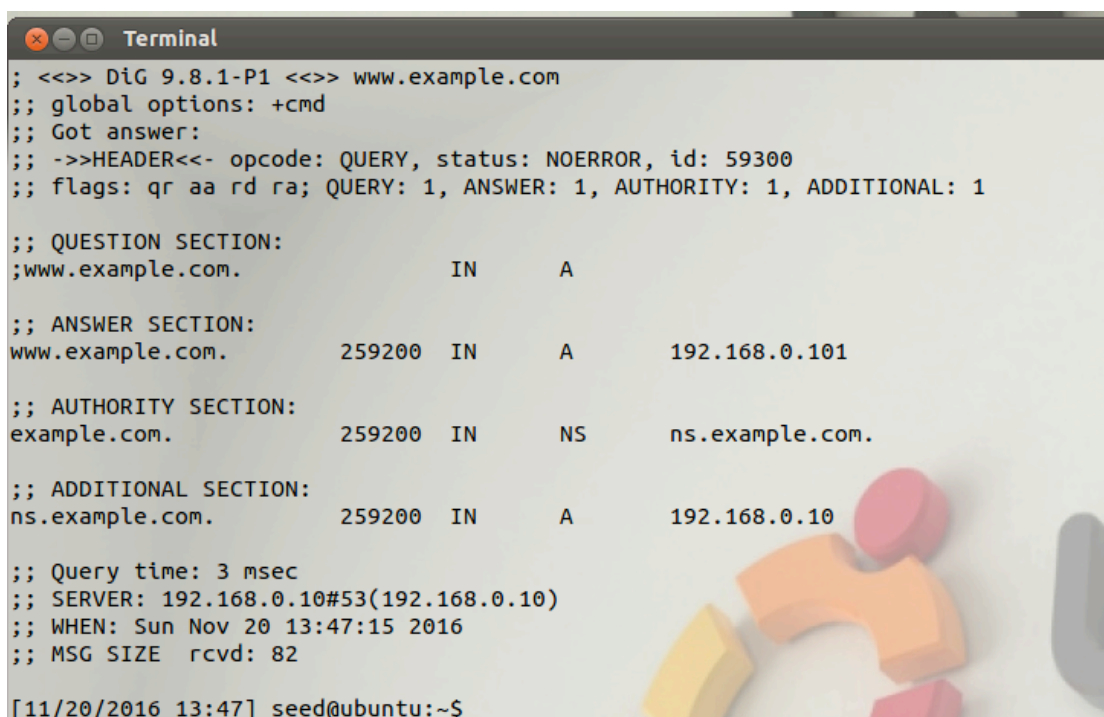
g. 攻击机只需要设置IP地址即可。

2. 任务说明

在经过前面的设置以后，要保证整个DNS查询的过程是畅通的，我们在用户机上输入以下命令进行查询：

```
dig www.example.com
```

这个时候会出现如下的画面：



```
Terminal
; <<>> DiG 9.8.1-P1 <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59300
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;www.example.com.                IN      A

;; ANSWER SECTION:
www.example.com.                259200  IN      A      192.168.0.101

;; AUTHORITY SECTION:
example.com.                    259200  IN      NS      ns.example.com.

;; ADDITIONAL SECTION:
ns.example.com.                259200  IN      A      192.168.0.10

;; Query time: 3 msec
;; SERVER: 192.168.0.10#53(192.168.0.10)
;; WHEN: Sun Nov 20 13:47:15 2016
;; MSG SIZE rcvd: 82

[11/20/2016 13:47] seed@ubuntu:~$
```

你会发现返回的IP地址为你在DNS服务器中设定好了的192.168.0.101.

接下来就要开始我们这个实验的任务了，第一个任务是本地DNS攻击，在

这个实验中我们主要是通过远程ssh已经被攻破的用户机器来修改hosts文件达到把用户引导到错误的网址上去的目的。第二个任务是通过向用户机器发送伪造的DNS回应包来达到目的，而第三个实验是通过向DNS服务器发送欺骗包来达到DNS攻击的目的。

对用户实施网域嫁接攻击的主要目的是当用户使用A的域名想要访问机器A时，将用户重定向到另一台机器B。比如，当用户试图登陆网上银行时，比如www.chase.com，如果攻击者可以把用户定向到一个和正常网站很像的恶意网站，那么用户可能会被误导而提交自己网上银行的账号和密码。

当用户在浏览器中输入www.chase.com时，用户的机器会访问一个DNS队列来找出这个域名对应的IP地址。攻击者的目标就是用个虚假的DNS回复欺骗用户机器，也就是把www.chase.com重定向到一个恶意IP地址。在实验中，我们会以www.example.com作为用户想要登陆的网站作为实例。

三、实验流程

Task 1: 修改HOSTS文件

Hosts文件是一个用于存储计算机网络中节点信息的文件，它可以将主机名映射到相应的IP地址，实现DNS的功能，它可以由计算机的用户进行控制。Hosts文件的存储位置在不同的操作系统中并不相同，甚至不同Windows版本的位置也不大一样：Windows NT/2000/XP/2003/Vista/win7：默认位置为%SystemRoot%\system32\drivers\etc\，但也可以改变。Linux下一般放在/etc/hosts下。

有很多网站不经过用户同意就将各种各样的插件安装到你的计算机中，其中有些说不定就是木马或病毒。对于这些网站我们可以利用Hosts把该网站的域名映射到错误的IP或本地计算机的IP，这样就不能访问了。在大多数系统中，约定127.0.0.1为本地计算机的IP地址，0.0.0.0是错误的IP地址。

如果，我们在Hosts中，写入以下内容：

127.0.0.1 # 要屏蔽的网站 A

0.0.0.0 # 要屏蔽的网站 B

这样，计算机解析域名 A和 B时，就解析到本机IP或错误的IP，达到了屏蔽网站A 和B的目的。

我们进行本地攻击时也是利用相同的原理，主机名和IP地址以成对的方式存放在HOSTS文件中，它用于本地查找，而不是远程的DNS查找。比如，如果用户机器里有一个这样的HOSTS文件，www.example.com将会被解释成IP地址为1.2.3.4的主机，而不会询问任何的DNS服务器。

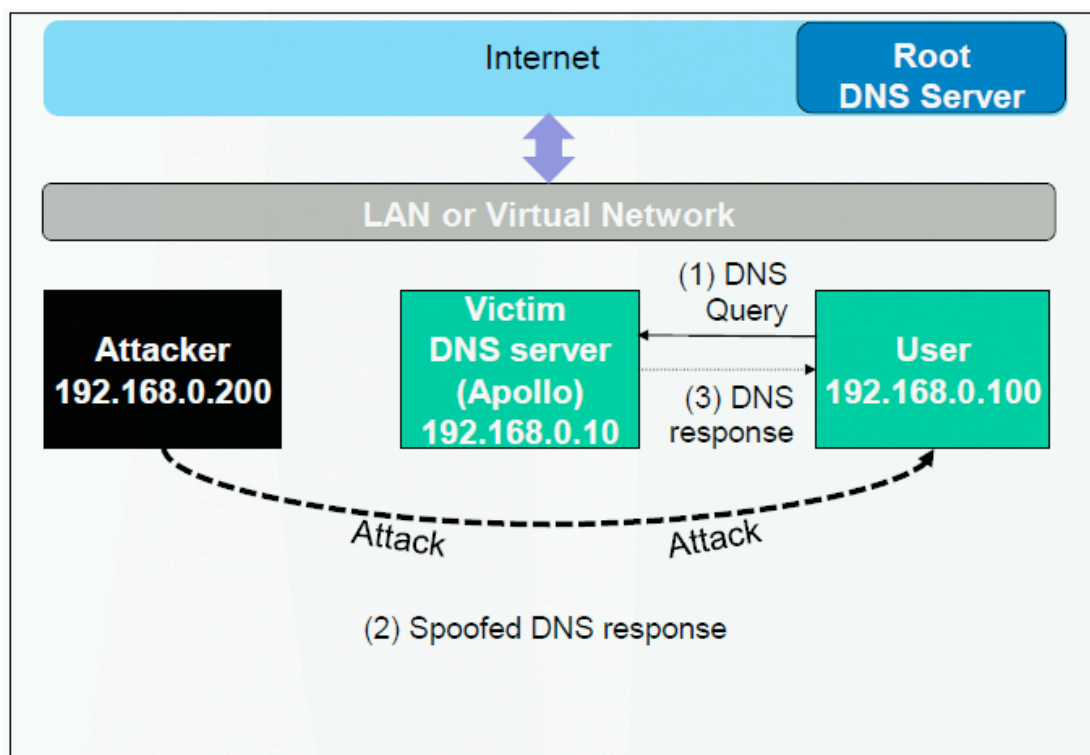
当然了，ping命令和dig命令是有所区别的，这个我们在结果分析中在进行讲述。

Task 2: 本机DNS劫持-对用户的DNS查询欺骗

在这种攻击中，受害者机器没有直接被攻破，所以攻击者不能直接改变DNS列表进程。然而，如果攻击者和受害者在同一网段，他们仍然可以造成很大的危害。当一个用户输入网站的域名时，用户机器将会询问DNS服务器来获取一个IP地址。在侦听到DNS服务器的回应后，攻击者可以创建一个假的DNS回应。假的DNS回应如果具有与真的DNS回应相同的格式将会被用户机器接受。

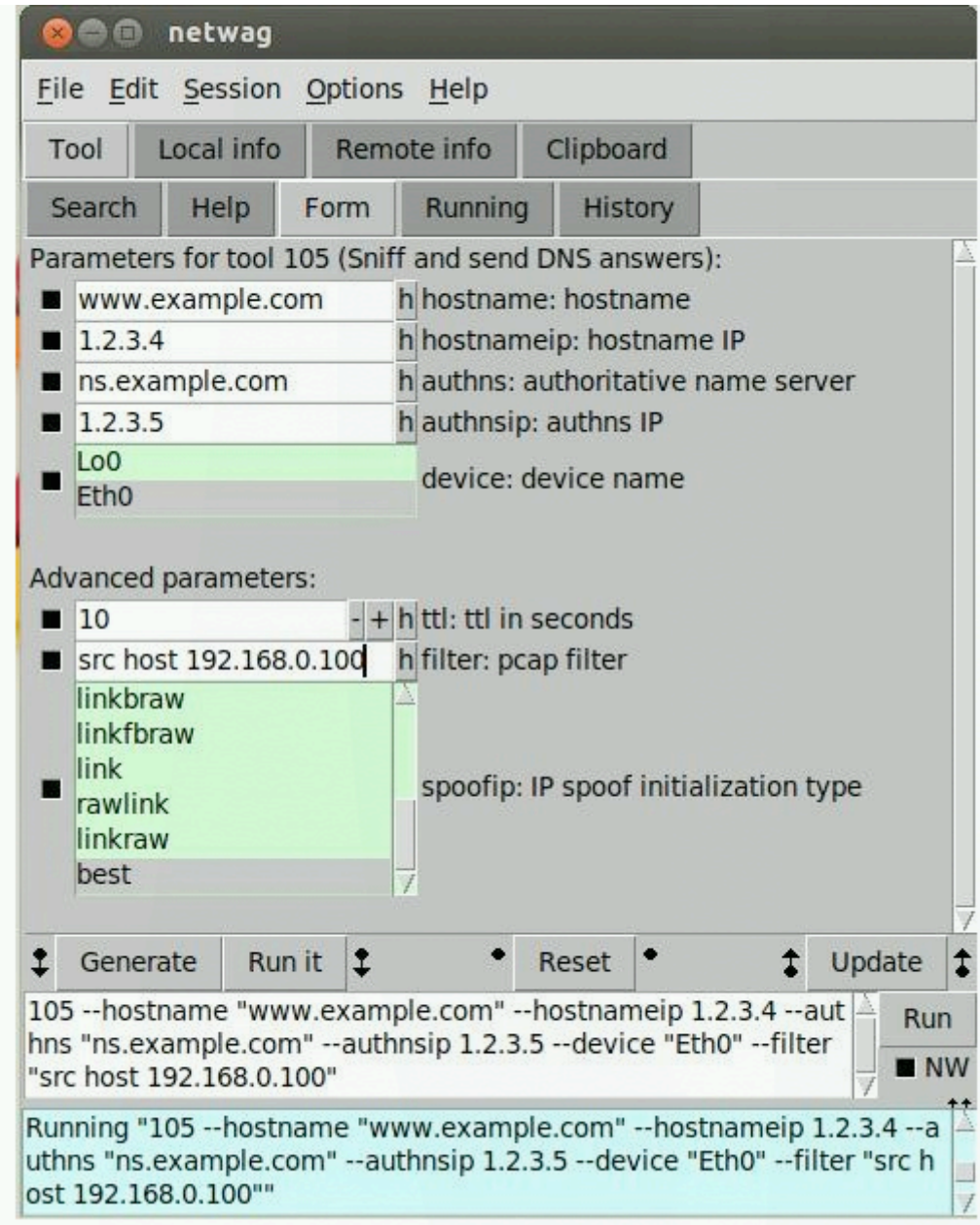
为了满足标准1-8，攻击者需要截取用户的DNS请求，并且在DNS发出回应之前给用户发送DNS请求回应。可以利用Netwox/Netwag来监听DNS服务器的请求与回应。

整个过程如下图所示：



当User向DNS Server发送DNS查询的时候，Attacker监听了这个DNS查询请求，然后在DNS Server回复正确的DNS Response之前，先回复一个伪造欺骗的DNS Response给用户，从而达到了DNS欺骗的效果。

实验中我们借用了Netwox/Netwag tool 105来进行DNS欺骗，具体的设置如下：



攻击机根据上图进行设置，然后在用户机器上再次运行dig www.example.com的命令，就可以看到如下的结果了：

```
Terminal
[11/20/2016 14:36] root@ubuntu:/home/seed# dig www.example.com

; <<>> DiG 9.8.1-P1 <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 26742
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;www.example.com.                IN      A

;; ANSWER SECTION:
www.example.com.                600     IN      A      1.2.3.4

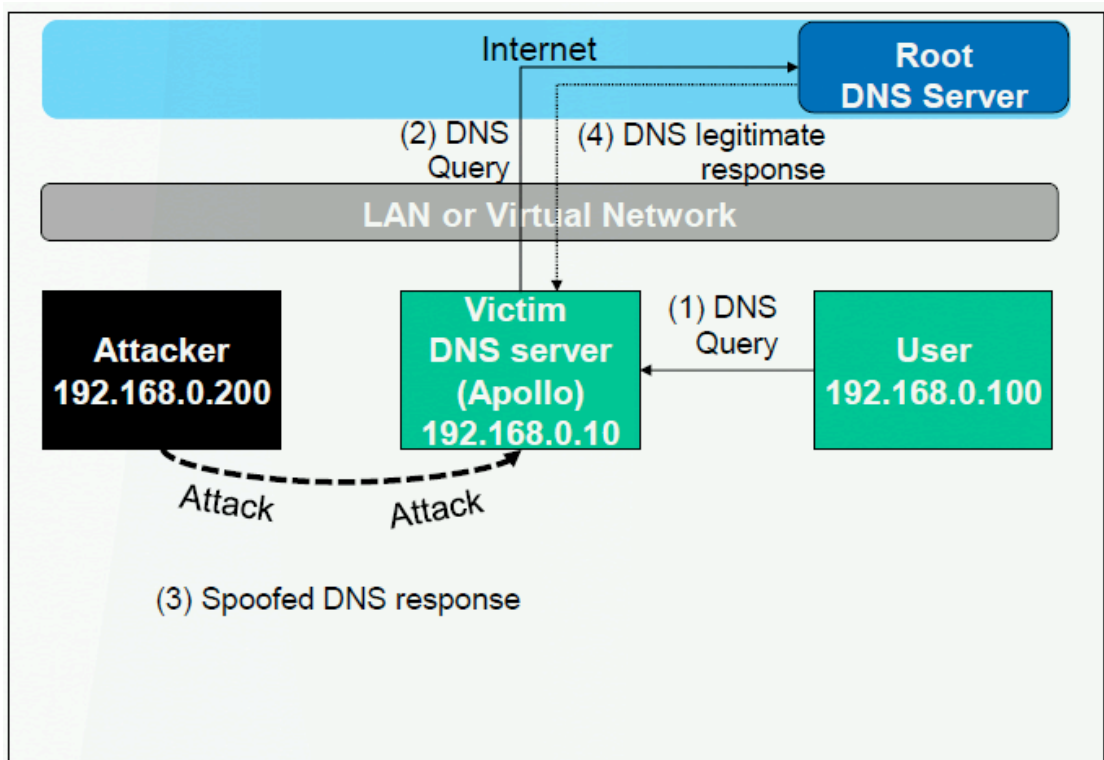
;; AUTHORITY SECTION:
ns.example.com.                 600     IN      NS      ns.example.com.

;; ADDITIONAL SECTION:
ns.example.com.                 600     IN      A      1.2.3.5

;; Query time: 6 msec
;; SERVER: 192.168.0.10#53(192.168.0.10)
;; WHEN: Sun Nov 20 14:36:22 2016
;; MSG SIZE rcvd: 88
```

Task 3: 本机DNS劫持-对服务器的DNS欺骗

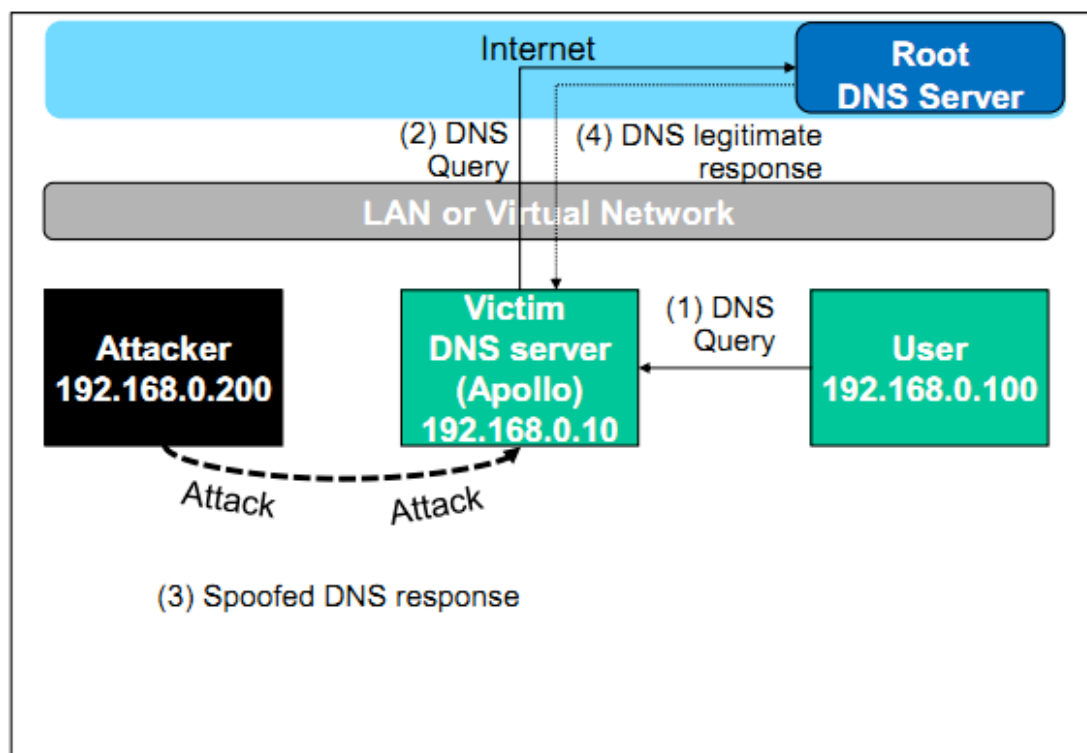
对于服务器的欺骗过程如下图所示:



在DNS的缓存还没有过期之前, 如果在DNS的缓存中已经存在的记录, 一

且有客户查询, DNS服务器将会直接返回缓存中的记录。

上述的攻击都是针对于用户机器, 为了达到保存时间长的目的, 每一次用户机器发送DNS问询时, 攻击者都需要发送一个捏造的DNS回应, 这效率会很低。现在有一个更好的方法, 即攻击DNS服务器, 而不是用户机器。



四、结果分析

Task 1: 修改HOSTS文件结果分析

用户本来的hosts文件中, www.example.com对应的是127.0.0.1, 结果如下图所示:

```
;; ANSWER SECTION:
www.example.com.      259200  IN      A       192.168.0.101

;; AUTHORITY SECTION:
example.com.          259200  IN      NS      ns.example.com.

;; ADDITIONAL SECTION:
ns.example.com.       259200  IN      A       192.168.0.10
```

ping命令的结果:

```
[11/20/2016 14:14] root@ubuntu:/home/seed# ping www.example.com
PING www.example.com (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_req=1 ttl=64 time=0.020 ms
```

修改hosts文件后的Ping结果如下所示:

```
[11/20/2016 15:00] root@ubuntu:/home/seed# ping www.example.com
PING www.example.com (1.2.3.4) 56(84) bytes of data.
```

可以看出, 修改hosts文件会影响ping命令但是不会影响dig命令, 也就是说, dig查询命令依然会问询DNS服务器, 但是Ping命令为了用户体验, 需要快速的查找到域名所对应的IP, 所以会先遍历hosts文件来进行一次查询。

Task 2: 本机DNS劫持-对用户的DNS查询欺骗结果分析

在尝试几次之后, 我们会得到如下的结果:

```
Terminal
[11/20/2016 14:36] root@ubuntu:/home/seed# dig www.example.com

; <<>> DiG 9.8.1-P1 <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26742
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;www.example.com.                IN      A

;; ANSWER SECTION:
www.example.com.                600     IN      A      1.2.3.4

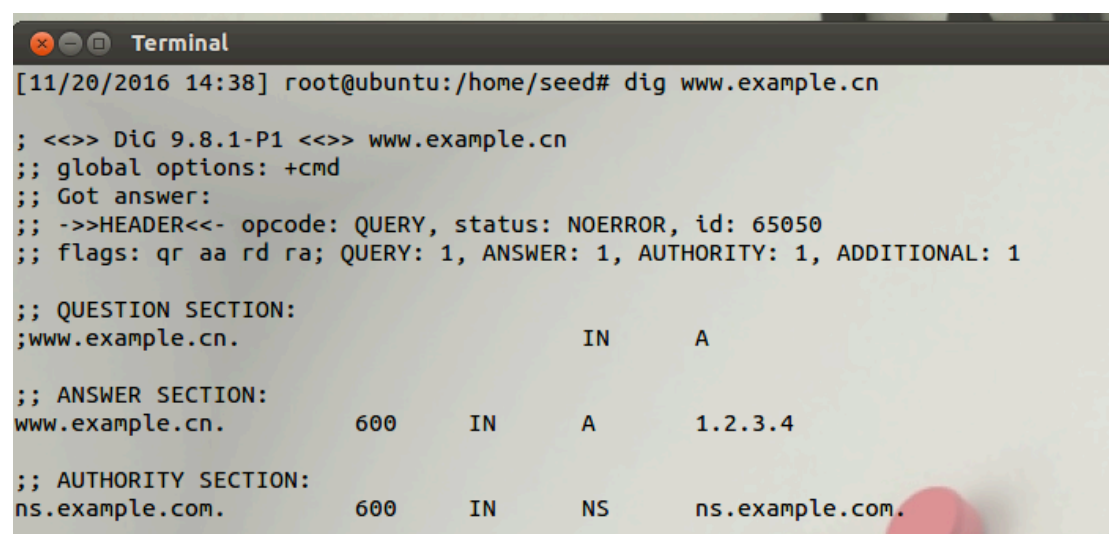
;; AUTHORITY SECTION:
ns.example.com.                 600     IN      NS      ns.example.com.

;; ADDITIONAL SECTION:
ns.example.com.                 600     IN      A      1.2.3.5

;; Query time: 6 msec
;; SERVER: 192.168.0.10#53(192.168.0.10)
;; WHEN: Sun Nov 20 14:36:22 2016
;; MSG SIZE rcvd: 88
```

但是这其中有一个问题, 因为在局域网内, 互相之间的通信速度是非常快的, 所以虽然把TTL设置的很长, 但是有的时候用户机仍然会接受到正确的DNS包, 这里还需要再改进。

Task 3: 本机DNS劫持-对服务器的DNS欺骗结果分析



```
[11/20/2016 14:38] root@ubuntu:/home/seed# dig www.example.cn

; <<>> DiG 9.8.1-P1 <<>> www.example.cn
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 65050
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;www.example.cn.                IN      A

;; ANSWER SECTION:
www.example.cn.                 600     IN      A      1.2.3.4

;; AUTHORITY SECTION:
ns.example.com.                 600     IN      NS      ns.example.com.
```

在对Netwag进行了相应的设置以后，我们达到了攻击服务器的目的。

五、调试报告

1. 原理概述

一、DNS 篡改（DNS Hijacking or DNS Redirection）

DNS 篡改通常是指直接修改根服务器中的 DNS 条目，当然这种篡改更有可能发生在域名注册商的存放客户注册配置信息的服务器中，从而导致相应的域名在全球范围内都解析错误。（有些情况下，也有可能只是在中层的 DNS 服务器作篡改，这样仅仅劫持某部份的域名解析，只会影响部分地区。）

这种攻击方式的特点如下：

全球性。因为根服务器条目被更改，在下游 DNS 缓存时间过后，被篡改条目必然会更新到所有下游服务器，从而影响到所有用户。

通常所说的“技术性”并非必需。在多数情况下，这种攻击方式在DNS攻击阶段甚至不需要涉及技术性，可以借助社会工程学等手段，攻击者通常已经拥有了足够的修改DNS条目的权限。虽然不排除黑客能够通过技术手段入侵根DNS服务器或者域名服务商，在这种情况下，黑客就可以指哪儿打哪儿无法无天了。但相信这种天下大乱的情况我们不一定能有幸见到。更可能的情况是，某个网站运营商的域名的账号密码被窃取了，黑客

拿到账号密码之后，就能像网站管理员一样冠冕堂皇地登陆域名提供商网站，修改相应的IP地址。

二、DNS 欺骗（DNS Spoofing）

一种 DNS 欺骗方式是利用漏洞，去年上半年，DNS 协议被发现存在严重漏洞，攻击者可以欺骗下游服务器，使其相信一台假冒的服务器是它的权威服务器。这样攻击者就可以通过在假冒服务器上添加虚假的 DNS 信息来欺骗下游服务器，最终欺骗客户端。在 Windows 系统中，这个漏洞已经在稍早的安全补丁发布获得中解决：

<http://www.microsoft.com/china/technet/security/bulletin/ms08-037.msp>

其他操作系统中也可以查询到相应的漏洞。还有一种 DNS欺骗思路，借助其他欺骗达到DNS欺骗的目的，例如先在目标客户端或DNS服务器的同一局域网网段作ARP欺骗，使受攻击的计算机向ARP攻击的发起者作DNS查询。但是这种攻击方式必须先攻破与目标主机同网段的另一台计算机，而互联网中的DNS服务器通常在ISP的控制之下，渗透进ISP网络的难度不小。攻击根服务器的难度就更别提了。

DNS 欺骗攻击方式的特点如下：

区域性。因为这种攻击只能攻击下游服务器，因此攻击的有效范围必然有限，只会影响到此台服务器下层的服务器和客户端。

漏洞利用。要实现 MS08-037 攻击必然需要利用 DNS 协议的漏洞（MS08-037 或者其他可能存在的漏洞）。如果 DNS 服务器上的漏洞已经修复，那么这种攻击就无法成功。

技巧性。不论利用漏洞还是利用 ARP 的设计缺陷，这类攻击方式在 DNS 攻击阶段都需要一定的技术手段。

实验二 Heart Bleed漏洞

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。

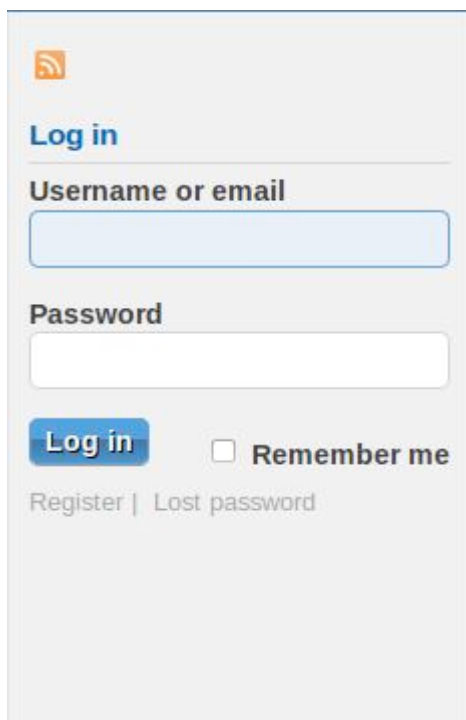
三、实验过程分析

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
da wants pages about "irl games". Unlocking
secure records with master key 5130985733433
User Karen uploaded the file

用户Meg需要这6个字母：POTATO



服务器，你还在吗？在的话
请回复“BIRD”（4个字母）



POTATO

secure connection using key "4538538374224"
User Meg wants these 6 letters: **POTATO**. User
da wants pages about "irl games". Unlocking
secure records with master key 5130985733433
User Karen uploaded the file

用户Meg需要这4个字母：BIRD



服务器，你还在吗？在的话
请回复“HAT”（300个字母）



BIRD

User Olivia from Osaka wants pages about "n
ees 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
kevinin (uploaded 5130985733433)

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=88u51nr3r2rf9c140h2r0qovc6
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
#####

..PAAAAAAAAAAAAAAAAAAAAABCDEF GHIJKLMNOPABC...
...!.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], *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：

```

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。

心得体会

本次实验中自己动手配置了 DNS 服务器和进行了 DNS 欺骗，学习到了 DNS 的工作原理和服务器的搭建方式。更加熟悉了 Linux 系统的使用。另外，也了解了 HeartBleed 漏洞的利用方式和它的危险性。总的来说，这次实验难度不大，但是英文的文档阅读起来还是有一些难度，在实验的过程中也遇到了不少问题，但是最后都一一解决了，还是从中受益匪浅的。