**实 验 报 告**



**课程名称 《网络攻击与防御技术》**

**学 院 计算机科学技术学院**

**专 业 信息安全**

**姓 名 黄 力**

**学 号 15307130275**

**开 课 时 间 2018 至 2019 学年第 1 学期**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | 实验项目  名 称 | 基于raw socket的防TCPsyn端口扫描机制 | 成绩 |  |   **一、实验目的**  （1）理解TCPsyn端口网络扫描的原理  （2）利用raw socket编程防止主机受到TCPsyn扫描，利用raw socket对局域网内的主机进行TCPsyn扫描  **二、实验内容**  （1）采用raw socket方法编程，利用TCP协议的建立原理实现防TCPsyn端口扫描，即实现主机上的所有TCP端口均开放的假象。  （2）验证所编软件，通过虚拟机本身进行自扫，扫描本机开放的TCP端口。当软件运行成功时，本机的所有TCP端口对于扫描的机器来说都是开放的（实际未开放）  （3）通过截包工具截获扫描过程，分析扫描成功/失败的原因。  **三、实验环境**  （1）PC机操作系统：macOS Mojave 10.14  （2）虚拟机操作系统（paralllels 13.1.1）：ubuntu 16.04 x86\_64  （3）开发语言：python 3.7.0  （4）工具链：端口扫描：nmap 7.01  **四、实验原理**  （1）通过向目标主机的某个端口发送TCPsyn数据报文并根据该目标主机返回报文情况初步判断端口的开放情况：  1、若目标主机返回TCPsynack报文，则该TCP端口为开放状态  2、若目标主机返回TCPrst报文，则该UDP端口为关闭状态  **五、实验步骤及结果**  （1）根据实验要求和原理编写代码，本次实验使用的开发语言为python3.7.0，用到的库主要有struct（数据包封装与拆解）、socket。  代码共包含三个主要函数：antiTcpScan、tcpSynAckBack、createTcpHeader、createIpHeader  antiTcpScan函数使用raw socket建立套接字并通过该套接字上监听所有报文，若报文为TCP报文（ip头部的protocol字段为6）且为syn报文（tcp头部的syn字段为1）则调用tcpSynAckBack函数发送自己伪造的数据包：  tcpSynAckBack函数调用createIpHeader函数与createTcpHeader函数伪造相应的IP头部与TCP头部，其中TCP头部字段置syn字段与ack字段为1，并组装报文发送回去。  createIpHeader函数构造IP头部，createTcpHeader函数构造TCP头部：其中ack确认的序号应为收到的报文的序列号加1，syn和ack字段应置为1。同时在计算校验和时应加入伪头部参与计算。计算方式在checkSum函数中。注：这3个函数的实现参考了两个博客的代码，在参考资料中已列出。  （2）使用（1）中编写的程序防止TCPsyn扫描。  先在一个终端里执行sudo python3 antiTcpScan.py命令开启防护程序  再在另一个终端里执行sudo namp -sS -p 233 -n 127.0.0.1使用TCPsyn扫描233端口  运行结果截图如下：  从上图可知，本次编写的程序未能阻挡nmap程序的TCPsyn扫描。  尝试换用其他端口（如：6666、2333）后结果仍然如上所示未能成功。  失败原因分析：  antiTcpScan.py没有抢在内核发送TCPrst报文之前发送伪造的TCPsynack报文，从截图中可以看出防护程序成功截获并筛选出了nmap发送的TCPsyn报文，并将伪造的报文成功发送出去（打印出了succeed消息），但nmap扫描结果仍然为233端口关闭，这说明它已经在收到伪造报文之前收到了内核正确发出的TCPrst报文。  **六、实验总结**  通过本次实验，我理解了TCPsyn端口扫描的原理和方式，尝试编写代码对主机进行TCPsyn扫描的防护。代码的逻辑和实现方式是正确的，但因为未能屏蔽内核正常发出的rst报文或赶在它之前发送synack报文而没有成功实现防护。本次实验的难点主要有两点：一是在于需要自己构造IP头部和TCP头部，二是网络环境的复杂性可能导致实验失败。针对第一个难点解决方式是参考网上博客学习构造IP头部和TCP头部，针对第二个难点解决方式是使用虚拟机自己扫自己，避免了复杂的网络环境。  **七、主要参考资料**  1、https://blog.csdn.net/cheng\_fangang/article/details/38709549  2、https://segmentfault.com/q/1010000012223391 |