

南京大学本科生实验报告

181860097 计算机科学与技术系 王明骁

Email: 1164151483@qq.com

开始日期: 2020年6月13日 结束日期: 2020年6月17日

lab7

实验名称

Lab 7: Firewall

实验目的

- 掌握并理解防火墙的基本原理
- 实现防火墙的各种基础功能

task1

1. 实验内容

task1为准备阶段，只需要按照实验手册上的说明，将相关文件copy到lab_7文件夹并做好命名即可

task2

1. 实验内容

task2的内容分为以下两部分：

- 根据firewall_rules.txt文件中的规则语法，建立规则，而实现的关键在于根据规则的语法，正确建立规则表，这里采取了正则表达式的方法建立，这里采取了六条正则表达式，`(permit|deny) (ip|icmp) src (\S+) dst (\S+) impair` 对应于未指定端口号的带impair的规则，`(permit|deny) (ip|icmp) src (\S+) dst (\S+) ratelimit (\S+)` 对应于未指定端口号的带速度限制的规则，`(permit|deny) (udp|tcp) src (\S+) srcport (\S+) dst (\S+) dstport (\S+) ratelimit (\S+)` 对应于指定端口号的带速度限制的规则，`(permit|deny) (ip|icmp) src (\S+) dst (\S+)` 对应于未指定端口号的，不带速度限制也不带impair的规则，`(permit|deny) (udp|tcp) src (\S+) srcport (\S+) dst (\S+) dstport (\S+) impair` 对应于指定端口号的带impair的规则，`(permit|deny) (udp|tcp) src (\S+) srcport (\S+) dst (\S+) dstport (\S+)` 对应于指定端口号的不限速度也不带impair的规则

按照正则表达式进行匹配，并将相关信息建立一个表即可，值得一提的是，由于python里的正则表达式是采取前缀匹配，因此对上述六个正则表达式进行匹配时，要先匹配带速度限制的和带impair的，否则他们会被缺省的正则表达式所匹配，从而丢失信息

- 第二部分是收到包时检查是否与规则匹配，匹配时首先匹配包的类型与规则的类型，然后在类型匹配的基础上进一步匹配源ip，目的ip，之后进一步匹配源端口和目的端口，若都匹配上了，则返回对应的规则，否则返回None

2. 实验结果

等待所有内容完成后一起测试

3. 核心代码

规则表中每一项的内容：

```
class ruleitem(object):
    def __init__(self):
        self.rulenum = 0
        self.permit = True
        self.src = IPv4Network('0.0.0.0/0',False)
        self.types = 'ip'
        self.dst = IPv4Network('0.0.0.0/0',False)
        self.srcport = -1
        self.dstport = -1
        self.ratelimit = 0
        self.tokens = 0
        self.impair = False
```

其中permit表示该规则是否允许通过，src和dst是源ip和目的ip，types表示该规则所对应的数据包的类型（ip/icmp/tcp/udp），srcport和dstport表示源端口和目的端口，ratelimit表示速率限制，tokens表示该规则的令牌，impair表示是否减损

当匹配到任何一条正则表达式时，在rules中添加对应的项，这里以其中一条正则表达式为例：

```
result = re.match(r'(\s*)(permit|deny) (\s*)(ip|icmp) (\s*)(src|dst) (\S+) (\S+) (\S+) (\S+)',line)
if result:
    rule.rulenum = 0
    if result.group(1) == 'permit':
        rule.permit = True
    else:
        rule.permit = False
    if result.group(2) == 'icmp':
        rule.types = 'icmp'
    else:
        rule.types = 'ip'
    if result.group(3) == 'any':
        rule.src = IPv4Network('0.0.0.0/0',False)
    else:
        rule.src = IPv4Network(result.group(3),False)
    if result.group(4) == 'any':
        rule.dst = IPv4Network('0.0.0.0/0',False)
    else:
        rule.dst = IPv4Network(result.group(4),False)
    rule.ratelimit = int(result.group(5))
    rule.tokens = 2*rule.ratelimit
    rule.impair = False
    return rule
```

这是针对ipv4的包和icmp的包的规则，并且包含速率限制，首先根据permit或者是deny来设定rule的permit值，当permit为true时表示允许，为false时表示阻止，然后根据icmp/ip设置rule的类型，然后填写对应的srcip和dstip，最好将速率限制的值转换为int后赋值给ratelimit，同时设置该rule的初始令牌数，最后返回rule，在main函数中将该rule添加到rules数组中

对要转发的数据包进行匹配：

```
def pktmatch(pkt,rules):
    ipv4 = pkt.get_header(IPv4)
    #debugger()
    icmp = pkt.get_header(ICMP)
    tcp = pkt.get_header(TCP)
    udp = pkt.get_header(UDP)
    pkttype = 'ip'
    if icmp is not None:
        pkttype = 'icmp'
    elif tcp is not None:
        pkttype = 'tcp'
    elif udp is not None:
        pkttype = 'udp'
    if pkttype == 'ip':
        return None

    for index in rules:
        #debugger()
        if index.types != 'ip' and index.types != pkttype:
            continue
        if ipv4.src in index.src and ipv4.dst in index.dst:
            if pkttype == 'icmp':
                return index
            elif pkttype == 'tcp':
                if (tcp.src == index.srcport or index.srcport == -1) and (tcp.dst == index.dstport or index.dstport == -1):
                    return index
            elif pkttype == 'udp':
                if (udp.src == index.srcport or index.srcport == -1) and (udp.dst == index.dstport or index.dstport == -1):
                    return index

    return None
```

在匹配之前首先过滤所有非ipv4的数据包，因此传进该函数都是ipv4的数据包，在此基础上首先判断该数据包的类型，然后扫描规则表中的每一条规则，只有规则类型为ip的或者规则类型和该数据包的类型相同的规则才进行下一步匹配，下一步匹配时，首先检查源ip和目的ip是否能对应于该规则，在此基础上再检查源端口号和目的端口号是否一致，若一致，则匹配上该规则，直接返回该规则，否则返回None

由于该规则的优先级是先写的规则优先级高，因此从数组开头开始检查的逻辑是符合要求的

task3

1. 实验内容

task3要求实现令牌桶算法，简单来说就是每隔0.5秒向桶中加入令牌，只有桶中的令牌数量足够才能发数据包，否则将数据包丢弃。具体实现时分为两个部分：加入令牌部分，发送包时检查令牌部分

2. 实验结果

等待所有内容写完后再一起测试

3. 核心代码

加入令牌部分：

```
while True:
    pkt = None
    try:
        timestamp,input_port,pkt = net.recv_packet(timeout=0.25)
    except NoPackets:
        pass
    except Shutdown:
        break

    if time.time()-pretime > 0.5:
        for index in rules:
            if index.ratelimit != 0:
                index.tokens = index.tokens + index.ratelimit/2
                if index.tokens > 2*index.ratelimit:
                    index.tokens = 2*index.ratelimit
        pretime = time.time()
```

无论是否收到包都进行判断，如果已经过了0.5秒，就向每个带有速度限制的规则中都加入令牌，如果加入的令牌数量超过最大值，则改为最大值2r

发送部分：

```
if pkt is not None:
    ip = pkt.get_header(IPv4)
    if ip is None:
        net.send_packet(portpair[input_port], pkt)
        continue

    matchrule = pktmatch(pkt, rules)
    #debugger()
    if matchrule is None:
        net.send_packet(portpair[input_port], pkt)
        continue

    if matchrule.permit == True:
        for index in rules:
            if index == matchrule:
                if index.ratelimit != 0:
                    l = len(pkt) - len(pkt.get_header(Ethernet))
                    if index.tokens >= l:
                        index.tokens = index.tokens - l
                        net.send_packet(portpair[input_port], pkt)
                        break
                if index.impair:
                    ran = randint(0,1)
                    #if ran > 0.5:
                    if ran > 2:
                        net.send_packet(portpair[input_port], pkt)
                        break
                net.send_packet(portpair[input_port], pkt)
                break
```

针对每一个包，首先检查其是否是ipv4的包，若不是则直接发，然后进行规则的匹配，若没匹配上则直接发，当匹配上时查看对应的规则的permit值是否为true，只有为true的规则才发包，发包时检查该规则是否包含速率限制，若包含速率限制，则检查是否有足够的令牌，若有，则将相应数量的令牌去除，然后发包，否则不发包，而对于没有速率限制的规则则不管

task4

1. 实验内容：

task4要求实现impair功能，这里采取的实现方式是随机丢包，设置一个初始丢包率，通过python自带的随机数库生成0到1的随机数，若随机数的值大于丢包率，则丢包，否则发包即可，而需要impair的规则，其impair值为true

2. 实验结果：

在task5中进行测试

3. 核心代码：

```

if index.impair:
    ran = randint(0,1)
    #if ran > 0.5:
    if ran > 2:
        You, 3 hours ago • Uncommitt
    net.send_packet(portpair[input_port], pkt)
break

```

若匹配到的规则的impair值为true，则表示需要进行随即丢包处理，通过设置不同的丢包率实现不同程度的impair效果

task5

1. 使用助教提供的测试文件进行测试：

```

(gven) nlu@hpc-VirtualBox:~/switchyard/lab_7$ soyard -t firewalltests.py firewall.py
16:20:33 2020/06/17 INFO Starting test scenario Firewalltests.py

Results for test scenario Firewall tests: 35 passed, 0 failed, 0 pending

Passed:
1 Packet arriving on eth0 should be permitted since it matches rule 3.
2 Packet forwarded out eth1; permitted since it matches rule 3.
3 Packet arriving on eth1 should be permitted since it matches rule 4.
4 Packet forwarded out eth0; permitted since it matches rule 4.
5 Packet arriving on eth0 should be permitted since it matches rule 5.
6 Packet forwarded out eth1; permitted since it matches rule 5.
7 Packet arriving on eth1 should be permitted since it matches rule 6.
8 Packet forwarded out eth0; permitted since it matches rule 6.
9 Packet arriving on eth0 should be permitted since it matches rule 9.
10 Packet forwarded out eth1; permitted since it matches rule 9.
11 Packet arriving on eth1 should be permitted since it matches rule 10.
12 Packet forwarded out eth0; permitted since it matches rule 10.
13 Timeout for 1s
14 Timeout for 1s
15 Timeout for 1s
16 Timeout for 1s
17 Packet arriving on eth0 should be permitted since it matches rule 7.
18 Packet forwarded out eth1; permitted since it matches rule 7.
19 Packet arriving on eth1 should be permitted since it matches rule 8.
20 Packet forwarded out eth0; permitted since it matches rule 8.
21 Packet arriving on eth0 should be permitted since it matches rule 13.
22 Packet forwarded out eth1; permitted since it matches rule 13.
23 Packet arriving on eth1 should be permitted since it matches rule 13.
24 Packet forwarded out eth0; permitted since it matches rule 13.
25 Packet arriving on eth0 should be blocked due to rate limit.
26 Packet arriving on eth0 should be blocked since it matches

```

可以看到所有的测试都通过

2. 在mininet中测试速率：

使用命令mininet> internal ping -s72 192.168.0.2

得到以下结果：

```

mininet> internal ping -s72 192.168.0.2
PING 192.168.0.2 (192.168.0.2) 72(100) bytes of data:
80 bytes from 192.168.0.2: icmp_seq=1 ttl=64 time=123 ms
80 bytes from 192.168.0.2: icmp_seq=2 ttl=64 time=165 ms
80 bytes from 192.168.0.2: icmp_seq=3 ttl=64 time=106 ms
80 bytes from 192.168.0.2: icmp_seq=4 ttl=64 time=162 ms
80 bytes from 192.168.0.2: icmp_seq=5 ttl=64 time=143 ms
80 bytes from 192.168.0.2: icmp_seq=6 ttl=64 time=216 ms
80 bytes from 192.168.0.2: icmp_seq=7 ttl=64 time=138 ms
80 bytes from 192.168.0.2: icmp_seq=8 ttl=64 time=175 ms
80 bytes from 192.168.0.2: icmp_seq=9 ttl=64 time=217 ms
80 bytes from 192.168.0.2: icmp_seq=10 ttl=64 time=168 ms
80 bytes from 192.168.0.2: icmp_seq=11 ttl=64 time=191 ms
80 bytes from 192.168.0.2: icmp_seq=12 ttl=64 time=178 ms
80 bytes from 192.168.0.2: icmp_seq=13 ttl=64 time=155 ms
80 bytes from 192.168.0.2: icmp_seq=14 ttl=64 time=138 ms
80 bytes from 192.168.0.2: icmp_seq=15 ttl=64 time=205 ms
80 bytes from 192.168.0.2: icmp_seq=16 ttl=64 time=205 ms
80 bytes from 192.168.0.2: icmp_seq=17 ttl=64 time=216 ms
80 bytes from 192.168.0.2: icmp_seq=18 ttl=64 time=209 ms
80 bytes from 192.168.0.2: icmp_seq=19 ttl=64 time=203 ms
80 bytes from 192.168.0.2: icmp_seq=20 ttl=64 time=182 ms
80 bytes from 192.168.0.2: icmp_seq=21 ttl=64 time=154 ms
80 bytes from 192.168.0.2: icmp_seq=22 ttl=64 time=163 ms
80 bytes from 192.168.0.2: icmp_seq=23 ttl=64 time=134 ms
80 bytes from 192.168.0.2: icmp_seq=24 ttl=64 time=162 ms
80 bytes from 192.168.0.2: icmp_seq=25 ttl=64 time=142 ms
80 bytes from 192.168.0.2: icmp_seq=26 ttl=64 time=223 ms
80 bytes from 192.168.0.2: icmp_seq=27 ttl=64 time=191 ms
80 bytes from 192.168.0.2: icmp_seq=28 ttl=64 time=167 ms
80 bytes from 192.168.0.2: icmp_seq=29 ttl=64 time=181 ms
80 bytes from 192.168.0.2: icmp_seq=30 ttl=64 time=154 ms
80 bytes from 192.168.0.2: icmp_seq=31 ttl=64 time=235 ms
80 bytes from 192.168.0.2: icmp_seq=32 ttl=64 time=218 ms
80 bytes from 192.168.0.2: icmp_seq=33 ttl=64 time=131 ms
80 bytes from 192.168.0.2: icmp_seq=34 ttl=64 time=212 ms
80 bytes from 192.168.0.2: icmp_seq=35 ttl=64 time=211 ms
80 bytes from 192.168.0.2: icmp_seq=36 ttl=64 time=194 ms
80 bytes from 192.168.0.2: icmp_seq=37 ttl=64 time=174 ms
80 bytes from 192.168.0.2: icmp_seq=38 ttl=64 time=144 ms
80 bytes from 192.168.0.2: icmp_seq=39 ttl=64 time=134 ms
80 bytes from 192.168.0.2: icmp_seq=40 ttl=64 time=225 ms
80 bytes from 192.168.0.2: icmp_seq=41 ttl=64 time=162 ms
80 bytes from 192.168.0.2: icmp_seq=42 ttl=64 time=169 ms
80 bytes from 192.168.0.2: icmp_seq=43 ttl=64 time=147 ms
80 bytes from 192.168.0.2: icmp_seq=44 ttl=64 time=215 ms
80 bytes from 192.168.0.2: icmp_seq=45 ttl=64 time=218 ms
80 bytes from 192.168.0.2: icmp_seq=46 ttl=64 time=167 ms
80 bytes from 192.168.0.2: icmp_seq=47 ttl=64 time=157 ms
80 bytes from 192.168.0.2: icmp_seq=48 ttl=64 time=135 ms
80 bytes from 192.168.0.2: icmp_seq=49 ttl=64 time=207 ms
80 bytes from 192.168.0.2: icmp_seq=50 ttl=64 time=198 ms
80 bytes from 192.168.0.2: icmp_seq=51 ttl=64 time=193 ms
80 bytes from 192.168.0.2: icmp_seq=52 ttl=64 time=237 ms
80 bytes from 192.168.0.2: icmp_seq=53 ttl=64 time=184 ms

```

通过icmp的序列号可以看到最开始是递增了两次1，之后基本上是在递增2，偶尔出现一次递增3，符合要求

3. 通过start_webserver.sh测试速率：

输入命令：mininet> external ./www/start_webserver.sh

mininet> internal wget <http://192.168.0.2/bigfile> -O /dev/null

得到以下结果：

```
mininet> internal wget http://192.168.0.2/bigfile -O /dev/null
--2020-06-17 16:52:50-- http://192.168.0.2/bigfile
Connecting to 192.168.0.2:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 102400 (100K) [application/octet-stream]
Saving to: '/dev/null'

/dev/null      100%[=====>] 100.00K  10.8KB/s   in 8.2s

2020-06-17 16:52:59 (12.2 KB/s) - '/dev/null' saved [102400/102400]

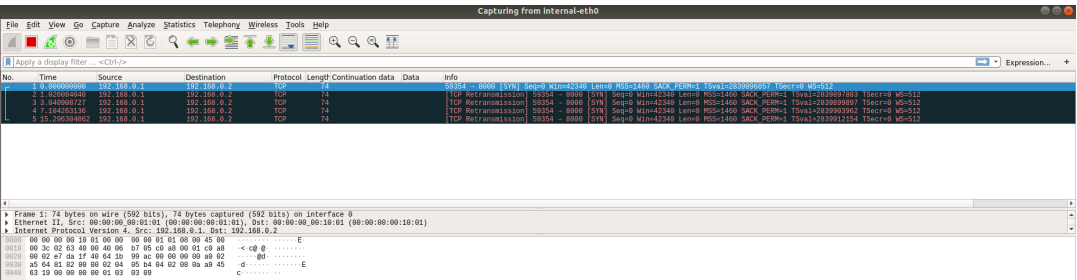
mininet> 
```

可以看到平均速率为12.2，与理论值12.5非常接近！

4. 在mininet中测试impair效果：

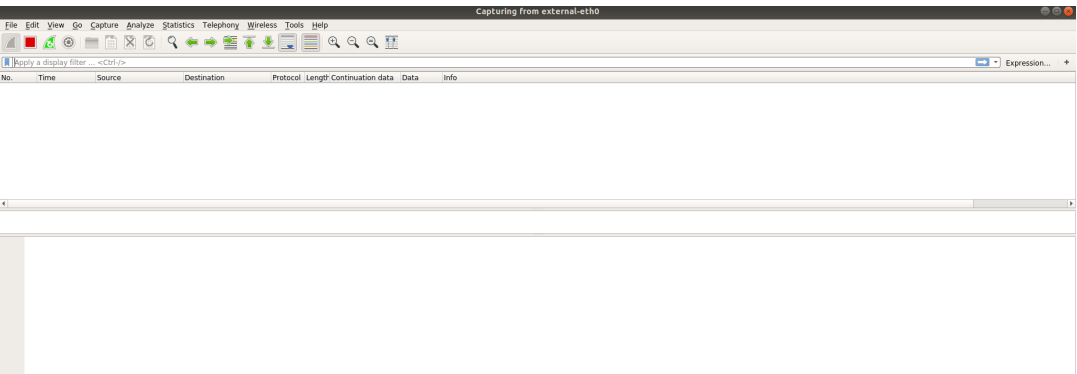
为了方便效果展示，这里直接将丢包率设为1

通过wireshark抓包可以看到以下结果：



internal发包之后由于未收到ack进行重发，但是重发的包依旧丢包

而external部分则是一个包也没收到：



可以看到与理想结果完全一致

经过以上若干测试，可以知道该防火墙代码基本正确

总结与感想

- 测试文件还有待完善，当两个测试都是需要deny的时候，若第一个包由于防火墙实现的bug而发送，此时测试文件会报错在第二个需要deny的位置
- 该实验让我深刻理解了防火墙的基本工作原理，由于自己动手实现了一遍，更加深了理解
- 回顾本学期的整个计网实验，感觉计网实验的设计确实很好（虽然好像不是原创），每个实验都与课程内容紧密结合，让我们在学习理论知识之后马上动手实践，加深对理论知识的理解，而且每个实验的工作量其实并不大，也没有给课后造成太大负担，感觉比之前的计网实验的设计要好