南京大学本科生实验报告

课程名称：**计算机网络**　　任课教师：**李文中**

学院	计算机科学与技术系	专业（方向）	计算机科学与技术系
学号	181860109	**姓名**	**吴润泽**
Email	181860109@smail.nju.edu.cn	**开始/完成日期**	2020/5/27-2020/6/14

1. 实验名称： Firewall

2. 实验目的

1. 构建一个简易的防火墙，允许或者禁止特定流量；
2. 实现基于令牌桶的速率控制和对特定流量的破坏。

---

3. 实验过程

Task 2 Firewall rules

模块1 RULE 类

a. 实现原理

######

1. 记录规则的各个属性，并实现相等的重载操作，来实现数据包与规则的匹配；

b. 代码编写

```
class Rule(object):
    def __init__(self, items: dict):
        self.items = items
        self.perimit = items['permit']
        self.type = items['type']
        self.src = '0.0.0.0/0' if items['src'] == 'any' else
items['src']
        self.dst = '0.0.0.0/0' if items['dst'] == 'any' else
items['dst']
        self.src_port = items['srcport']
        self.dst_port = items['dstport']
        self.ratelimit = items['ratelimit']
        self.impair = items['impair']

    def __str__(self):
        return "{}".format(self.items)

    def __eq__(self, pkt):#判断规则的协议 IP 端口号 是否匹配
        if pkt[Ethernet].ethertype != EtherType.IPv4:
            return False
        protocol, src, dst = pkt[IPv4].protocol, pkt[IPv4].src,
pkt[IPv4].dst
        if not src in IPv4Network(self.src, strict=False): return False
        if not dst in IPv4Network(self.dst, strict=False): return False
        if protocol == IPProtocol.ICMP:
            if not (self.type == 'ip' or self.type == 'icmp'): return
False
```

```
        elif protocol == IPProtocol.TCP:
            if not (self.type == 'ip' or self.type == 'tcp'): return
False
            if not self.type == 'ip':... #判断端口号是否匹配
        elif protocol == IPProtocol.UDP:
            if not (self.type == 'ip' or self.type == 'udp'): return
False
            if not self.type == 'ip':... #判断端口号是否匹配
        else:#不合法的协议
            return False
        return True
```

模块2 RULE 的初始化

a. 实现原理

######

1. 读取文件中的每一条语句，判断是否为规则，进行拆分，传递给 $Rule$ 进行构造；

b. 代码编写

```
def translate(cur_rule: list):  #判断语句是否为规则，并进行拆分
    items = dict()
    if len(cur_rule) == 0 or cur_rule[0] == '#':
        return (None,False)
    else:
        items['permit'] = True if 'permit' == cur_rule[0] else False
        items['type'] = cur_rule[1]
        items['srcport'], items['dstport'] = None, None
        items['ratelimit'], items['impair'] = None, False
        # 是否为TCP和UDP的规则
        length = 10 if cur_rule[1] == 'udp' or cur_rule[1] == 'tcp'
else 6
        for i in range(2, length, 2):
            items[cur_rule[i]] = cur_rule[i + 1]
        if 'ratelimit' == cur_rule[-2]:
            items[cur_rule[-2]] = cur_rule[-1]
        if 'impair' == cur_rule[-1]:
            items[cur_rule[-1]] = True
    return (items, True)

def init_rules(): #读取所有的规则
    rules = list()
    firewall_rules = open('firewall_rules.txt', 'r')
    for line in firewall_rules.readlines():
        line = line.strip().split() #将每一行开头的空格全部去掉
        (items, legal) = translate(line) #进行规则按属性拆分
        if legal: rules.append(Rule(items))
    firewall_rules.close()
    return rules
```

模块3 发包机制

a. 实现原理

######

1. 判断该数据包是否被规则所匹配，其次判断是否被禁止转发；
2. 如果未被匹配或允许转发，则正常转发即可。

b. 代码编写

```
def judge_rule(pkt, rules):# 判断数据包是否被匹配
    for i in range(0, len(rules)):
        if rules[i] == pkt:
            return i
    return -1

def main(net):
    ...if pkt is not None:
        match = judge_rule(pkt, rules)# 判断是否被匹配
        if match == -1: net.send_packet(portpair[input_port], pkt)
        elif rules[match].perimit:
net.send_packet(portpair[input_port], pkt)
```

Task3 Token bucket

模块1 RULE 的初始化修改

a. 实现原理

1. 在初始化规则时，对每一个规则均设置一个令牌桶：

   如果该条规则存在速率限制，则令牌数量初始化为 $2r$，并记录速率限制，否则设为None；

b. 代码编写

```
def init_rules():
    rules, token_bucket = list(), list()
    for line in firewall_rules.readlines():
        ...
            rules.append(Rule(items))
            if items['ratelimit'] != None:#存在速率限制
                token_bucket.append(#初始化容量为2r, 并记录速率限制
                    [int(items['ratelimit']), 2 *
int(items['ratelimit'])])
            else: #不存在
                token_bucket.append(None)
    return rules, token_bucket
```

模块2 发包机制修改

a. 实现原理

1. 如果发送的包存在速率限制，则判断令牌数量是否可以发送该数据包，容量不足则丢弃；

b. 代码编写

```
def token_get(pkt, rule, token_bucket):
    if token_bucket[rule] == None: return True
    pkt_size = len(pkt) - len(pkt.get_header(Ethernet))
    if token_bucket[rule][1] >= pkt_size:
```

```python
        token_bucket[rule][1] = token_bucket[rule][1] - pkt_size
        return True
    else: return False

def main(net):
    ...if pkt is not None:
        if match == -1:...
        elif rules[match].perimit:
            if token_bucket[match] != None:
                if not token_get(pkt, match, token_bucket):continue #丢弃
            net.send_packet(portpair[input_port], pkt)
```

模块3令牌桶更新

a. 实现原理

1. 在每次收到包或者阻塞超时后均进行判断是否需要更新令牌容量；

   令牌数量增加 $r/2$，上限为2r，

b. 代码编写

```python
if time.time() - timer >= 0.5:
    for i in range(len(token_bucket)):
        if token_bucket[i] != None:
            token_bucket[i][1] = min(
                token_bucket[i][0] * 2,
                token_bucket[i][1] + token_bucket[i][0] // 2)
            timer = time.time() #更新计时器
```

Task4 Impairment

a. 实现原理

1. 选择修改数据包的 $payload$ 进行验证；
2. 直接删除原有的 $payload$ 并改为自定义的数据；

b. 代码编写

```python
def impair_pkt(pkt):
    if pkt[Ethernet].ethertype != EtherType.IPv4:
        return pkt
    if pkt.has_header(RawPacketContents):
        index = pkt.get_header_index(RawPacketContents)
        del pkt[index]
        pkt.insert_header(index,RawPacketContents(b'impaired'))
    else:pkt.add_payload(RawPacketContents(b'impaired'))
```

Task5 实现测试

test scenario测试

1. 使用更新后的测试文件 $firewalltests.py$ 进行测试

```
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
    rule 1.
27  Packet arriving on eth1 should be blocked since it matches
    rule 1.
28  Packet arriving on eth0 should be blocked since it matches
    rule 2.
29  Packet arriving on eth1 should be blocked since it matches
    rule 2.
30  UDP packet arrives on eth0; should be blocked since
    addresses it contains aren't explicitly allowed (rule 14).
31  UDP packet arrives on eth1; should be blocked since
    addresses it contains aren't explicitly allowed (rule 14).
32  ARP request arrives on eth0; should be allowed
33  ARP request should be forwarded out eth1
34  IPv6 packet arrives on eth0; should be allowed.
35  IPv6 packet forwarded out eth1.


All tests passed!


(syenv)  njucs@njucs-VirtualBox  ~/switchyard/lab_7   master ● 
```

2. 编写 $impairedtest.py$ 测试 $impair$ 实现

```
#修改TCP的payload
pkt = mketh() + ip + t
s.expect(PacketInputEvent('eth0',pkt),)
impaired_pkt=pkt+b'impaired' #数据包被匹配，应该被修改payload
s.expect(PacketOutputEvent('eth1',impaired_pkt),)
#修改ICMP的payload
pkt = mketh() + ip + icmp_pkt
payload = '''lambda pkt:
pkt.get_header(ICMP).icmpdata.data[:8]==b'impaired' '''
s.expect(PacketInputEvent('eth0', pkt),)
s.expect(PacketOutputEvent('eth1',pkt,exact=False,predicate=pay
load),)
```

测试结果如下

```
(syenv)  njucs@njucs-VirtualBox  ~/switchyard/lab_7   master ●  swyard -t impairetest.py firewall.py
21:14:53 2020/06/15     INFO Starting test scenario impairetest.py
impair payload
impair payload

Results for test scenario Firewall tests: 4 passed, 0 failed, 0 pending


Passed:
1   Packet arriving on eth0 should be impaired since it matches
    rule 1.
2   Packet forwarded out eth1; impaired since it matches rule 1.
3   Packet arriving on eth0 should be impaired since it matches
    rule 2.
4   Packet forwarded out eth1; impaired since it matches rule 2.


All tests passed!
```

Mininet 测试

在 $Mininet$ 中部署 $firewall.py$ 后，进行限速和 $impair$ 的测试。

1. 令牌桶限速测试

   - $mininet$ 执行 $internal\ ping\ -s72\ 192.168.0.2$

```
80 bytes from 192.168.0.2: icmp_seq=23 ttl=64 time=249 ms
80 bytes from 192.168.0.2: icmp_seq=25 ttl=64 time=207 ms
80 bytes from 192.168.0.2: icmp_seq=27 ttl=64 time=183 ms
80 bytes from 192.168.0.2: icmp_seq=29 ttl=64 time=166 ms
80 bytes from 192.168.0.2: icmp_seq=32 ttl=64 time=161 ms
80 bytes from 192.168.0.2: icmp_seq=34 ttl=64 time=135 ms
80 bytes from 192.168.0.2: icmp_seq=36 ttl=64 time=204 ms
80 bytes from 192.168.0.2: icmp_seq=38 ttl=64 time=174 ms
80 bytes from 192.168.0.2: icmp_seq=41 ttl=64 time=157 ms
80 bytes from 192.168.0.2: icmp_seq=43 ttl=64 time=230 ms
80 bytes from 192.168.0.2: icmp_seq=45 ttl=64 time=210 ms
80 bytes from 192.168.0.2: icmp_seq=47 ttl=64 time=178 ms
80 bytes from 192.168.0.2: icmp_seq=49 ttl=64 time=225 ms
80 bytes from 192.168.0.2: icmp_seq=52 ttl=64 time=208 ms
80 bytes from 192.168.0.2: icmp_seq=54 ttl=64 time=191 ms
80 bytes from 192.168.0.2: icmp_seq=56 ttl=64 time=164 ms
80 bytes from 192.168.0.2: icmp_seq=58 ttl=64 time=147 ms
80 bytes from 192.168.0.2: icmp_seq=61 ttl=64 time=157 ms
80 bytes from 192.168.0.2: icmp_seq=63 ttl=64 time=165 ms
80 bytes from 192.168.0.2: icmp_seq=65 ttl=64 time=137 ms
80 bytes from 192.168.0.2: icmp_seq=67 ttl=64 time=224 ms
80 bytes from 192.168.0.2: icmp_seq=69 ttl=64 time=207 ms
80 bytes from 192.168.0.2: icmp_seq=72 ttl=64 time=205 ms
80 bytes from 192.168.0.2: icmp_seq=74 ttl=64 time=193 ms
80 bytes from 192.168.0.2: icmp_seq=76 ttl=64 time=156 ms
80 bytes from 192.168.0.2: icmp_seq=78 ttl=64 time=221 ms
80 bytes from 192.168.0.2: icmp_seq=80 ttl=64 time=159 ms
80 bytes from 192.168.0.2: icmp_seq=83 ttl=64 time=158 ms
80 bytes from 192.168.0.2: icmp_seq=85 ttl=64 time=220 ms
80 bytes from 192.168.0.2: icmp_seq=87 ttl=64 time=200 ms
80 bytes from 192.168.0.2: icmp_seq=89 ttl=64 time=157 ms
80 bytes from 192.168.0.2: icmp_seq=91 ttl=64 time=227 ms
80 bytes from 192.168.0.2: icmp_seq=93 ttl=64 time=219 ms
80 bytes from 192.168.0.2: icmp_seq=96 ttl=64 time=216 ms
80 bytes from 192.168.0.2: icmp_seq=98 ttl=64 time=182 ms
80 bytes from 192.168.0.2: icmp_seq=100 ttl=64 time=158 ms
^C
--- 192.168.0.2 ping statistics ---
101 packets transmitted, 47 received, 53% packet loss, time 101087ms
rtt min/avg/max/mdev = 95.226/181.160/253.838/35.283 ms
```

因为规则中对于 $icmp$ 的速率限制为 $150B/s$，而每个数据包大小为$100B$，故平均在2s内有3个包经由防火墙被发送出去，故基本上为发送2个 $ping$ 收到一个$echo$。

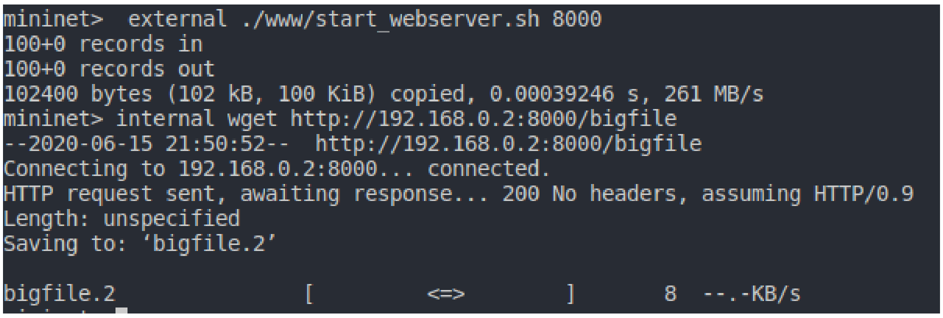
- 利用 $wget$ 测试 $internal\ wget\ http://192.168.0.2/bigfile\ -O\ /dev/null$

```
mininet> external ./www/start_webserver.sh
100+0 records in
100+0 records out
102400 bytes (102 kB, 100 KiB) copied, 0.000317393 s, 323 MB/s
mininet> internal wget http://192.168.0.2/bigfile -O /dev/null
--2020-06-15 21:33:43--  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  10.4KB/s    in 8.3s

2020-06-15 21:33:52 (12.1 KB/s) - '/dev/null' saved [102400/102400]

mininet> internal wget http://192.168.0.2/bigfile -O /dev/null
--2020-06-15 21:34:01--  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  11.6KB/s    in 8.8s

2020-06-15 21:34:10 (11.4 KB/s) - '/dev/null' saved [102400/102400]
```

平均速率在$12KB/s$左右，而规则速率限制为$12.5KB/s$，满足要求。

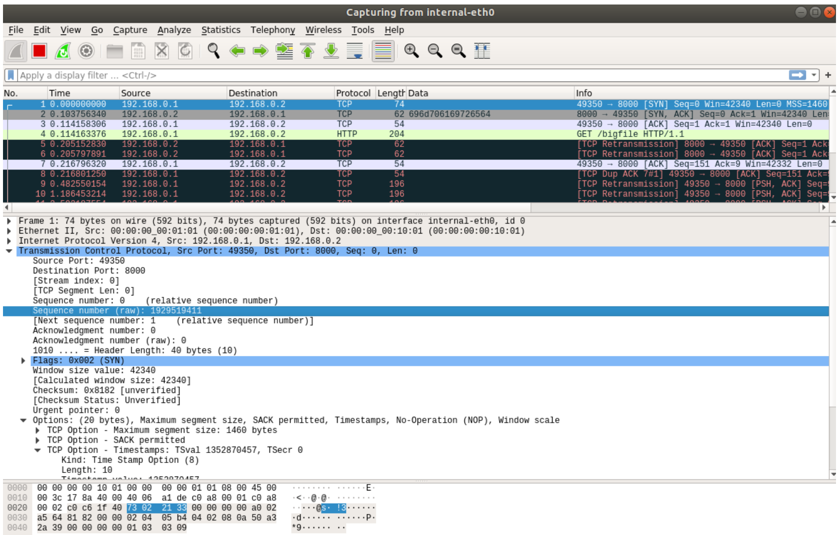
2. impair测试

- 利用 $wget$ 测试 $internal\ wget\ http://192.168.0.2:8000/bigfile$

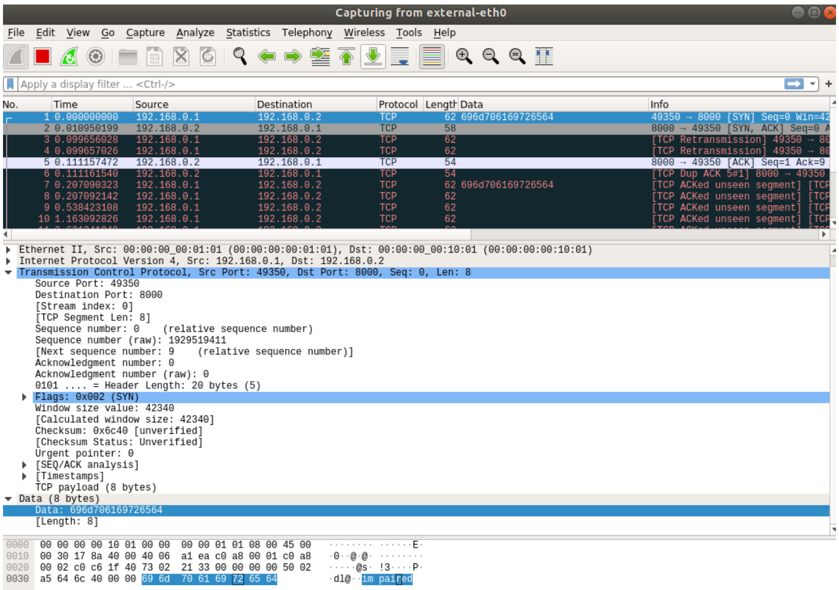


由于在建立HTTP连接时，TCP中的payload便被修改，故无法建立正确连接。

- 观察 *internal* 和 *external* 的抓包结果



internal 发送 SYN 请求建立TCP连接。



而在$external$端收到的数据包多出来了$Data$节，其中的内容为防火墙中修改的内容$impaired$，故TCP的payload被成功覆盖。

4. 总结与感想

防火墙机制的简单实现就此完成。一路坎坷走来，自身的知识理解和具体实践能力在实验中得到了提升，感谢助教们热心为我们讲解实验中遇到的种种问题。