南京大学计算机网络实验报告

任课教师:田臣

实验七 Firewall

计算机科学与技术系

181860055 刘国涛

邮箱:181860055@smail.nju.edu.cn

2020年6月8日

实验目的

- 实现防火墙静态规则
- 实现防火墙流量控制——令牌桶算法
- 实现防火墙的损害检测

实验内容

TASK 2 Implement firewall rules

任务概述 实现防火墙的静态规则，读取规则文件、存储规则、对接收到的包进行规则的匹配

任务实现

根据任务描述，编码任务为编写一个`FireWall`类，并完善以下框架：

```
class FireWall():
    rules = []
    def __init__(self):
        pass
    def read_rule(self,filename):  #从文件中读入规则
        pass
    def permit(self,pkt):   #对包进行规则匹配，判断是否允许通过防火墙
        pass
```

为了对规则进行存储，还要编写一个`Rule`类，框架如下：

```
class Rule():
    def __init__(self,rule): # 根据从文件中读入的一行rule进行构造
        pass
    def match(self,pkt): # 规则与数据包的匹配
        pass
```

在构造函数中，对src和dst使用`IPv4Network`进行存储，如果是`any`的话，则有

```
self.src = IPv4Network('0.0.0.0/0')
```

在实现`match`方法中，需要对前缀和端口进行匹配，通过以下方法实现

```
def prefixMatch(netAddr,ipAddr):
    return int(netAddr) & int(ipAddr) == int(netAddr)
def portMatch(rulePort,port):
    if rulePort == -1: #any
        return 1
    if rulePort == port:
        return 1
    return 0
```

编写完`Rule`类后，则可以实现`FireWall`的`read_rule`方法

```
    def read_rule(self,filename):
        f = open(filename,'r')
        for line in f:
            if not line or line[0] == '\n' or line[0] == '#':
                continue
            rule = line.strip().split(' ')
            self.rules.append(Rule(rule))
        f.close()
```

然后实现`permit`方法

```
    def permit(self,pkt):
        for r in self.rules:
            permit = r.match(pkt)
            if permit == 1:
                return 1
            if permit == -1:
                return 0
        return 1
```

接下来在main中引入：

```
def main(net):
    # ...
    firewall = FireWall()
    firewall.read_rule("firewall_rules.txt")

    while True:
        # recv pkt
        if pkt is not None:
            eth = pkt.get_header(Ethernet)
            if eth.ethertype == EtherType.IPv4:
                if not firewall.permit(pkt): # deny
                    continue

            net.send_packet(portpair[input_port], pkt)
     net.shutdown()
```

> 至此，TASK 2中的Firewall完成

TASK 3 Implement the token bucket algorithm

任务概述 实现令牌桶算法，限制流量速率

任务实现

令牌桶算法在主循环中主要的任务是：

- 每0.5秒更新每一条规则对应的令牌桶，加入令牌
- 接收到包后，判断对应的规则的令牌桶中的令牌是否足够，足够就转发，否则丢包

所以，接下来需要实现一个 `TokenBucket` 类

```
class TokenBucket():
```

```
    update_time = 0.5
    token_num = 0
    def __init__(self,rate):    # 初始化
        if rate < 0:
            self.rate = -1
            return
        self.rate = rate*self.update_time
        self.maxsize = 2*rate
        self.token_num = 0
    def update(self):           # 更新
        if self.rate < 0:
            return
        self.token_num += self.rate
        if self.token_num > self.maxsize:
            self.token_num = self.maxsize
    def permit(self,pkt):       # 判断令牌量是否足够发出数据包
        if self.rate < 0:

            return 1
        pktsize = len(pkt) - len(pkt.get_header(Ethernet))
        if self.token_num > pktsize:
            self.token_num -= pktsize
            print("token use {},left
  {}".format(pktsize,self.token_num))
            return 1
        else:
            return 0
```

为了对所有的令牌桶进行同一的更新管理，这里通过将托管给`TokenBucketManager`的方式实现:

```python
class TokenBucketManager():
    def __init__(self,token_buckets):
        self.token_buckets = token_buckets
        self.last_time = time.time() - 0.5
        self.update_time = 0.5
    def update(self):
        now = time.time()
        if now - self.last_time < self.update_time:
            return
        self.last_time = now
        for i in self.token_buckets:
            i.update()
```

然后在`Rule`中引入令牌桶：

```python
class Rule():
    def __init__(self,rule):
        # ...

        # 引入令牌桶
        if rule[-2] == 'ratelimit':
            self.rate = int(rule[-1])
        else:
            self.rate = -1
        self.token_bucket = TokenBucket(self.rate)

```

在`FireWall`中引入令牌桶的判断：

```python
class FireWall():
    def permit(self,pkt):
        for r in self.rules:
            permit = r.match(pkt)
            if permit == 1:
                if r.token_bucket.permit(pkt): # 引入令牌桶的判断
                    return 1
                else:
                    return 0
            if permit == -1:
                return 0
        return 1
```

在main中引入令牌桶的更新：

```python
def main(net):
    # ...
    firewall = FireWall()
    firewall.read_rule("firewall_rules.txt")

    # 引入令牌桶
    manager = TokenBucketManager([r.token_bucket for r in firewall.rules])

    while True:
        # recv pkt

        # 更新所有令牌桶
        manager.update()

        if pkt is not None:
            eth = pkt.get_header(Ethernet)
            if eth.ethertype == EtherType.IPv4:
                if not firewall.permit(pkt): # deny
                    continue

            net.send_packet(portpair[input_port], pkt)
     net.shutdown()
```

> 于是所有TASK 3的令牌桶算法实现完成

TASK 4 Implement some other type of network impairment

任务概述 可选任务，这里实现的是 `随机丢包`

任务实现

首先为 `FireWall` 实现一个 `impair` 的接口，这样可以方便后续引入不同地impair方法：

```
def lucky_packet():
    drop_rate = 0.2
    ran = random.random()
    if ran > drop_rate:
        return 1
    else:
        return 0

class FireWall():
    def impair(self,pkt):
        return not lucky_packet()

    def permit(self,pkt):
        index = 1
        for r in self.rules:
            permit = r.match(pkt)
            if permit == 1:
                if r.token_bucket.permit(pkt):
                    if r.impair and self.impair(pkt): # 引入impair
                        return 0
                    return 1
                else:
                    return 0
            if permit == -1:
                return 0
            index += 1
        return 1


```

> 至此，TASK 4实现完成

实验结果

Testing:

```
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
    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$
```

Deploying:

测试速率限制：

`internal ping -c40 -s72 192.168.0.2`

```
mininet> internal ping -c40 -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=219 ms
80 bytes from 192.168.0.2: icmp_seq=3 ttl=64 time=229 ms
80 bytes from 192.168.0.2: icmp_seq=5 ttl=64 time=213 ms
80 bytes from 192.168.0.2: icmp_seq=8 ttl=64 time=237 ms
80 bytes from 192.168.0.2: icmp_seq=11 ttl=64 time=227 ms
80 bytes from 192.168.0.2: icmp_seq=14 ttl=64 time=263 ms
80 bytes from 192.168.0.2: icmp_seq=17 ttl=64 time=159 ms
80 bytes from 192.168.0.2: icmp_seq=20 ttl=64 time=155 ms
80 bytes from 192.168.0.2: icmp_seq=22 ttl=64 time=130 ms
80 bytes from 192.168.0.2: icmp_seq=24 ttl=64 time=211 ms
80 bytes from 192.168.0.2: icmp_seq=27 ttl=64 time=185 ms
80 bytes from 192.168.0.2: icmp_seq=30 ttl=64 time=166 ms
80 bytes from 192.168.0.2: icmp_seq=32 ttl=64 time=136 ms
80 bytes from 192.168.0.2: icmp_seq=34 ttl=64 time=219 ms
80 bytes from 192.168.0.2: icmp_seq=37 ttl=64 time=227 ms
80 bytes from 192.168.0.2: icmp_seq=40 ttl=64 time=218 ms

--- 192.168.0.2 ping statistics ---
40 packets transmitted, 16 received, 60% packet loss, time 39511ms
rtt min/avg/max/mdev = 130.390/200.212/263.431/37.997 ms
```

根据`rule 13`的限制（`permit icmp src any dst any ratelimit 150`）

由于ping每次发出100bytes数据包后，需要等待约200ms（最多等待1s，未收到则判断为丢包），才能接受到回复数据包。接收到回复数据包后（或者1s后）立即发出下一100bytes的数据包。

分析前0.5s这个过程：

1. 最初令牌桶共有300个令牌
2. 第一个回显数据包和回显回复数据包，和第二个回显数据包共消耗300个令牌
3. 此时令牌桶空，防火墙接收到第二个回显回复数据包，丢弃
4. 令牌桶加入75个令牌（即使令牌桶和接收到第二个回显回复数据包的顺序改变也是同样的结果）

假如没有速率限制，传输速率约为200bytes / 200ms = 1000 bytes/sec > 150 bytes/sec

所以后续会约每隔两次成功一次ping

接着对`rule 7`和`rule 8`进行测试

```
mininet> external ./www/start_webserver.sh
100+0 records in
100+0 records out
102400 bytes (102 kB, 100 KiB) copied, 0.000611377 s, 167 MB/s
mininet> internal wget http://192.168.0.2/bigfile -O /dev/null
--2020-06-08 01:53:56--  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

2020-06-08 01:54:04 (12.1 KB/s) - '/dev/null' saved [102400/102400]

mininet>
```

将速率限制降低一半

```
# rule 7
permit tcp src 192.168.0.0/16 srcport any dst any dstport 80 ratelimit 6250
# rule 8
permit tcp src any srcport 80 dst 192.168.0.0/16 dstport any ratelimit 6250
# rule 9
```

得到的结果为

```
mininet> internal wget http://192.168.0.2/bigfile -O /dev/null
--2020-06-08 02:01:41--  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  5.91KB/s    in 16s

2020-06-08 02:01:58 (6.23 KB/s) - '/dev/null' saved [102400/102400]

mininet>
```

可以看到，平均传输速率也降低了一半

测试限制措施

采用了 `0.1` 的丢包率

```
mininet> internal wget http://192.168.0.2:8000/bigfile -O /dev/null
--2020-06-09 03:19:49--  http://192.168.0.2:8000/bigfile
Connecting to 192.168.0.2:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 102400 (100K) [application/octet-stream]
Saving to: '/dev/null'

/dev/null           100%[===================>] 100.00K  7.97KB/s    in 27s

2020-06-09 03:20:17 (3.66 KB/s) - '/dev/null' saved [102400/102400]

mininet>
```

总结与感想

最后一次计网实验了，感觉这学期的计网实验体验非常良好，手册很详细，跟着手册做真的可以实实在在地深入理解到计算机网络中的知识。通过这次的实验充分理解了防火墙的工作原理以及速率控制的方式