

Link to Mini Project: <https://github.com/LordJatonyas/CWM-ProgNets/tree/main/firewall>

Mini Project: Simple Firewall

The goal of this project firewall is to prove the capability of blocking websites. This is done by dropping packets at the reflector when the destination address matches a blacklisted website IP ADDRESS. As such, the mini project heavily relies on what's been covered in Assignment 3 (Reflector).

First, instead of relying on the MAC ADDRESS for decisions, we will look at the destination IP ADDRESS.

```

/*****
*****  I N G R E S S   P R O C E S S I N G   *****/
*****/

control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {

    action swap_mac_addresses() {
        macAddr_t tmp_mac;
        tmp_mac = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = hdr.ethernet.srcAddr;
        hdr.ethernet.srcAddr = tmp_mac;

        //send it back to the same port
        standard_metadata.egress_spec = standard_metadata.ingress_port;
    }

    action drop() {
        mark_to_drop(standard_metadata);
    }

    table dst_ip_drop {
        key = {
            hdr.ipv4.dstAddr: lpm;
        }
        actions = {
            swap_mac_addresses;
            drop;
            NoAction;
        }
        size = 65000;
        default_action = swap_mac_addresses();
    }

    apply {
        if (hdr.ethernet.isValid()) {
            if (hdr.ipv4.isValid()) {
                dst_ip_drop.apply();
            }
        }
    }
}

```

The P4 code section above forces packet drops when the destination IP ADDRESS matches any table entry. Another change is the size of the table. Here, I have changed it from 1024 to 65,000. This change was made to accommodate for the large number of IP ADDRESSES owned by YouTube, the target website to block.

YouTube IP Address Ranges

To support a large and growing network of web servers, YouTube owns a number of IP addresses in ranges called blocks.

These IP address blocks belong to YouTube:

- 199.223.232.0 - 199.223.239.255
- 207.223.160.0 - 207.223.175.255
- 208.65.152.0 - 208.65.155.255
- 208.117.224.0 - 208.117.255.255
- 209.85.128.0 - 209.85.255.255
- 216.58.192.0 - 216.58.223.255
- 216.239.32.0 - 216.239.63.255

Administrators who want to block access to YouTube from their network should block these IP address ranges if their router allows.

These IP ADDRESSES are compiled into a commands.txt via a Python script:

```
class YouTube_list():
    def __init__(self):
        self.youtube_ip = []

    def list_youtube_ip(self):
        self.youtube_ip = []
        for i in range(232, 240):
            for j in range(0, 256):
                self.youtube_ip.append(f'199.223.{i}.{j}')
        for i in range(160, 176):
            for j in range(0, 256):
                self.youtube_ip.append(f'207.223.{i}.{j}')
        for i in range(152, 156):
            for j in range(0, 256):
                self.youtube_ip.append(f'208.65.{i}.{j}')
        for i in range(224, 256):
            for j in range(0, 256):
                self.youtube_ip.append(f'208.117.{i}.{j}')
        for i in range(128, 256):
            for j in range(0, 256):
                self.youtube_ip.append(f'209.85.{i}.{j}')
        for i in range(192, 224):
            for j in range(0, 256):
                self.youtube_ip.append(f'216.58.{i}.{j}')
        for i in range(32, 64):
            for j in range(0, 256):
                self.youtube_ip.append(f'216.239.{i}.{j}')
        return self.youtube_ip

if __name__ == "__main__":
    youtube_list = YouTube_list()
    youtube_ip = youtube_list.list_youtube_ip()
    with open('commands.txt', 'w') as f:
        for ip in youtube_ip:
            f.write(f'table_add MyIngress.dst_ip_drop MyIngress.drop {ip}/32 => ')
            f.write('\n')
    f.close()
```

Looking back, this could have been improved by just matching the first 24 bits (since the last 8 bits are 0 to 255 anyway).

The compiled P4 program is then run as a switch on P4Pi-14, and commands.txt used to add tables to the simple switch. In total, there are 64511 entries.

The task is to test if certain destination IP ADDRESSES are affected. This is conducted via randomising destination IP ADDRESSES using the function below.

```
def random_ip():
    youtube_list = YouTube_list()
    ip_list = random.sample(youtube_list.list_youtube_ip(),50)
    ip_list.append("192.168.10.2")
    ip_list.append("8.8.8.8")
    ip_list.append("0.0.0.0")
    ip_list.append("127.89.44.0")
    return random.choice(ip_list)
```

It randomly picks 50 blacklisted destination IP ADDRESSES and puts them into an array. This array also contains 4 perfectly legal destination IP ADDRESSES, and the final line just randomly picks 1 destination IP ADDRESS from this array. The result of running this (using send.py) for 20 packets is shown below, with just 1 reflection and 19 dropped packets (1 x 2 + 19 = 21).

Capturing from enx0c37965f8a23

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-------------|--------------|----------------|----------|--------|---------------------|
| 13 | 1.133010861 | 192.168.10.1 | 209.85.203.112 | UDP | 64 | 50000 → 1024 Len=22 |
| 14 | 1.248166551 | 192.168.10.1 | 209.85.214.72 | UDP | 64 | 50000 → 1024 Len=22 |
| 15 | 1.344073549 | 192.168.10.1 | 216.58.200.175 | UDP | 64 | 50000 → 1024 Len=22 |
| 16 | 1.444158964 | 192.168.10.1 | 208.117.246.87 | UDP | 64 | 50000 → 1024 Len=22 |
| 17 | 1.536122381 | 192.168.10.1 | 216.58.212.87 | UDP | 64 | 50000 → 1024 Len=22 |
| 18 | 1.624115262 | 192.168.10.1 | 216.58.217.165 | UDP | 64 | 50000 → 1024 Len=22 |
| 19 | 1.740264507 | 192.168.10.1 | 209.85.180.235 | UDP | 64 | 50000 → 1024 Len=22 |
| 20 | 1.848079459 | 192.168.10.1 | 127.89.44.0 | UDP | 64 | 50000 → 1024 Len=22 |
| 21 | 1.848970367 | 192.168.10.1 | 127.89.44.0 | UDP | 64 | 50000 → 1024 Len=22 |

Frame 1: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface enx0c37965f8a23, id 0

Ethernet II, Src: BizlinkT_5f:8a:23 (0c:37:96:5f:8a:23), Dst: 00:00:00:00:00:01 (00:00:00:00:00:01)

Internet Protocol Version 4, Src: 192.168.10.1, Dst: 209.85.238.26

User Datagram Protocol, Src Port: 50000, Dst Port: 1024

Data (22 bytes)

```
0000 00 00 00 00 00 01 0c 37 96 5f 8a 23 08 00 45 00  ....7._.#..E-
0010 00 32 00 01 00 00 40 11 f0 a0 c0 a8 0a 01 d1 55  -2...@-.....U
0020 ee 1a c3 50 04 00 00 1e f6 b2 6c 63 63 72 61 6e  ...P....lccran
0030 6e 62 72 6d 6e 68 70 6a 78 67 78 77 72 63 63 6b  nbrmnhpj xgxwrck
```

enx0c37965f8a23: <live capture in progress> Packets: 21 · Displayed: 21 (100.0%) Profile: Default

This matches our expectation since it is much more likely for the randomiser to select a blacklisted destination IP ADDRESS (50 vs 4), and when there was a legal destination IP ADDRESS, a reflection occurred.