PROJECT 4

SDN FIREWALL

CS-6250-O01 Summer 2019

INTRODUCTION

In this project, you will use SDN to create a configurable firewall using an OpenFlow enabled switch. This is beyond what is possible with traditional L2 switches, and shows how the simplest of SDN switches are more capable than even the fanciest of L2 switches.

You are going to create an externally configurable firewall using Pyretic. That means that the firewall rules are provided in a configuration file, so they can be changed without altering the switch code.

IMPORTANT RESOURCES FOR THIS PROJECT

Pyretic Manual (especially How To Use Match and Language Basics)
https://github.com/frenetic-lang/pyretic/wiki
https://github.com/frenetic-lang/pyretic/wiki/How-to-use-match ⇐
https://github.com/frenetic-lang/pyretic/wiki/Language-Basics ⇐

IP Header Format with TCP and UDP Extensions https://en.wikipedia.org/wiki/IPv4#Header and https://en.wikipedia.org/wiki/Transmission_Control_Protocol

Wireshark

https://wiki.wireshark.org/FrontPage

http://mininet.org/walkthrough/#start-wireshark

http://www.linuxandubuntu.com/home/how-to-use-wireshark-toinspect-network-traffic

WHAT IS PYRETIC?

https://github.com/frenetic-lang/pyretic/wiki/Language-Basics

Conceptually, a Pyretic policy is a function that takes a packet as input and returns a set of packets. This function describes what the network switches should do with incoming packets. For example a function that takes any packet and returns the empty set when applied as the overall network policy will cause the network to drop all packets.

Likewise a function that takes any packet arriving at a given location (switch and port) and returns the set of packets which are identical to the original but located respectively at the ports at that switch which lie on the network spanning tree, applied as the overall network policy will cause the network to flood all packets.

However, Pyretic policies can be built from other policies using combinators such as *parallel* and *sequential* composition, so policies such as the ones above can be building blocks of the overall network policy, as opposed to only being used as the overall network policy itself.

PYRETIC LANGUAGE BASICS

https://github.com/frenetic-lang/pyretic/wiki/Language-Basics

match - returns set containing packet if packet's field f matches value v, empty set otherwise

modify - returns set containing copy of packet where field f is set to value v

forward - returns set containing copy of packet where outport field is set to a

Identity - returns a copy of the packet

~ - inverse the rule (Negation)

PYRETIC FILTER POLICIES

Filter policies (or "filters" for short) are policies that don't change the packet - either a set containing just the packet is returned or the empty set is returned. match, drop, and identity are filters.

Filter policies are exactly what you want to conditionally execute code.

+ **parallel composition** - returns the union of A's output and B's output (Policy)

>> **sequential composition** - returns B's output where A's output is B's input (Policy)

& conjunction - similar to parallel composition, but returns a Filter Policy instead of Policy

| **intersection -** similar to sequential composition, but returns a Filter Policy instead of Policy

Note: If you use intersection, parallel composition, or sequential composition, please triple check your firewall implementation to ensure that it works properly. If you wish to use these operators, please refer to pyretic requirements to ensure that you pass it the proper parameters in the proper order.

MATCH LANGUAGE REFERENCE

https://github.com/frenetic-lang/pyretic/wiki/How-to-use-match

The fields used for this project include:

srcmac	dstmac
srcip	dstip
srcport	dstport
ethtype	protocol

IMPLEMENTATION CODE (EXCERPTED)

```
def make_firewall_policy(config):

    # You may place any user-defined functions in this space.
    # You are not required to use this space - it is available if needed.

    rules = []

    for entry in config:

        # TODO - This is where you build your firewall rules...

        # rule = match(srcport = int(entry['port_src'])) (Example)
        rule = match(dstport=1080, ethtype=packet.IPV4, protocol=packet.TCP_PROTO)  #Delete
This Line

        rules.append(rule)
        pass
    allowed = ~(union(rules))
    return allowed
```

ENTRIES DATA STRUCTURES

Your data will be parsed into the following lists of dictionary items using firewall.py

```
# Add it to the policies structure
pol = {'rulenum':rulenum,
       'macaddr_src':macaddr_src,
       'macaddr_dst':macaddr_dst,
       'ipaddr_src':ipaddr_src,
       'ipaddr_dst':ipaddr_dst,
       'port_src':port_src,
       'port_dst':port_dst,
       'protocol':protocol}
policies.append(pol)
```

PROJECT FILES

firewall-policies-bad.pol - This is an example firewall configuration policy that is broken. When parsing, an error message will be thrown. Each line has an error of some type. Try removing lines to see the different possible error messages.This shows how the policy file is parsed by the parse_config function in firewall.py

firewall-policies-good.pol - This is an example firewall configuration policy that disallows all devices to connect to port 1080 on all devices. You can use this as the base for the firewall-config.pol file you will generate later in the instructions. The code to implement this firewall policy is included in the Firewall_policy.py file.

firewall_policy.py - This is the file where you will implement the firewall using python and pyretic code, based on the policy configuration that is passed in via the configuration files.

firewall.py - This is the file that sets up pyretic application and reads the firewall config policy into a data object. DO NOT MODIFY THIS FILE. A shell script is provided to help run it. This file contains the code that is used to parse your firewall-config.pol file, so please look here for the import format for your firewall-config.pol.

firewall-topo.py - This is a mininet program to start your topology. It consists of one switch and two groups of hosts. Modifying this file isn't necessary, but you may choose to try different topologies for testing your code (make sure that your firewall-config.pol works with the original firewall-topo.py, however).

PROJECT FILES (PART 2)

pyretic_switch.py - This implements a learning switch. You do not need to modify this file.

run-firewall.sh - This script runs the firewall using pyretic. (It starts the firewall.py application.) The files need to be in the pyretic directory trees, and this script makes that happen. Also, it allows for different configuration files to be used by giving the filename on the command line.

test-tcp-client.py - This acts as a TCP client: opens a connection, sends a string, then waits to hear it echoed back. You can use this to test your firewall policies.

test-tcp-server.py - This acts as a TCP server: listens on a specified port, echos back whatever it hears. You can use this together with the test-tcp-client.py program.

test-udp-client.py - This acts as a UDP client to test your firewall policies.

test-udp-server.py - This acts as a UDP server which echos back whatever it hears. You can use this together with the test-udp-client.py program.

PROJECT BASICS

The goal in this project is to write a generic implementation of a set of pyretic query and filter policies to implement a blacklist firewall from a firewall configuration policy. This firewall configuration policy contains the following:

rulenumber,srcmac,dstmac,srcip,dstip,srcport,dstport,protocol

Protocols include:

T - TCP Only

U - UDP Only

B - TCP and UDP

I - ICMP

Rule number is for your use only and isn't used in the match policy.

All field MAY have "-" as an option. This indicates that this field is to be ignored for matching.

PROJECT BASICS

Firewall Configuration Policy Information

1,-,-,-,-,-,-,- this rule should match nothing and not block anything

1,-,-,-,-,-,1080,- this is an invalid rule. A protocol must be specified if a port is specified.

Implementation Notes

DO NOT HARD CODE YOUR IMPLEMENTATION TO MATCH THE RULES IN STEP 5.

Start with a simple Firewall Configuration Policy to test your implementation code. You are allowed to share Firewall Configuration Policies for test purposes **AS LONG AS IT DOES NOT INCLUDE ANY RULES SPECIFIED IN STEP 5.**

ERROR ISSUES

You may ignore any errors that reference a dot parser. We are not using DOT files with this project.

If you get an error like nw_proto, you have an error in one of your match statements. Your match is being ignored. Look at the "How to Use Match" pyretic documentations and check your match rules.

You may assume that IPV4 is the only IP Protocol is used, but be careful if you use this to start your policy. It may cause unintended results.

The project instructions give other error conditions and a solution to these problems.

FIREWALL POLICY RULES

One common implementation for a virtual private network solution utilizes PPTP (Point-to-Point Tunnelling Protocol). It now has many issues related to the way it authenticates users. Write firewall policy rules to block PPTP that will prohibit all hosts from accessing a PPTP server running on server2. (TCP Port 1723)

SSH is used to provide a remote shell which can be used to forward other ports or to bypass firewalls. Write firewall policy rule(s) that will prohibit all hosts from connecting to a SSH server on the east hosts (e1-e3). (TCP and UDP Port 22) (https://en.wikipedia.org/wiki/Secure_Shell)

One common way to perform a distributed denial of service (DDOS) attack is to use an amplification attack using the Network Time Protocol (NTP) and Domain Name Services (DNS) taking advantage of the UDP protocol to saturate the links. Write firewall policy rule(s) to protect the DNS and NTP Services on server1 and server2 from being accessed all hosts. However, the DNS and NTP services on server3 should remain accessible to all hosts. (UDP Ports 123 and 53, respectively)

FIREWALL POLICY RULES (PART 2)

Write a series of firewall policy rules disallowing hosts w1 and w2 from pinging mobile1. This rule will be satisfied if the full ICMP response is not completed. Analyzing the interactions with Wireshark is helpful, but not required.

Write a series of firewall policies to disallow all traffic destined to TCP ports 9950-9952 on host e3 from host e1.

Write a series of firewall policies to restrict host mobile1 from communicating to any of the east hosts (e1-e3) on both TCP and UDP protocols.

WHAT TO SUBMIT?

For this project you need to turn in two files (and an optional 3rd file) to Canvas in a ZIP file.

firewall_policy.py - The SDN firewall you created in Step 4. MANDATORY

firewall-config.pol - The configuration file you created in Step 5. MANDATORY

feedback.txt - This is an optional file to include if you have any issues with the VM or pyretic stability for this project. Post information on any difficulties you may have experienced. Your help would be appreciated.