CS 6250 Fall 2017

Project 7 - SDN Firewall

Goal

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.

This firewall is the type that allows all traffic that isn't expressly disallowed (a blacklist firewall). This is the type of firewall that you would find in an office environment where you wish to restrict access to certain resources from other devices. The alternative is to disallow all traffic that isn't allowed by the policy (whitelist firewall). A whitelist firewall is typically used in edge situations protecting internal resources from the outside/Internet (an example is a home internet router). The code we have provided for this project could be modified to implement this second type of firewall, and would be a worthwhile project for those who are interested in learning further.

In the `Project-7` directory, there are many files, described below:

- `firewall-policies-bad.pol` - This is an example firewall configuration policy that is broken. When parsing, an error message will be thrown. Note that only one line in this file is considered “bad”. 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.
- `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.
- `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).
- `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.

References

The following documents and papers will help with completing and understanding this project:

- Pyretic Manual (especially How To Use Match and Language Basics) - https://github.com/frenetic-lang/pyretic/wiki
- IP Header Format with TCP and UDP Extensions https://en.wikipedia.org/wiki/IPv4#Header and https://en.wikipedia.org/wiki/Transmission_Control_Protocol
-

Instructions

DISCLAIMER: Read all the instructions carefully! Then read them again after you finish but before you submit, so you can verify that what you did matches what the assignment says ***exactly****. Even seemingly small details in the instructions can be very important! Some of those details exist to allow our grading code to interface with your project properly.* ***You will lose points if your project does not work with our grader because you didn't follow the instructions.*** *We do not arbitrarily deduct points for not following directions, but you will not receive credit for your project if our grader cannot tell whether or not it's working.*

1. Before getting started, you will need to download Project-7.zip to the VM and unzip.
    - `unzip Project-7.zip`
2. Next, in the `Project-7` directory try out `test-tcp-client.py`, `test-tcp-server.py`, `test-udp-client.py`, and `test-udp-server.py` tools to learn how to use them by following the following steps:
    - Open a terminal and copy the `pyretic_switch.py` file to the pyretic modules folder:

        - `cp pyretic_switch.py ~/pyretic/pyretic/modules`

    - Start the `pyretic_switch.py` file:
        - `cd ~/pyretic`
        - `python pyretic.py pyretic.modules.pyretic_switch`

    - In a second terminal, start the topology in the `Project-7` directory:
        - `sudo python firewall-topo.py`

    - At the Mininet prompt, you need to open up a couple of new terminals:
        - `e1 xterm &`
        - `e2 xterm &`

    - In the e1 terminal, start the server:
        - `python test-tcp-server.py 10.0.0.1 1234`
        - The parameters are the server's IP and port. You can run `ifconfig` in the xterm window to check the IP.
        - The server simply uses an infinite loop around the accept function, so you can use `Ctrl-C` to kill it when you're done.
        - The UDP server works in the same manner.

    - In the e2 terminal, start the client:
        - `python test-tcp-client.py 10.0.0.1 1234`
        - The parameters here are the server's IP and port, and should be the same as specified above.
        - Likewise, the UDP client functions in a similar manner to the TCP client.

    - You'll be using this quite a bit, so feel free to play around with it for a bit before moving onto creating the firewall.

3. The following set of steps shows how to start and run the firewall. This step will use the one included match rule in `firewall_policy.py` file and the `firewall-policies-good.pol` file. This will implement a minimal firewall that will block TCP port 1080 for all hosts.

    - In the first terminal, start the firewall as specified below. Please provide the firewall-policies-*.pol file that you want to test as the second parameter. **DO NOT RUN THIS AS ROOT (i.e., sudo).**

        - `./run-firewall.sh firewall-policies-good.pol`

          Note: You may get the following error message when you run this command that you can safely ignore: **Couldn't import dot_parser, loading of dot files will not be possible.** We are not using dot files with this project.

          If you accidentally run this script as root, you will need to run the following commands to avoid the "access denied" error.

```
sudo chown mininet:mininet /home/mininet/pyretic/pyretic/modules/firewall_policy.py
sudo chown mininet:mininet /home/mininet/pyretic/pyretic/modules/firewall-policies.cfg
sudo chown mininet:mininet /home/mininet/pyretic/pyretic/modules/firewall.py
```

- In a second terminal, start the topology:
  - `sudo python firewall-topo.py`
- At the mininet prompt, open client and server terminals:
  - `e1 xterm &`
  - `e2 xterm &`
- In the e1 terminal, start the server:
  - `python test-tcp-server.py 10.0.0.1 1234`
  - The UDP server can be used in the same manner.
- In the e2 terminal, start the client:
  - `python test-tcp-client.py 10.0.0.1 1234`
  - The UDP server can be used in the same manner. Note that the IP and port here should be that of the server.

Using the provided good firewall-policies-good.pol files, these connections should connect and pass traffic.

- Stop the test client/server from the last two steps.
- In the e1 terminal, start the server:
  - `python test-tcp-server.py 10.0.0.1 1080`
  - The UDP server can be used in the same manner.
- In the e2 terminal, start the client:
  - `python test-tcp-client.py 10.0.0.1 1080`
  - The UDP server can be used in the same manner.

Using the provided good firewall-policies-good.pol files, these connections should **NOT** connect, indicating that the firewall is working (using Rule 1 from the firewall-policies-good.pol file)

NOTE: A good practice is to run "sudo mn -c" and "killall python" to make sure all processes are shut down. If the error message includes the word "pox", also consider running "killall pox"

If you see the following error, please kill mininet and all python processes and try again:

```
ERROR:core:Exception while handling OpenFlowNexus!PortStatus...
Traceback (most recent call last):
 File "/home/mininet/pox/pox/lib/revent/revent.py", line 231, in raiseEventNoErrors
   return self.raiseEvent(event, *args, **kw)
 File "/home/mininet/pox/pox/lib/revent/revent.py", line 278, in raiseEvent
   rv = event._invoke(handler, *args, **kw)
 File "/home/mininet/pox/pox/lib/revent/revent.py", line 156, in _invoke
   return handler(self, *args, **kw)
 File "/home/mininet/pyretic/of_client/pox_client.py", line 577, in _handle_PortStatus
   self.switches[event.dpid]['ports'][event.port] = event.ofp.desc.hw_addr
KeyError: 1
```

4. Now you will create the firewall implementation by editing `firewall_policy.py` file. This python script will process the configuration file which is parsed into a list of dictionaries by firewall.py and creates the appropriate match/pyretic actions based on the configuration file. You work with one entry at a time.

    - **NOTE: DO NOT HARDCODE FIREWALL RULES IN THIS FILE!** The SDN Firewall we are creating is a general solution, and should support a wide variety of configuration files. You will not receive credit for hard-coding the firewall rules!
    - The code you generate must support all items contained in the parsed configuration file. For example, you should be able to handle a source IP address and a destination MAC address. Also note that some parameters may need additional information. Refer to the Pyretic How to Match (https://github.com/frenetic-lang/pyretic/wiki/How-to-use-match) and Language Basics (https://github.com/frenetic-lang/pyretic/wiki/Language-Basics) for the information needed to fill in the code in firewall-policy.py file.
    - The format of the parsed configuration file is as follows:

      rulenum, source MAC, destination MAC, source IP, destination IP, source Port, destination Port, Protocol number

      MAC Addresses are in the form of 00:11:22:33:44:55; IP Addresses are in the form of 10.0.0.1. You cannot use CIDR notation for IP Addresses; for protocols, provide support by using the assigned number (refer to https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml for a list of protocol numbers as defined in the header). If you have questions to what is acceptable for the parsed configuration file, refer to the parse_config function in firewall.py.

- Treat “-“ in a parsed configuration file field as if you were to ignore it.
- You can assume that if a Port number is included, the protocol must be included.
- You can assume that you will never be asked to block a protocol from all hosts to all hosts. (i.e., no rule like: 1,-,-,-,-,-,-,6)
- Only IPV4 will be used in this project. It is acceptable to hard code this parameter WHERE APPROPRIATE.
- You do not need to take ARP traffic into consideration for this project.
- You may make any changes you need to the *make_firewall_policy* function, but it is recommended to drop any code you need in the #TODO areas. You may or may not need both areas. You may also make auxiliary functions in this file. DO NOT ADD CODE TO OTHER FILES.
- When running the firewall, you may get errors like:

  ```
  Warning: libopenflow_01:Fields ignored due to unspecified prerequisites: nw_proto
  ```

  If you get this error, note that this rule will be ignored and not processed. Read the **How to Use Match** pyretic document to find the error.

5. Once you have a working firewall, test it out by creating your own unique configuration files and repeat the steps from Step 3 above. You are free to share example configuration files with other users. You are also allowed to create your own topologies and are allowed to share them with others. You are NOT allowed to share your firewall-policy.py implementation, or the `firewall-config.pol` you will be creating in subsequent steps.

   Your firewall-policy.py file will be tested against alternate configuration files and topologies, so please make sure that your code is robust to handle different combinations of situations including using combinations of IP Addresses and MAC Addresses, different ports, and different protocols..

6. The next step of the project is to create a firewall policy configuration file in the format as specified in Step 4. This file MUST be named `firewall-config.pol`. The rules you need to implement are detailed below. Please note that you may need to research for some of the necessary information. Please know that you need to craft the rules in such a way that you don't over-block items (i.e., if you are asked to block TCP port 50, make sure that you do not block UDP port 50). Also, make sure that you utilize the "-" operator to ignore a particular parameter. When you are writing your rules, you may use either IP Addresses or MAC Addresses. However your `firewall_policy.py` MUST be able to handle any combinations of IP Addresses or MAC Addresses.

   For rules that specify a particular TCP/UDP protocol by name, utilize the common port used by that protocol.

   Rules to be implemented:

   1. **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 systems from accessing a PPTP server running on server2. (https://en.wikipedia.org/wiki/Point-to-Point_Tunneling_Protocol)**
   2. **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 computers from connecting to a SSH server on the west hosts (w1-w3). (https://en.wikipedia.org/wiki/Secure_Shell)**
   3. **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 all hosts. However, the DNS and NTP services on server3 should remain accessible.**
   4. **Write a series of firewall policy rules disallowing host w1 from pinging mobile1.**
   5. **Disallow all traffic destined to TCP ports 9950-9952 on host e3 from host e1.**

6. **Write firewall policy rule(s) to prohibit the mobile1 host from accessing the NetBIOS over TCP/IP (NetBT – a total of 3 ports) on any host.**

What to turn in

For this project you need to turn in two files to T-Square:

- `firewall_policy.py` - The SDN firewall you created in Step 4
- `firewall-config.pol` - The configuration file you created in Step 6

PLEASE MAKE SURE THAT YOU NAME THESE FILES WITH THE NAMES SPECIFIED. YOU WILL LOSE 15 POINTS PER FILE IF IT IS MISNAMED.

What you can and cannot share

For this project, you can and are encouraged to share testing techniques, your testing firewall policies, and testing topologies. What you cannot share is code for `firewall_policy.py` or examples of configuration policies that would belong in `firewall-config.pol`. See Part 1 Step 5 for more examples of what can and cannot be shared. If you have any doubt please ask the Instructors privately on Piazza before sharing.

Questions to Ponder

To help you implement your firewall policy and configuration rules properly, think about the following topics. Feel free to discuss these on Piazza:

- When creating firewall rules, should you be using BOTH source and destination ports when creating a rule? Why or why not?

- Pyretic has specific requirements when port numbers are specified. Why is this so?
- If you do a wireshark traffic dump while testing your rules, why do you sometimes see one-sided responses?
- Given a rule to block access to a port 80 server on host E2, should you be able to access the port 80 service FROM host E2? Why?

Notes

Wireshark may be helpful for this project. You can start it from any terminal with the following command: `sudo wireshark &` (it needs to run as root to sniff traffic) and use it to look at traffic on specific ports. You may wish to start *two* instances - one either side of the switch - to see if traffic is being if actually being completed and received.

Note that you can increase the RAM or processor usage for the VM for this project to improve performance. Please remember to set it back to the original settings for Project 8 if you do so.

Grading Policy and Rubric

Your `firewall_policy.py` will be tested with a set of known good configuration files and different topologies that are not provided. These configurations will range from simple port blocking to others that are more complex than what was defined for this project. Also, both simple and complex topologies will be used to evaluate your policy.

Your `firewall-config.pol` file will be tested for validity and functionality by using a known good firewall_policy.py file. It will also be tested in conjunction with your `firewall_policy.py` file.

If you have trouble coding the `firewall_policy.py` file, at least attempt to create the `firewall-config.pol` to get additional partial credit.

20 pts	Correct Submission	for turning in all the correct files with the correct names, and significant effort has been made in each file towards completing the project. This will be 10 points for each file.
40 pts	Firewall Policy	the policy in `firewall-config.pol` passes a variety of tests to ensure correct blocking of traffic per the rules, and allows all other traffic. If there are significant issues with your firewall_policy.py file, your work may be verified against a known good firewall_policy.py file.
40 pts	Firewall Implementation	the firewall implementation in `firewall_policy.py` passes a variety of tests to ensure it works properly with several different firewall configurations (i.e., different firewall rules and topologies)