Rotating Adaptive Network Defense (RAND)

Final Report

Team ID: sddec18-07

Client: Argonne National Laboratory, Dr. Benjamin Blakely and Joshua Lyle

Faculty Advisor: Dr. Hongwei Zhang

Team Members and Roles:
Andrew Thai — Project Manager
Connor Ruggles — Usability Manager
Emily Anderson — Deliverable Manager
Ryan Lawrence — Communication Manager
Corey Wright — Quality Assurance Manager

Team email: sddec18-07@iastate.edu
Team Website: https://sddec18-07.sd.ece.iastate.edu/

Table of Contents

List of Figures

List of Definitions and Acronyms

SDN: Software-Defined Network

- Software-defined networking is a concept where the actual routing of data packets is moved to a separate layer and is taken care of programmatically by a network controller, that then sends the packets down to the main network switch to route to the individual servers on the network.

MTD: Moving Target Defense

- This is a concept where you detect if a specific machine is being attacked and you have preset rules to mitigate to rotate that machine out of being public facing, and rotate in a “honeypot”, or something that looks like a real machine but it distracts the attacker long enough to block them out.

NIC: Network Interface Card

- This is the physical device that connects all of the machines connected to a switch, to the Internet.

CDC: Cyber Defense Competition

- A type of competition where teams try and defend a set of servers against a team of attackers in a pre-defined scenario.

VM: Virtual Machine

- A software emulation of physical aspects needed to run a full computer operating system.

Honeypot Server

- A server that is used to trap hackers from accessing the actual production systems.

1 Revised Project Design

We revised our project design last semester into this improved design to help create a software defined network moving target defense system. This design consist of two servers: Floodlight Controller and Snort. The Floodlight Controller is our brains of the system in which it handles all the traffic routing and dynamic packet flow. The Open vSwitches that are within the network will connect to the Floodlight Controller and will route traffic based on the rules that are created within the Floodlight Controller. The Snort server is our network-based intrusion detection system which handles network traffic and alerts us of any malicious traffic that may going through our network. These two systems will be put in front of the network of our Web Server allow for traffic to be blocked or redirected to a honeypot server.

In our design, we have incoming traffic coming from the Internet, once the traffic makes it into the network where the Snort server is located, Snort will start analyzing all the traffic and alerting for any malicious traffic that is incoming based on the community rules and the local rules that we have created. We decided to use Snort as our intrusion detection tool because it will be scalable in the long run with updated community rules. Once Snort creates an alert we have a custom python script that we created and runs every 30 seconds on the system (based on using the system's crontab) that checks if there are any alerts in the file /var/log/snort/alert. If there are any logs, the script will parse through the alerts to grab the source and destination IP. Once the script gathers the IP it creates a flow to either block or redirect the current traffic to a honeypot server by communicating with the Floodlight Controller via API calls.

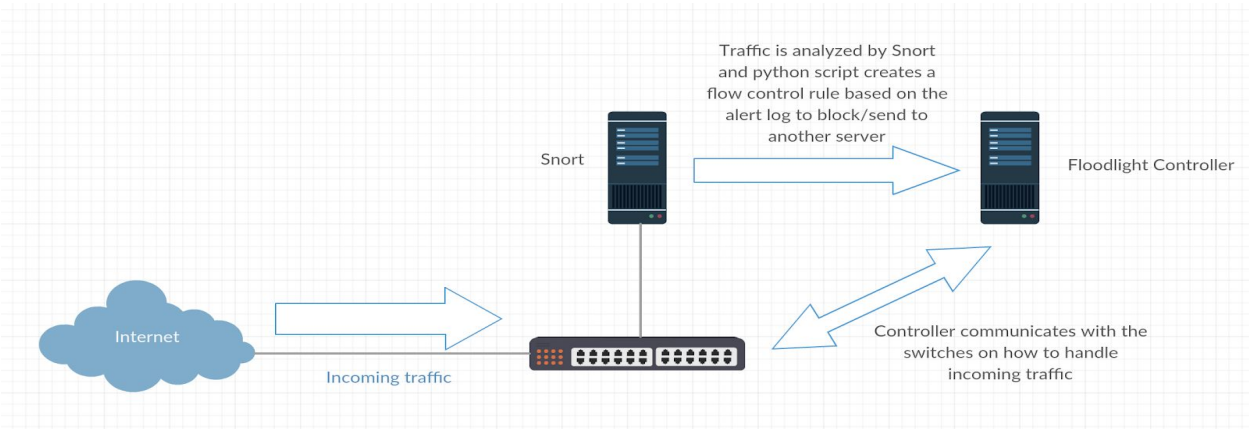

Figure 1: Visual representation of how Snort and the Floodlight controller communicate with the network

Once the rules are pushed to the Floodlight Controller, we can see that the overall design made such that once the alert is created that it will have 3 options: 1) block the traffic 2) redirect the traffic to the honeypot server 3) let the traffic continue as normal

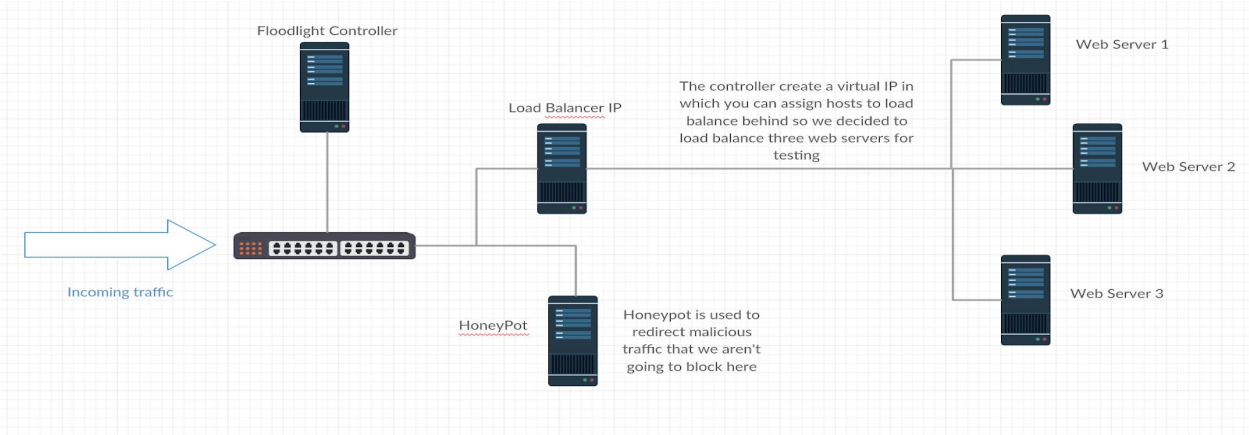

Figure 2: Detailed design of the internal network

2 Implementation Details

We implemented this design on our own personal server running VMware Hypervisor 6.5 running the following 4 servers within the VMware Hypervisor:

Floodlight Controller

- CPU: 4 vCPUs
- RAM: 4 GB
- Hard Disk: 16GB
- OS: Ubuntu 16.04.4 LTS
- Software: Floodlight version 1.2

Snort

- CPU: 4 vCPUs
- RAM: 4 GB
- Hard Disk: 100 GB
- OS: Ubuntu 16.04.4 LTS
- Software: Snort Version 2.9.11.1 GRE (Build 268)

Kali - Pentesting Machine

- CPU: 4 vCPUs
- RAM: 2 GB
- Hard Disk: 16 GB

XenServer Hypervisor

CPU: 8 vCPUs

RAM: 16 GB

Hard Disk: 255 GB

OS: XenServer 7.3.0

We decided to use the XenServer Hypervisor because it implements Open vSwitch protocols within its networking so that we can easily use its virtual networking with our Floodlight Controller. Within the XenServer Hypervisor we have two servers in which one will be acting as our web server and the other as our honeypot server. They will both be running Ubuntu 16.04.4 LTS with the same web configuration. On the snort machine we created a file named “idp.py” which is the python script that parses through the log file /var/log/snort/alert (the location in which snort alerts are created and stored) every 30 seconds because we created the following two crontab entries:

* * * * * (python idp.py 192.168.1.40)

* * * * * (sleep 30; python idp.py 192.168.1.40)

Where 192.168.1.40 is the IP address of the Floodlight Controller. Once the alert is created then it will send a python API call using a POST request to the Floodlight Controller that will create the new follow to redirect or block.

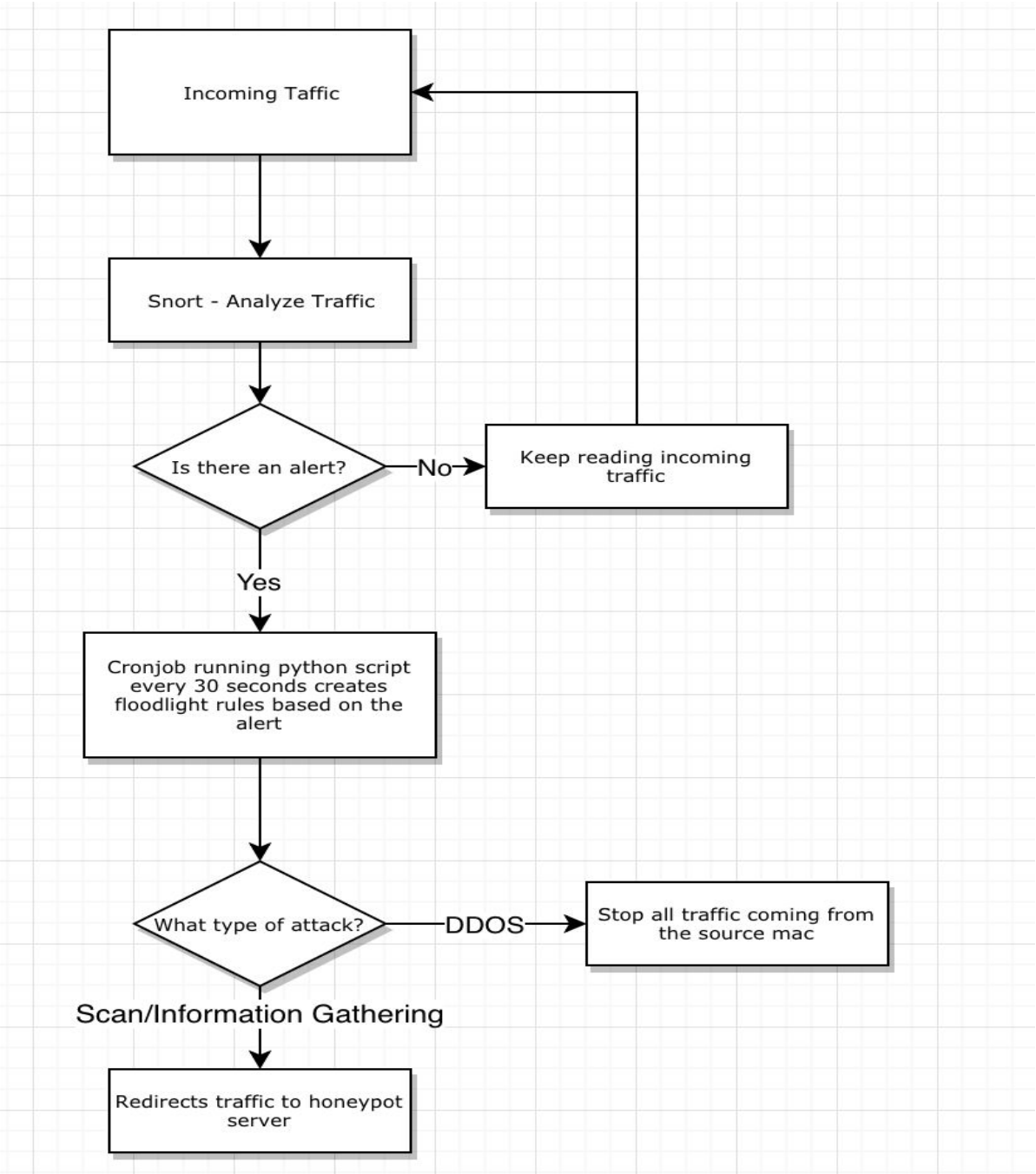

Figure 3: Block diagram of how our implementation is designed based on our servers

3 Testing

3.1 Testing Process

Testing of the SDN MTD system consisted of two portions. First, our own testing for a baseline with internal testing of lag due to overhead. Additionally, we attempted to the test results of our system

defending from attacks propagated by the *red team* during the Community College CDC that was held in Coover Hall on December 1st, 2018.

To form a baseline to test against for all future applications and to determine the overhead that our new integration and code causes within our system a Cron tab was setup in our testing environment to run metrics over time. The crontab measured response time every 30 minutes over several weeks to analyze packet loads under normal circumstances which provided us with long term, stable numbers on the min, max, and average ping rates. At that point we could test the ping rates while running various tests including those that are redirected to the honeypot such as mapping attacks and those that are simply blocked such as DDoS attacks.

During the Community College CDC held at Iowa State on December 1st our team attempted to set up our software defined network to be tested by the red team. We migrated the Snort machine with the Floodlight controller that our work is based on to the required CDC servers. The stated goal of the competition was to set up administrative servers to be tested against by the red team and to take control of stored simulated files that emulate an Iowa State administrative system, primarily a Canvas box with stored names, birthdates, classes, and other assorted information. With our system in place between the "public" facing connections and the stored information, all malicious traffic should be intercepted and either misdirected or stopped. Ideal results would show that malicious attacks from the red team are unable to reach our machines with generated logs from the Canvas box and alerts from the Snort machine. Unfortunately, due to difficulties with the CDC required machines and their implementation, we were not able to enter this project into the CDC for testing. While we were not able to compete, we were able to set up our SDN MTD system with various servers being protected.

3.2 Testing Results

Through our testing we determined that the packet speed in which a packet transfers to the same server vs load balancer vs a scan is shown in the table above. We can see that when directly pinging the server we get a decent average response time, with a comparable response time using the load balancer. With the scans we determined out of every few hundreds of packets were transferred that we only lost one packet due to when the rule was added and applied to the controller since there was a long enough delay for that packet to be dropped not knowing where to go. The average response times for the packets seemed to have increased during our scanning period to suggest that there is a little more overhead when adding the Floodlight Controller rule in our network for where the destination of where the packet would be going. Below are the times of our results as well as graphs to show the ping times during our testing of our network design:

Packet Test	Min (ms)	Avg (ms)	Max (ms)
Direct	0.34331481	0.66207407	2.70742593
Load Balance	0.34418182	0.68587273	2.42092727
Nikto Scan	0.43866968	0.77452036	5.67220814
Nmap Scan	0.471375	0.719875	1.9785

Figure 4: Table of testing packet delay results

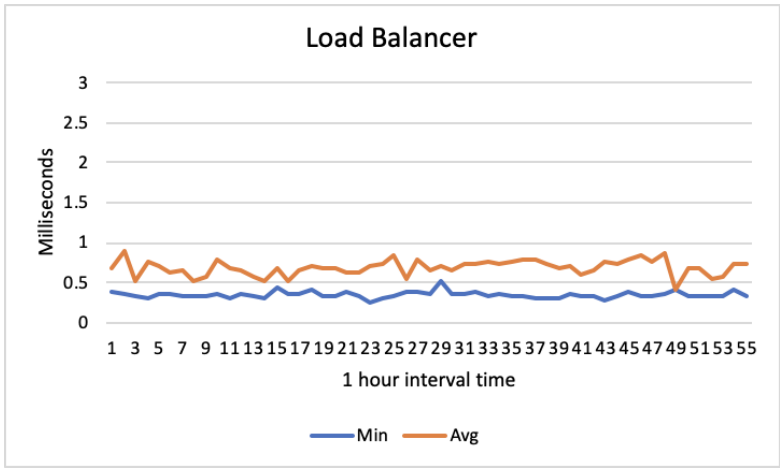

Figure 5: Load Balancer baseline ping graph

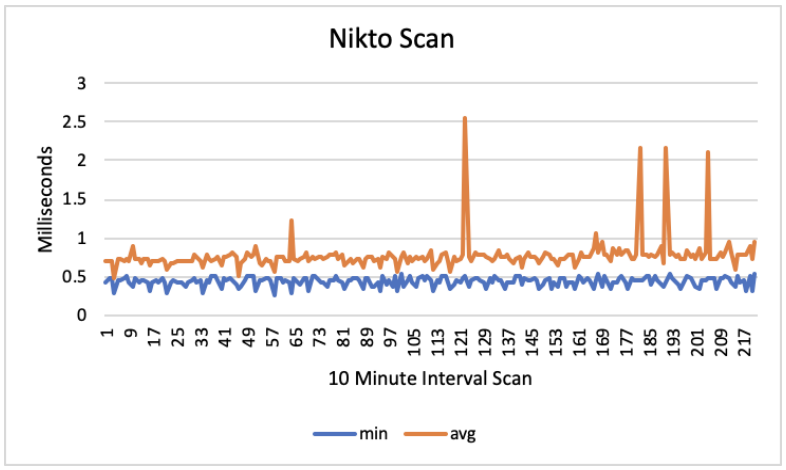

Figure 6: Nikto Scan packet response times

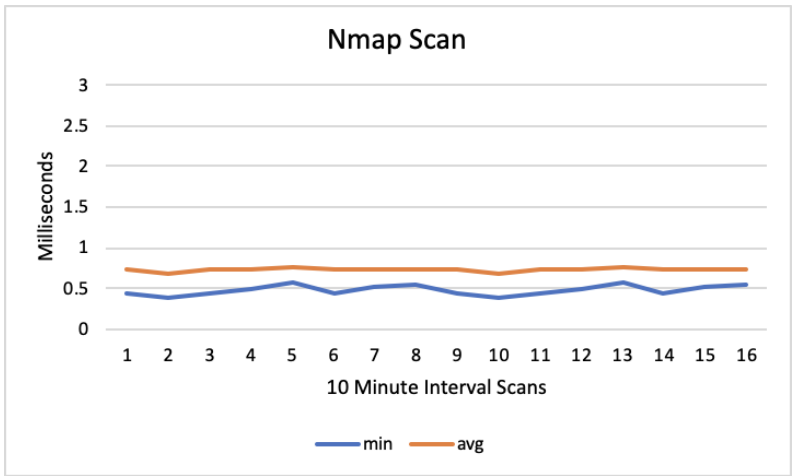

Figure 7: nmap scan packet response times

As mentioned above, our testing with the CDC did not go as planned, but we still got some useful information from it. We were able to set up our system on the competition network without too much difficulty. There are a lot of small components that need to work together in order for this system to work, so it was good to test being able to successfully set it up in a simulated network environment. Not being able to test the functionality, although disappointing, isn't the end of the world. The attacks run by red team are so broad that we suspect the results from our specific testing are more useful than those would have been. Red team will do a variety of attacks that would not necessarily contain just scanning our machines but of other penetration testing tools that our intrusion detection system, Snort, may not have alerted which would result in missed opportunities for the design to be fully tested. This is avoided with our manually testing because we know exactly which types of scans will alert in our intrusion detection system so we can continuously test our network design and make sure that it gets reasonable results from the rules that we created.

4 Related Products/Literature

Software Defined Networking Moving Target Defense is a relatively new field with many of the papers and research in this area published in the last five years. At this point the theory is proven and a lot of the related literature is on how much of a positive effect such a system can have. Our area of research on the other hand is to recreate a system that Argonne National Labs can use in their future research and as a reference and starting point. At Argonne they have a type of hardware moving target defense and wanted to explore the viability and savings associated with moving to a software defined system.

5 Appendices

5.1 Operation Manual

Instructions for setting up the network and systems

Configure each of the servers that you have with their own static IP and make sure that the servers that you will be dynamically routing will be plugged into the OpenFlow compatible switch.

Floodlight Server

```
# Navigate to our git repo and go to the loadbalance directory
cd loadbalance

# Setup Load Balance of web servers on your floodlight machine run the configure
# loadbalance script with mac addresses
./configureLoadBalance.sh <mac address 1> <mac address 2> <mac address 3>

# Then copy and paste the output into the terminal and run it such as the example
# below
curl -X POST -d
'{"id":"1","name":"vip1","protocol":"icmp","address":"192.168.1.50","port":"20"}'
http://localhost:8080/quantum/v1.0/vips/
curl -X POST -d '{"id":"1","name":"pool1","protocol":"icmp","vip_id":"1"}'
http://localhost:8080/quantum/v1.0/pools/
curl -X POST -d '{"id":"2","name":"pool2","protocol":"tcp","vip_id":"1"}'
http://localhost:8080/quantum/v1.0/pools/
```

```
curl -X POST -d '{"id":"1","address":"192.168.1.43","port":"2","pool_id":"1"}'
http://localhost:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"2","address":"192.168.1.44","port":"3","pool_id":"1"}'
http://localhost:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"3","address":"192.168.1.45","port":"7","pool_id":"1"}'
http://localhost:8080/quantum/v1.0/members/
```

Start snort server such that the alerts output to the log file. We created a script to start snort and ran it in a screen session to allow for easy accessibility of restarting and configuring snort.

```
# Start screen session
Screen -S snort

# Navigate to snort directory in our git repo
cd snort

# Run start script for snort
./start_snort.sh
# Exit screen session
Control-A Control-D
# this will print out a message like
# "[detached from <name of screen session>]"
# if you want to get back into the same screen session that snort is running in,
# you can run the command below
screen -r <name of screen session>
```

We will need to connect the XenServer OpenFlow switches to the Floodlight Controller

```
# OpenFLow command to connect to switch to the Floodlight Controller
ovs-vsctl set-controller xenbr0 tcp:<floodlight controller ip>:6653
```

In order to set up Floodlight, the interface and the optional CORS reroute backend, head over to https://git.ece.iastate.edu/sd/sddec18-07/tree/master and read all of the corresponding README's. They outline everything that needs to be done in order to set up the Floodlight software and the new GUI (Note: There's an optional NodeJS backend that can be started to get around a CORS issue with the Floodlight server, you can read the README in the 'cors-reroute-backend' folder in the link above to get more details).

First, you'll need NodeJS, Java, ant and git installed. Then, you'll need to clone the repository to the machines that are running the interface and floodlight, if there is more than one. Read the README in the repository to find out how to run the startup script and which options to provide in order to start the correct servers. For all files, directories, and filepaths that are mentioned in this section assume that the working directory is the cloned repository directory.

On the machine that you want to start the interface, go to the file 'interface/src/config.ts'. There are a few variables that need to be looked at:

- NEEDS_CORS_REROUTE
    - By default set to true. This variable will determine whether the interface attempts to send it's requests to the optional backend or the Floodlight server

- PORT
    - The port that the interface should send the requests to, which is determined by the 'NEEDS_CORS_REROUTE' variable. This should not be changed, unless you change the ports that the optional backend and Floodlight are started on
- API_ROUTE
    - The IP address that Floodlight is running on; change this to whatever the IP is of the machine you would like to put Floodlight on
- MAC
    - The MAC address of the machine that floodlight is running on; this could potentially be changed to an Array or a Map, if more than one machine will be set up to run Floodlight

As a quick reference, you can start Floodlight and the optional backend by running the 'start_floodlight.sh' script with the '-cors' option, but as noted in the script and all of the README's, you only need to start the optional backend if you want to run the interface on a different machine than Floodlight. If you want to start Floodlight and the interface on the same machine, just run the 'start_floodlight.sh' script with no options, as that is the normal behavior.

If you intend to use the 'cors-reroute-backend' app, there's a variable similar to the interface that will need to be changed in 'cors-reroute-backend/lib/app.ts':

- API_ROUTE
    - The IP or hostname of the server that Floodlight is running on. Change this to whatever is correct for your setup

That script (start_floodlight.sh) runs all the commands necessary to start all servers, but here are those same commands in case the script just won't quite do the job, or your software setup isn't how the script might expect it to be:

```
# start the floodlight server
cd floodlight
ant
java -jar target/floodlight.jar

# start the optional backend
cd cors-reroute-backend
npm install
# for dev purposes
npm run dev
# for compiling and starting a prod version
npm start

# start the interface
cd interface
npm install
npm run start
```

Instructions for configuring action automation script

The following is an example portion of a script for automating addition of flows to the floodlight controller. This specific example is a transparent redirect from any source pointing at the loadbalancer to a honeypot server. Important information needed here is the destination ip, mac address, and port of the honeypot, address of the switch, and ip of the loadbalancer. Upon swapping these fields with correct information, the follows will be automated any time a specific type of protocol matches.

```
src_data = pusher.get_rest_call (src_ip, 'GET')
src_parsedData = json.loads(src_data)
src_mac = json.dumps(src_parsedData['devices'][0]['mac'][0])
src_macAddress = src_mac.replace('"','')
src_port =
     json.dumps(src_parsedData['devices'][0]['attachmentPoint'][0]['port'][0])
src_port = src_port.replace('"','')

flow1 = {
           'Switch':"00:00:d2:bd:aa:aa:ca:56", (Switch address)
           "name":name,
           "cookie":"0",
           "priority":"32767",
           "in_port":src_port,
           "active":"true",
           "eth_type": "0x0800",
           "eth_src": src_macAddress,
           "ipv4_src": src_ip,
           "ipv4_dst": "192.168.1.50", (Load Balancer IP)
           "hard_timeout":"240",
           "actions":
                "set_field=eth_dst->12:c4:47:10:52:75, (Honeypot Mac)
                 set_field=ipv4_dst->192.168.1.46,     (Honeypot IP)
                 output=4"                             (Honeypot Port)
}
name += 1
flow2 = {
           'Switch':"00:00:d2:bd:aa:aa:ca:56", (Switch address)
           "name":name,
           "cookie":"0",
           "priority":"32767",
           "in_port":"4",
           "active":"true",
           "eth_type": "0x0800",
           "eth_src": "12:c4:47:10:52:75",              (Honeypot Mac)
           "eth_dst": src_macAddress,
           "ipv4_src": "192.168.1.46",                  (Honeypot IP)
           "ipv4_dst": src_ip,
           "hard_timeout":"240",
           "actions":"set_field=ipv4_src->192.168.1.50, (Loadbalancer IP)
                     output="+src_port
```

```
}
```

For an action consisting of blocking, the actions field can be removed from flow1 to push a static field blocking by the source IP, mac address, and port.

For an in-depth reference, you can look at this script in our repository: https://git.ece.iastate.edu/sd/sddec18-07/blob/master/snort/idp.py

5.2 Alternative Designs

We looked at using OpenDaylight and Open vSwitch to be our SDN controller software, but ended up settling on Floodlight because it fit our needs more than the other two. However, we did end up using Open vSwitch as part of our network stack.

Another option we tried was an alternative to using Snort. We started writing customized Floodlight modules to detect the malicious traffic, but we eventually decided to stop. The modules were difficult to integrate and did not work well even after spending countless hours on them. We decided it was too much work for its worth when we could use Snort to do the exact same thing and it was way more user-friendly. Snort is also the better choice because if our client expands on our project after we are finished, it is much easier to customize and expand.

Due to our choosing Floodlight as our SDN controller, we started using the interface to utilize Floodlight. This led to us really hating it, and deciding that we needed to create our own since we could use the Floodlight API that is accessible on our local instance. The decision to create a simpler and more performant interface wasn't made until about halfway through the first semester, so we had to change up our design a bit due to that decision.

5.3 Other Considerations

A couple times while testing DDoS attacks, we took down Andrew's entire home network which is what we used for testing.

5.4 Code

Link to repository with all code: https://git.ece.iastate.edu/sd/sddec18-07/tree/master