Career Explorations  
Computer Security  
Programming Lab 2

Aaaaaand we’re back!

Today your task is to explore more of the experimental side of computer science. We’ll be investigating different ways to cause routers to fail.

**Task 0: Login, Download and Install**

Sadly, your work from last time has been erased. To kick things off, you will need to repeat the first step from last time: install *Processing* and download the starter code. Here are the instructions from last time:

Download the starter code.  
Follow this link: [TODO Link]. A folder will be downloaded to your Desktop. Right-click on the folder and select “Extract”. You will see a folder entitled Main on your Desktop.

Download *Processing*.   
Follow this link: https://bit.ly/2GwpO4L. Again, a download will start automatically. Once it is downloaded, right click on the file and select “Extract”. This may take a few moments. You will see a folder entitled Processing on your Desktop.

Open Main with *Processing*  
Open *Processing*. You can do this by clicking on the start menu and typing *Processing* in the search bar. When the program opens, select File → Open, then click on “Desktop”. Select the file Main that you created previously, and click Open.   
A second window of Processing will appear. Close the original one. The new Processing window will have tabs called “Main”, “Lab\_1”, “Lab\_2”, “Lab\_3”, etc.

Change the labnumber in the main file to read 2, and select the Lab\_2 tab.

**Task 1: Experiment**

*You will not write any code to complete this task.*

*If you get stuck for more than 5 minutes on any Exercise in this task, discuss with a neighbor or an instructor for help!*

*If you completed this task yesterday, feel free to skip it!*

Once you’ve gotten *Processing* up and running, start the program by pressing the play button in the upper left corner of the window. You should immediately notice some similarities and differences from yesterday. Routers are indicated by a gray icon. Each has a vertical grey box to the side – its “queue”. The queue is the “lunch line” of packets waiting to be processed. At each step, each Router will process one packet from the front of its queue. When a packet arrives at the Router it will go to the back of the queue to be processed.

***Exercise 1a.*** Send packets around the network to get a sense of how queues work.

Routers can also fail! In our simulation, this is indicated by a Router turning yellow. Failed Routers cannot process additional packets – it’s like someone turned them off.

So when do Routers fail? When their queues are full and the Router receives a new packet, they have no place to put it and they will fail.

***Exercise 1b.*** Cause any Router to fail by filling up its queue.

***Exercise 1c.*** Cause *Router* 3 to fail

***Exercise 1d.*** Cause Router 3 to fail without sending a packet directly to it

***Exercise 1e.*** Bring down ONLY Router 3 without sending a packet directly to it

Once you’ve completed the exercises above, spend a few minutes to try the following exercise. If you can’t figure it out, feel free to move on to Task 2.

***Exercise 1f.*** (extra credit) Bring down ONLY Router 3 without sending a packet directly to it while making sure that Router 1 can still send packets to Router 0. Call over an instructor and have a discussion about why this is hard!

**Task 2: Denial of Service**

In this task you will program an EvilRouter that will target a single Router in the network and make it fail by sending too many packets for it to handle. This is called a denial of service attack or DoS attack for short.

In the simulation pane, you can turn a normal Router into an EvilRouter by double-clicking it. To target another Router, click first on the EvilRouter and then the Router that you wish to attack.

The EvilRouter has five commands that you can change. For now, you will only define the evilDoEveryStep command, which is executed by EvilRouters whenever the simulation moves forward a single step.

void evilDoEveryStep(Router target) {

// your evil code here   
 }

The target is the Router that the current EvilRouter should attack.

***Exercise 2.*** Write code between the curly braces of the evilDoEveryStep command to tell an EvilRouter how to attack a Router. Then run the program, create an EvilRouter, select a target and make sure that the target fails.

*Hint.* Routers process one Packet per step from the queue. How many should arrive per step for the queue to eventually fill up?

Once you’ve completed *Exercise 2*, you have a chance to do some real Computer Science. Try to think about other ways to design a solution. How many packets should you send at each step? 5? 10? 100? Can you think of other ways to bring down a Router? What might be the advantages of one approach over another?

**Task 3: Defend Against a DoS Attack**

Now, change the labnumber in the Main file to read 3 and switch to your Lab\_3 tab.

Your task is to defend against the attack that you just created! You will write a function called blockSender that determines when to block packets from a Router called sender. If the function returns true, then all packets from sender will be blocked. If the function returns false, then packets from sender will continue to be accepted. The current definition of blockSender is the following:

boolean blockSender(Router sender) {

return false;

}

This definition will never block packets. If you are not sure why, discuss with neighbor or call over an instructor for help.

***Exercise 3a.*** *Fill in the BlockSender method so that your DoS Attacks from Task 2 are ineffective.*

You will find the following methods useful in crafting your solution:

float portionOfQueueFrom(Router sender)

Writing the code portionOfQueueFrom(sender) will return a decimal number (often called floats in programming) describing the portion of the current Router’s queue that is filled by sender. For example, if 6 packets in Router 5’s queue come from a single sender, then portionOfQueueFrom(sender) will return 0.6 (recall that a queue has only 10 slots).

Writing the code return portionOfQueueFrom(sender) > 0.1 will return true if more than 1 packet in the queue is from sender, and will return false otherwise.

int amountInQueueFrom(Router sender)

Writing the code amountInQueueFrom(sender) will return the number of packets in a Router’s queue, which are from sender. This will always be a non-negative whole number, or integer, (i.e. 0, 1, 2, 3, … etc.). For example, if 6 packets in Router 5’s queue come from a single sender, then amountInQueueFrom(sender) will return 6.

Writing the code return amountInQueueFrom(sender) > 5 will return true if more than half of the queue is from sender, and will return false otherwise.

int amountInQueue()

Writing the code amountInQueue() will return the number of packets that are currently in the queue. This will always be a non-negative whole number, or integer, (i.e. 0, 1, 2, 3, … etc.). For example, if there are 8 packets in a Router’s queue from any number of senders, then amountInQueue() will return 8.

Writing the code return amountInQueue() > 5 will return true if the queue is at least half full, and will return false otherwise.

***Exercise 3b.*** *Modify your blockSender command so that it****only****blocks DoS attacks and not normal traffic. To ensure that blockSender isn*’*t blocking nice packets, try to send many (at least 4) packets from Router 1 to Router 2 at the same time. This should feel familiar to your solution to Exercise 1c*.

Now you can consider a more permissive defense approach. Perhaps a Router was only being over-eager to send non-malicious traffic (such as in Exercise 3b). Perhaps you accidentally blacklisted a node that wasn’t actually attacking you!

You will now decide when to unblock traffic from a sender by defining the unblockSender command. When unblockSender returns true, it changes the Router’s decision to blockSender and begins accepting packets from the sender once more. The current definition of unblockSender is as follows:

boolean unblockSender(Router sender) {

return false;

}

The current definition will *never* unblock a sender because it always returns false.

***Exercise 3c.*** *Try replacing*false*with*true*. What happens to the defense that you just implemented?*

Now, use the commands from the previous exercise to write code that returns true more thoughtfully, i.e. when you actually want to unblockSender. You will find it helpful to know that you can use the less-than sign (<) to compare numbers just as you did with the greater-than sign (>) in the previous exercise!

***Exercise 3d.*** *Implement the unblockSender command.*

***Exercise 3e.*** *Play with the thresholds that you set in blockSender and unblockSender so that DoS Attacks never bring down a*Router*, and over-eager*Routers*may be rejected at first, but if they slow down, they will be processed as normal.*

**Task 4: An Arms Race**

You have a defense (in your Lab\_3 tab) to an attack (in your Lab\_2 tab) that you just wrote! Your task now is to come up with creative ways to cause Routers to fail (in the Lab\_2 tab) and then defend against those creative attacks (in your Lab\_3 tab).

The EvilRouter has 4 commands where you can change the code. You’ve already seen how evilDoEveryStep works. There are also evil versions of the itsForMe, dontKnow and doKnow commands from the lab yesterday, called evilItsForMe, evilDontKnow, and evilDoKnow respectively. You can make changes to these commands while figuring out the answers to your questions.

If you’d like to see our solution to Lab\_1, simply scroll to the bottom of the Lab\_2 tab where it describes the UnsafeRouter. You may change this solution to modify the good Routers if you desire!

For now, try to be creative and do whatever you find interesting! You might, for example, implement one of the strategies of the Malicious Agents from our earlier activity, like refusing to forward packets. If you’re stuck on what to do, you can draw inspiration from some options we have provided below:

**Remember to Consult your Cheat Sheets to see what kinds of Commands are available to you!**

*Option 1. Spoofing for fun and profit*

You might try to implement spoofing, which is just a fancy term for “lying about who sent a packet”. To do this you will need to write

p.src = <insert your code here>;

before you forward the packet p along. This changes the sender (i.e. the src) of the packet. You will need to change some commands besides evilDoEveryStep for this to work.

Once you’ve implemented spoofing, try to defend against it! What strategies might you employ? This is hard, once you’ve thought about it for a few minutes, please ask for help!

*Option 2. Pretend to be a failed Router*

As you learned in task 1, failed routers do not forward packets through the network. They are black holes that receive everything and send nothing. This makes it difficult for other routers to send packets across the network! If you are evil, just pretending to fail can be an effective strategy.

How might you defend against this attack? Think about choosing alternate routes through the network.

*Option 3. Send only one packet on evilDoEveryStep*

Try to execute an attack on a router where you only send a single packet in evilDoEveryStep. One way to do this is to perform a Distributed Denial of Service attack or DDoS, which is where you send packets from multiple routers. You can do this in the interface (by creating multiple EvilRouters), or, even better, write code to “trick” other, normal routers into sending many packets to your target.

Honestly, I have no idea how to defend against this… Try out some solutions and see if they work!