Module 04: DDoS Attacks and Defences

Week 1: Amplification and Reflection Attack on the SCION Network

Christelle Gloor, Joel Wanner, Adrian Perrig

christelle.gloor@inf.ethz.ch, joel.wanner@inf.ethz.ch, adrian.perrig@inf.ethz.ch

November 2020

Welcome to the fourth module of this course, where we will work on network-level denial-of-service attacks using the SCION future internet architecture as a case study. This week, you will be on the offense and attempt to blast an unsuspecting victim with as much traffic as you can manage. Just like the SCION codebase, this module is written in Golang. If you are unfamiliar with Golang, there are very good resources online. For any concept of the language you can find *a tour of go* (1) explaining it. We recommend looking things up there as you encounter them in the code.

Overview and Problem Statement

You will be running an Autonomous System (AS) which contains a virtualized network shown in Figure 1, and you will be taking the role of the attacker. The VM we have distributed contains this AS, and you will be plugging it into SCIONLab (6), a research test-bed for the SCION internet architecture that is maintained by our group.

Your attack's vantage point has upstream connectivity, but the bandwidth on these links are low. Thankfully, you are aware of a "meow" server located in the same AS that it is connected via higher-bandwidth links. The victim of your attack is located in a different AS on SCIONLab and has the following SCION address:

```
17-ffaa:0:1115,[127.0.0.1]
```

Your task is to abuse the local server and stage a reflection-based amplification attack on the victim.

Part 1: Amplification

The server will serve varying amounts of data depending on the requests you send. Play around with it to find the payload which will result in the server responding with the most data.

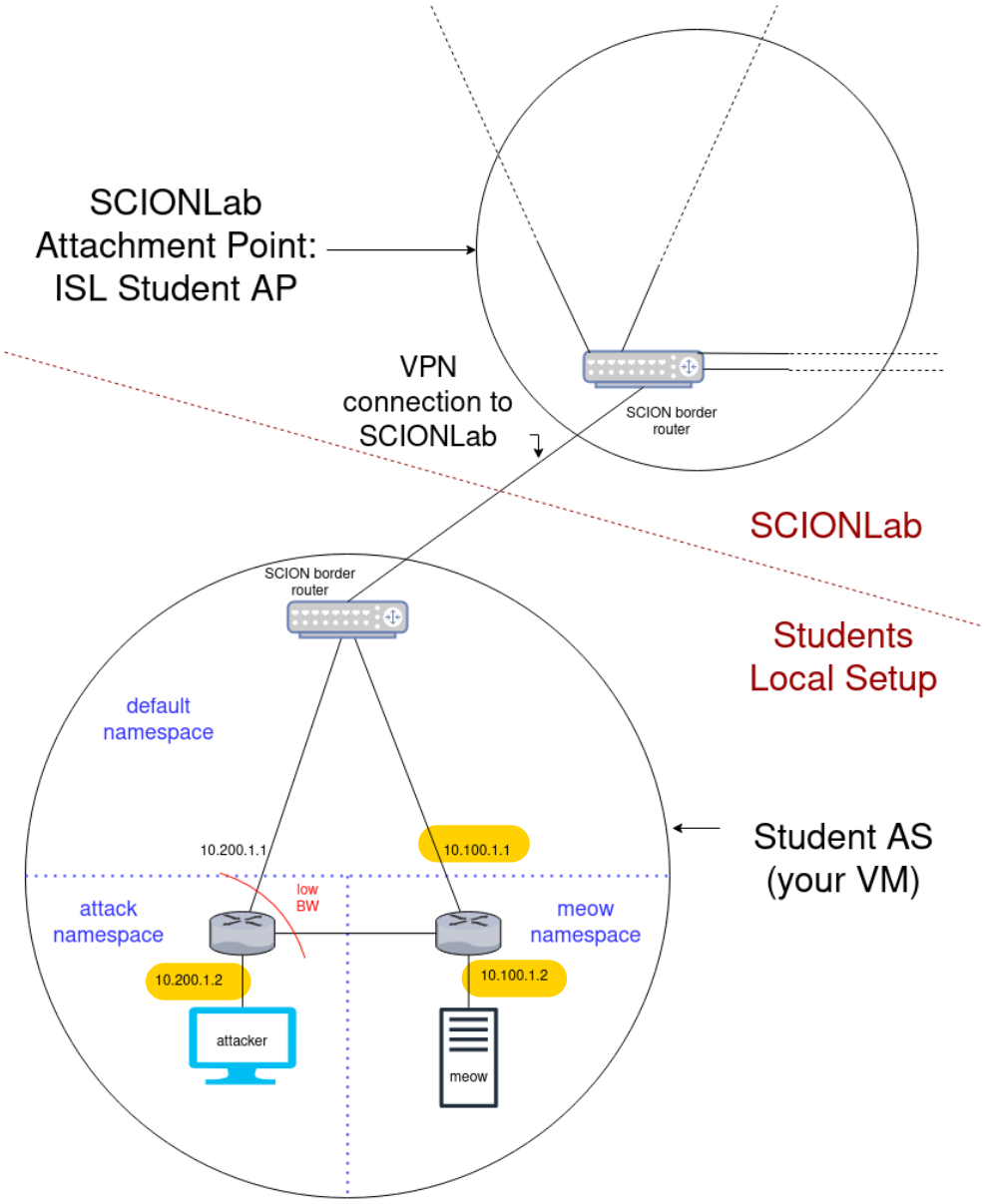

Figure 1: A snippet of the lab setup comprising your VM and the student attachment point (AP) in SCIONLab.

Part 2: Reflection

Now, you must manipulate your packets to trick the server into responding to the victim instead of back to you. To do this, you need to spoof the return address and the SCION path.

You have the possibility to execute this in two parts. First, you can execute an AS-local reflection and target an application that you start yourself in your VM. This will give you more immediately accessible feedback about the achieved level of amplification. In a second step, you can extend the reflection attack by directing it across the SCIONLab network to the victim located in a different AS.

Technical Setup

Our module will take place on the live SCIONLab network, which is a testbed for the SCION architecture.

To set up the lab environment, follow these steps:

1. Create a SCIONLab user AS
   (a) Navigate to the SCIONLab webpage at `https://scionlab.org` in your browser.
   (b) Register a new account with your ETH email.
   (c) Once you have logged in, click on *My ASes* and choose *Create a new SCIONLab AS*.
   (d) Enter any label you like (e.g. "ISL").
   (e) Select **SCION installation from packages**.
   (f) **Important:** choose the **ISL Student AP** as your attachment point and click the box **"Use VPN"**.
   (g) Copy the `scionlab-config` command in the second grey box above for later, as it will be used to configure your VM.
   (h) Confirm by clicking *Create AS*.
2. Install the VM
   (a) Download the OVA image from the course Moodle page.
   (b) Install VirtualBox on your machine. You are free to use other virtualization software, but we will only support VirtualBox. If at any point you encounter issues with your VM, please check the *Troubleshooting* document we have uploaded to Moodle.
   (c) Open the OVA in VirtualBox and import it. It should not be necessary to modify any settings in this process. You can see the settings that worked during testing in Figure 2.

(d) Start the VM. It may take a couple minutes to finish booting – you can tell it has finished when a login prompt appears.

(e) The VM has no GUI by default, and the standard VirtualBox command-line interface is annoying to use, we recommend `ssh` to access the guest machine from your host. The port is forwarded on your host to `:2222` with password authentication (user `vagrant`, password `vagrant`). You can use the following command to log in:

```
ssh -p 2222 vagrant@localhost
```

3. Configure the VM with your SCIONLab AS

(a) Paste the command you copied previously from the SCIONLab website in your VM:

```
sudo scionlab-config --host-id=... --host-secret=...
```

(b) Check the output from the script, everything should be set up now.

4. Shared folders

You may set up a shared folder such that you can program directly in your text editor or IDE on your host machine. Feel free to skip this step if you prefer to program directly on your guest.

Setting up a Shared Folder (optional):

(a) Set up a folder on your host machine that will be shared with the guest.

(b) Open the Virtualbox GUI and open the settings of the VM.

(c) Select "Shared Folders" $\mapsto$ "Add New Shared Folder"

(d) Browse for the folder you created in Step 4a to set the "Folder Path", choose any name for it, tick the box "Make Permanent", confirm and click "ok". You can see the settings in Figure 3.

(e) Login to your VM and mount your shared folder. The commands differ depending on your host OS.

Windows:

```
sudo mount -t vboxsf src ~/src
```

Linux / macOS:

```
sudo mount -t vboxsf src ~/src -o uid=1000,gid=1000
```

(f) Copy the contents of `~/dist` into `~/src`, which is the mount point for your folder:

```
cp -r ~/dist/* ~/src/
```

Now you can change the contents in this folder directly from your host machine. If you wish to use an IDE to code, we recommend installing Go version 1.13 to match the one installed on the guest VM. *Note:* in any case, you will only be able to run your code inside your guest.

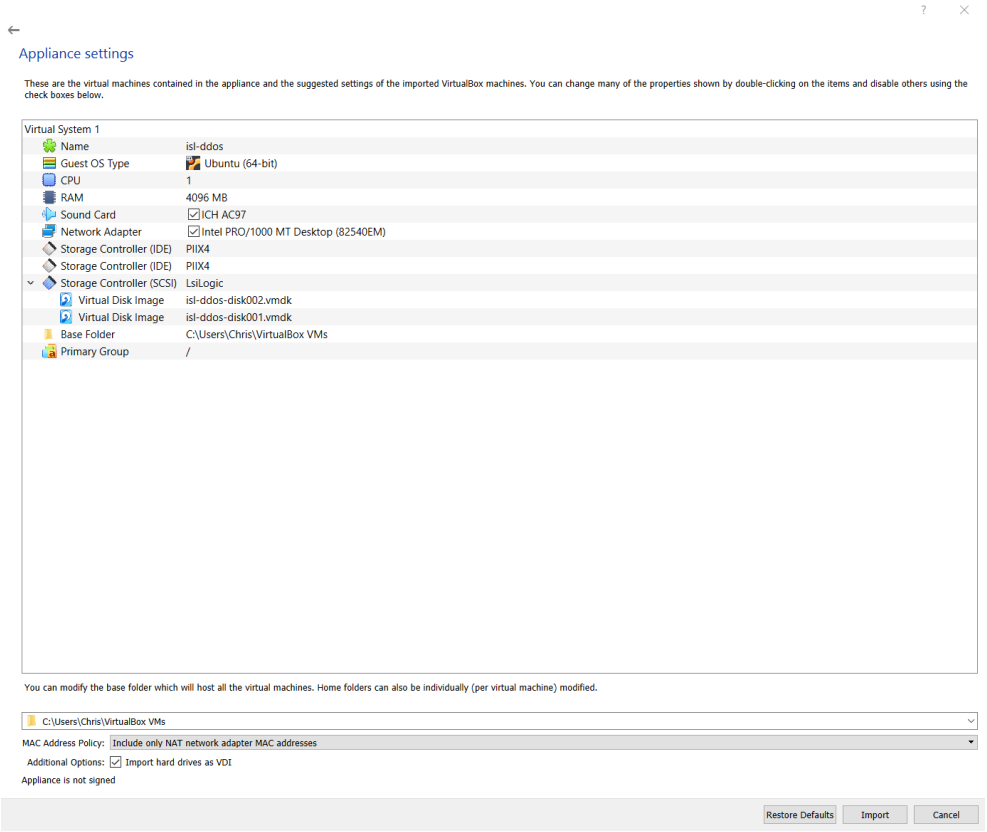

Figure 2: Default settings when importing your VM in Virtualbox.

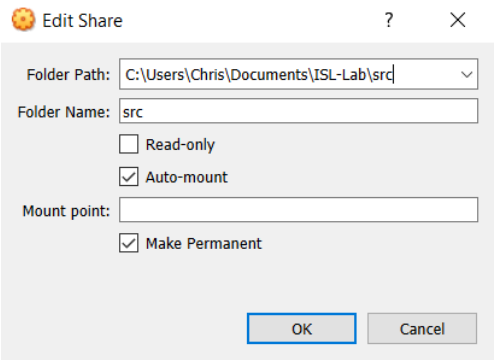

Figure 3: Settings when configuring a shared folder in Virtualbox.

A Tour of Your VM

Setup

Note: Do not change any of the configurations on your machine, especially those related to networking. We will be grading your code on an AS with the exact same configuration as the one distributed to you.

The VM image we have distributed contains the student AS shown in Figure 1. It includes the virtualized network and all the SCION services needed to communicate between the different parties and out to SCIONLab. These are implemented as `systemd` services and should always be up and running in your VM. You can check this by running:

```
systemctl list-dependencies isl.target
```

You should see 13 different services running, indicated by the green dot to the left of their listing. If you experience errors during the lab, its a good idea to check the services first. You can restart single services by directly referring to them, e.g.:

```
sudo systemctl restart customer.service
```

Or try to restart all of them with:

```
sudo systemctl restart isl.target
```

The virtual network consists of three different network namespaces. When you first start up your VM, you will be in the default namespace. Additionally, there is the *meow* namespace, where you will start the server from, and the *attack* namespace, where you will start your attack from. We have written some bash aliases for you to easily enter the namespaces. They are:

```
enter_meow_ns
enter_attack_ns
```

If your shell tells you that it didn't find the command, run:

```
source ~/.bashrc
```

Entering those commands will give you the view from the attacker or the meow machine shown in Figure 1. You can go back to the root namespace after entering one of these by simply calling:

```
exit
```

Interacting With the SCION Network

Your machine has some tools installed that are useful to explore the network topology. SCMP is a protocol similar to ICMP in today's Internet, and you can use the `scmp` command-line tool to ping and trace routes to hosts or ASes. You can check if the SCION connectivity is working by trying to ping the access point (the AS directly upstream of you) like this:

```
scmp echo -remote 17-ffaa:0:1114,[127.0.0.1]
```

Moreover, the graphical `scion-webapp` tools can be accessed by navigating to `http://localhost:8000` in a browser on your host machine.

Code Skeleton

The code you will be working with during the first week can be found in the `~/dist/attack/` directory (or `~/src/attack/` if you did the copying step from the shared folder setup).

- `meow/public.go` contains the API of the server. The documentation is kept sparse on purpose, and we will not answer any questions on the internal workings of the server. It is part of the task to experiment with different payloads and try to maximize the amplification.

  The requests comprise an ID, Query and Flags.

  Hint: Try to add the "metadata" flag to a request with the setters as a first step to raise your amplification.

- `client/example_client.go` contains a simple UDP client that sends a single request to the server and prints the servers answer to `stdout`. The function `GenerateClientPayload()` builds the request that is sent to the server. Feel free to change this code in order to try different things.

- `client/attack.go` is the **code skeleton for your attack and will be your deliverable for this week**. Make sure not to change the two function signatures!

- `client/attack_constants.go` contains some constants related to the lab needed by the various startup scripts among other things.

- `start_client.go` is a startup script that allows you to easily start either the example client or your attack. It already takes care of choosing the right addresses and passing it to the clients. Feel free to have a look at it to familiarize yourself a bit with Golang, and see how the client and attack get started. During grading, we will use this entry point to run your submissions. Also try:

  ```
  go run start_client.go -help
  ```

  Make sure to start your attack from the attack namespace! [1]

[1] You will still be able to invoke the attack from the default namespace. But the routing and bandwidth limitations will be off and so will the feedback you get from the victims. When grading, we will start your attack from the attack namespace. Do the same to get the most accurate feedback on how you are doing.

- `help/helpers.go` contains some helper code, mainly related to finding your local SCION address and exposing namespace-specific configurations. You may use these functions in your solution. To do this, add the following line to your imports in `attack.go`:

  ```
  student.ch/netsec/isl/attack/help
  ```

  and call the functions by the `help.foo` handle.

- `port.yaml` contains your personalized port. This is passed to the attack code. The remote victim will use this to multiplex its reports back to you after receiving your attack traffic. You can fetch your personalized port by executing

  ```
  isl port
  ```

  inside your VM. Set the port you get in the `port.yaml` file.

- `go.mod` is a Go-specific file related to linking dependencies. You must not modify it.

You are also given some binaries related to the task, which are located `/usr/local/bin/` and are in your `$PATH` variable:

- `start_meow`: The binary of the server you will abuse. We recommend starting it like this at first:

  ```
  start_meow -verbose
  ```

  This will make it log to the command line which will be helpful for debugging the early stages of your reflection attack. When you have managed the reflection, you can start it without this flag to speed it up. **Make sure to always start the server from the meow namespace!** [2]

  Hint: examining the binary might reveal some secrets.

- `local_victim`: This program will behave similarly to the remote victim, but you can start it in your AS in the default namespace for local testing. Make sure to start it in the same directory as the `port.yaml` file. To reach this program instead of the remote victim, start your attack with the `-spoof` flag but **without** the `-remote` flag.

[2] The server listens on a specific IP address only present in the meow network namespace. If you start the server anywhere else, it will no longer respond to your requests.

Part 1: Amplification

In general, you should start all the clients from the attack namespace and the server from the meow namespace. We will do the same when grading. To do this, multiplex your terminal (e.g. using tmux (3; 4), already installed in your VM) and run the aliases to switch to the meow and attack namespace respectively. Start the server in the meow namespace and invoke the example client in the attack namespace without passing it any flags. If all goes well, you should see the response of the server printed in your terminal on the attack side.

You may change the example client to try out different requests.

Once you find a payload which yields a satisfactory amplification, add this code to `GenerateAttackPayload()` in the attack skeleton.

Part 2 Reflection

For the second part of this exercise, you need to implement the reflection mechanism. In order to do this, it will be necessary to understand some of what happens under the hood when communicating using the SCION libraries. The example client uses the `appnet` library, which is a high level wrapper on top of the `SCION` library that abstracts away some of the details that are not relevant to simple applications.

But to implement the reflection, you will need finer access to the internals than `appnet` provides. We recommend using *delve* (2), a Go debugger for Golang similar to `gdb`, to trace the execution of the example client. Pay attention to the different steps that are carried out under the hood and figure out how to imitate this while changing only the parts you need for the spoofing.

You can get delve by running:

```
go get github.com/go-delve/delve/cmd/dlv
```

Navigate to the attack namespace and start the example client with delve like this:

```
~/go/bin/dlv debug start_client.go
```

When you see the `(dlv)` prompt next to your cursor, you have attached delve to the execution of your client.

Type:

```
b main.main
c
```

and you should have stopped in the main function. You can step over with `n` (next), step into functions with `s` (step), set another breakpoint with `b` (break) and continue to the next breakpoint with `c` (continue). For example, I can step through `start_client.go` by hitting `n` a number of times until I arrive at line 35, where the client is called. I can step into this function by hitting `s`. Now I switched to a different file. Right below your `s` command, you should see the description of where you are. First, you will see the function into which you just stepped, in relation to the module path (starting with `student.ch/...`). The second entry will be the path and line number. This is the important part for you, since you can use

this to set breakpoints. You can copy this line and use this as a breakpoint when you restart delve to jump there directly when using continue, e.g, after restarting delve:

```
b ./client/example_client.go:14
c
```

Stepping out of functions is achieved with `stepout`, and you can print the values and object structures of variables with `p` (print).

If you would like to use delve to examine your solution as well, you can pass the appropriate flags after two dashes like this:

```
~/go/bin/dlv debug start_client.go -- -spoof -remote
```

This should be enough for you to solve the exercise, but there are plenty of online resources if you want to learn more on how to use delve (5).

The SCION-related dependencies deployed on the VM can also be found here:

- `https://github.com/Cerenia/scion-apps/`
- `https://github.com/Cerenia/scion/`

It is a good idea to have a look at the files once you encounter them with delve to see what else you could use for your attack.

Testing

We have set up a visual Grafana interface for you to get feedback about your attack. You can access it from a browser on your host machine at `http://localhost:8001` when your VM is running. Navigate to the dashboards (sidebar on the left) and the *Attack Task* dashboard.

You should implement the reflection in two steps: AS-local, and remote. To trigger the local reflection, start `local_victim` in the default namespace and pass only the `-spoof` flag to `start_client.go`. If you are successful, the local victim will report the traffic it picked up to your Grafana dashboard.

The remote victim is already set up and listening to your traffic. To trigger a remote reflection, pass both the `-spoof` and `-remote` flags to `start_client.go`. **Don't forget to set your personalized port when you are spoofing the return address!** If you are successful, the remote victim will report back to your Grafana dashboard.

To seperate multiple runs of your attack, the victims have a cooldown period. The local victim will sleep for 5 seconds after an attack burst, and the `remote_victim`

will sleep for 10 seconds. Make sure not to start subsequent attacks without waiting for this amount of time or the feedback might only capture a slice of your traffic.

Keep in mind that the local traffic is bound to achieve a much higher spike than you would for the remote attack, since your packets never leave your machine and are not interfered with on the internet. Therefore, the first step is going to give you more realistic feedback on how good your payload was with respect to the amplification. But the second step, the remote reflection, is the main part of this task and you should focus on getting this to work. If you manage to get feedback to your Grafana interface from the remote victim, you are on a good track.

Grading

You can reach a total of 50 points during this exercise. 25 are distributed in relation to your achieved amplification and 25 are earned by implementing the remote reflection.

The local victim will give you points depending on the achieved amplification. The achievable bandwidth (when the services are started in the right namespaces) range from 60 to 2000kbps. It is normal that the bandwidth you achieve fluctuates, even for the same payload during the local reflection. When we grade, we will run your submission a number of times and pick the run which achieved the highest points. If you manage the remote reflection, you will get another 20 points. We will scale down the total available outbound bandwidth of the victim AS to your individually achieved attack traffic. If you attack would prevent the victim from any communication outside its AS, you will be awarded the last 5 points.

To account for the differences in performance of students machines when testing locally, we are publishing this *additional hint* for this iteration of the course: The maximum message size you can get out of the server is **888 characters** long.

References

[1] A Tour of Go. `https://tour.golang.org/`, 2020.

[2] dlv usage. `https://github.com/go-delve/delve/blob/master/Documentation/usage/dlv.md`, 2020.

[3] Chris Kuehl. tmux - a very simple beginner's guide. `https://www.ocf.berkeley.edu/~ckuehl/tmux/`, 2013.

[4] henrik. tmux cheatsheet. `https://gist.github.com/henrik/1967800`, 2011.

[5] James Sturtevant. Using the Go Delve Debugger from the command line. `https://www.jamessturtevant.com/posts/Using-the-Go-Delve-Debugger-from-the-command-line/`, 2019.

[6] Network Security Group, ETH Zurich. https://www.scionlab.org/, 2020.