



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Network Security Group, Department of Computer Science, ETH Zurich

Information Security Lab: Module 05

Setup and Attack Labsheet

TAs: Christelle Gloor, Felix Stöger

DDoS Attacks & Defenses

Fall 2022

Emails: christelle.gloor@inf.ethz.ch, felix.stoeger@inf.ethz.ch

Deadline: 05.12.22, 15:00 CET

Contents

1	Overview and Problem Statement	3
1.1	Reflection	4
1.2	Amplification	4
2	Technical Setup	5
2.1	Cloning the Lab Repository	5
2.2	VM Installation & Login	6
2.3	Configuring the Lab VM	9
3	A Tour of Your VM	12
3.1	General Setup	12
3.2	Interacting with the SCION Network	13
3.3	Quirks of working with VMs	14
3.4	Code Skeleton	14
4	Attack Task	18
4.1	Reflection	18
4.2	Amplification	19
4.3	Testing	19
4.3.1	Local Tests	19
4.3.2	Tests on the Grader	20
4.4	Grading	21
	Bibliography	24

1 Overview and Problem Statement

Welcome to the fifth module of this course. We will work on network-level denial-of-service attacks using the SCION future internet architecture as a case study. During the first week, you will be on the offense and attempt to direct as much traffic as you can toward a victim. Just like the SCION code base, this module is written in the Go language. If you are unfamiliar with Go, there are excellent resources online, such as *A Tour of Go* [7]. We recommend looking things up as you encounter them while getting familiar with the code-base and coding the assignment.

You will work in the following setup for the attack task:

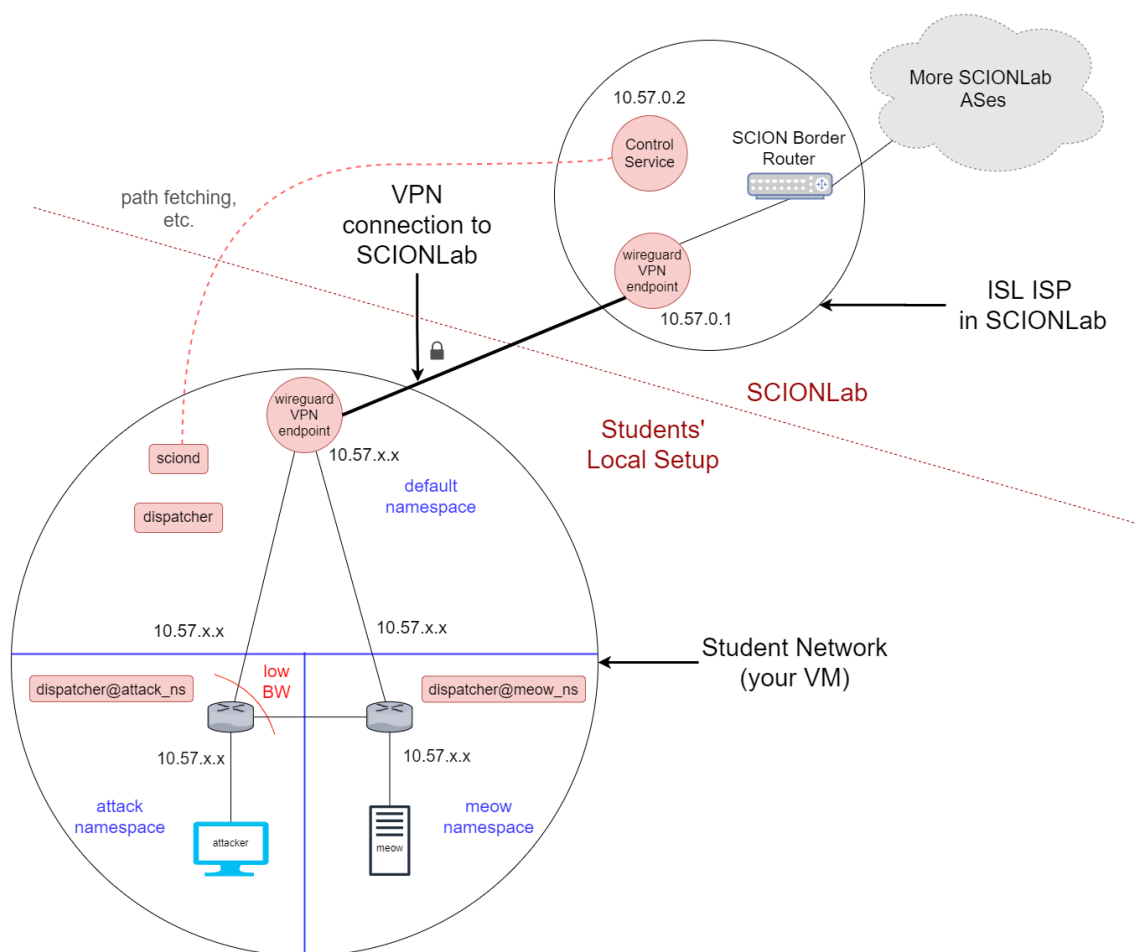


Figure 1.1: An overview of the local network topology for the first week.

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

`17-ffaa:0:1115,[100.0.0.1]`

Your task for the first week is to abuse the local server and stage a reflection-based amplification attack against the victim.

1.1 Reflection

In order to take advantage of the high-bandwidth link, you must manipulate your packets and make the server respond to the victim instead of you (the attacker). To achieve this, you need to spoof the return address and the SCION path in the packet headers.

You must execute this in two parts. First, you will stage an AS-local reflection and target a manually started victim application in your personal subnet. Second, you will extend the reflection by directing it across the SCIONLab network towards the remote victim located in a different AS.

1.2 Amplification

Once you have a working reflection, it is time to amplify your traffic. The server serves varying amounts of data depending on the request you send, and your goal is to find a request that causes the server to respond with the largest possible response (relative to the request size). The local victim will print a score, hinting at your achieved amplification.

2 Technical Setup

Our module will take place on the live SCIONLab network [19], which is a testbed for the SCION architecture running at ETH and across the globe. The following sections detail the steps for setting up your environment for this module.

2.1 Cloning the Lab Repository

Each enrolled student has a personalized GitLab repository, which contains the handout code, your assigned subnet and remote victim port, and your private key for the virtual private network (VPN) connection into SCIONLab. You can access it by logging into the D-INFK GitLab instance¹ with your ETH credentials. If you are unfamiliar with Git and/or GitLab, we recommend some basic resources [1, 2].

Clone your personal repository somewhere on your host machine. Unless you have previously set up ssh keys on your ETH Gitlab, we recommend cloning via

¹<https://gitlab.inf.ethz.ch>

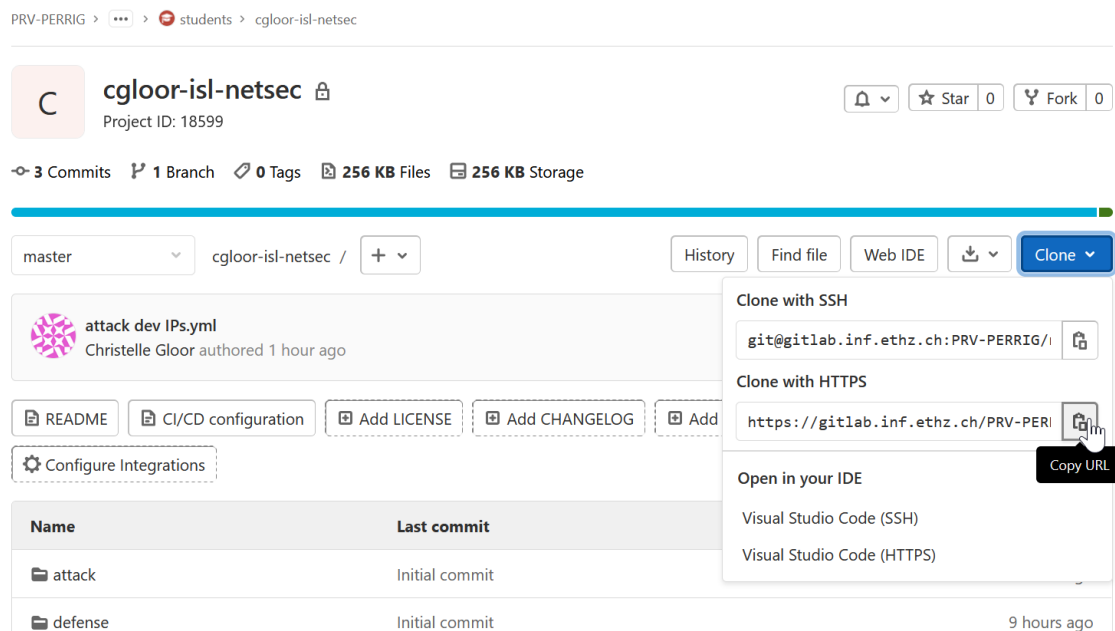


Figure 2.1: You can clone your repository with *SSH* (requiring an SSH keypair in your GitLab profile) or via *HTTPS*, which will simply prompt you for your ETH credentials.

https, which will prompt you for your ETH credentials:

```
git clone <your_isl_netsec_repo_URL>
```

You can find the URL by expanding the blue “Clone” button on the root page of your isl-netsec repository (Fig. 2.1).

Cloning the repository on the host machine (your personal computer) instead of the guest (the lab VM) has the advantage that you can use an Integrated Development Environment (IDE) or text editor of your choice with minimal setup.

Note: the code will not be executable on your host due to the lack of SCION services, and you will always need to run tests inside the VM.

2.2 VM Installation & Login

1. Install VirtualBox on your machine, if you have not done so already. You are free to use other virtualization software, but the TAs will only provide support for VirtualBox.
2. Download the OVA image from the Moodle course page. If at any point you encounter issues with your VM, please check the *Troubleshooting* document we have uploaded to Moodle [12]. It will be continuously expanded if new issues arise.
3. Import the OVA in VirtualBox. It should not be necessary to modify any settings during this process. You can see the settings we have tested in Fig. 2.2.
4. Run the VM (Fig. 2.3). It may take a couple of minutes to finish booting, after which a login prompt will appear in the VirtualBox preview interface (Fig. 2.4).
5. The VM has no graphical user interface (GUI) by default and the VirtualBox command-line interface is somewhat clunky. For this reason, we recommend using *ssh* [11] to access your guest machine from your host, giving you the freedom to choose the terminal emulator you are most comfortable with. The SSH port is forwarded on your host to `localhost:2422` and uses password authentication (user `vagrant`, password `vagrant`). While the VM is running, use the following command to log in:

```
ssh -p 2422 vagrant@localhost
```

You will then be logged in as `vagrant` on a *bash* [8] shell in the default network namespace of your VM.

Note that during this step, you may see a *Remote host identification has changed* warning. This is expected if you have previously run VMs from the other

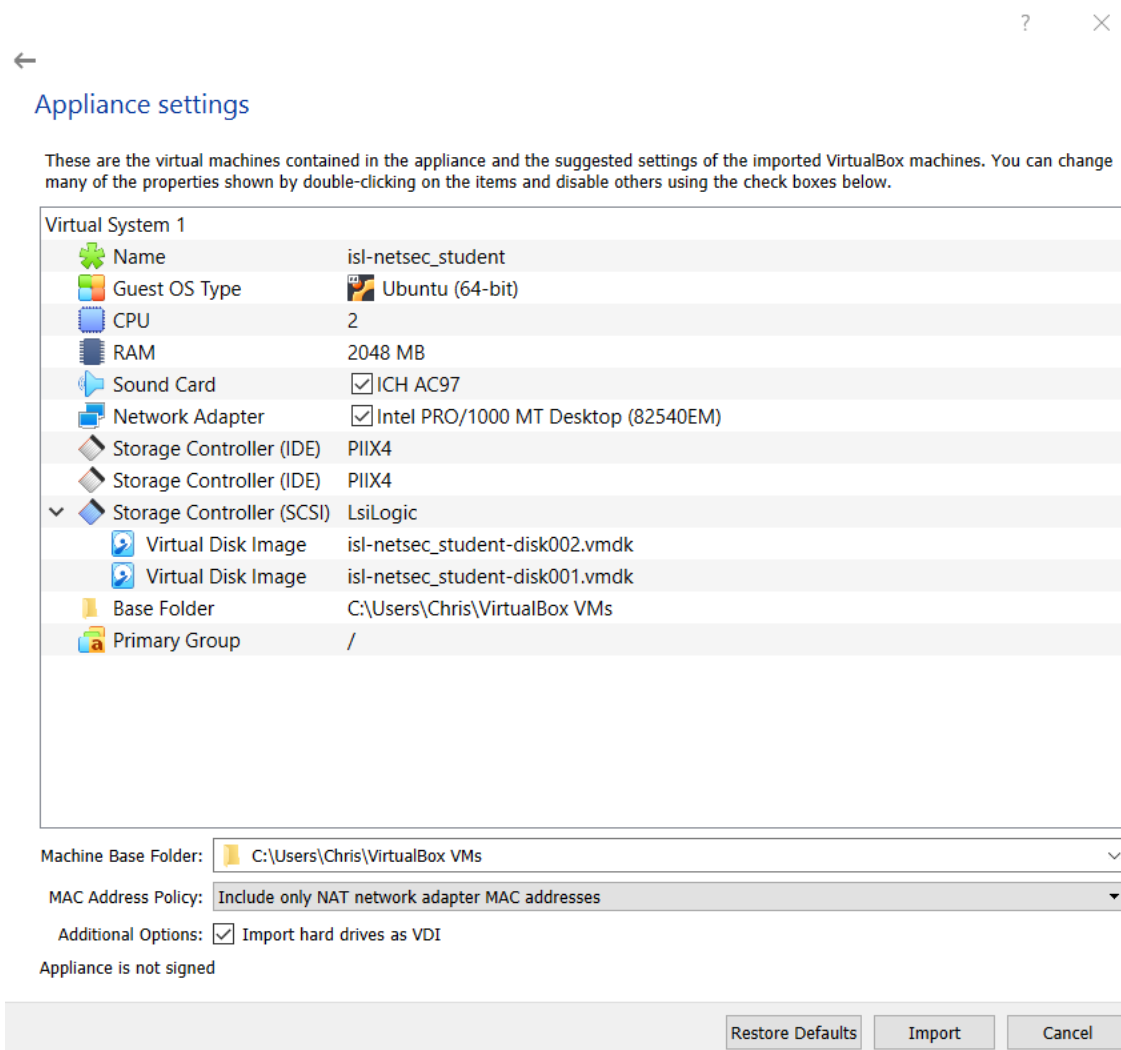


Figure 2.2: The settings of the virtual appliance when importing it to VirtualBox.

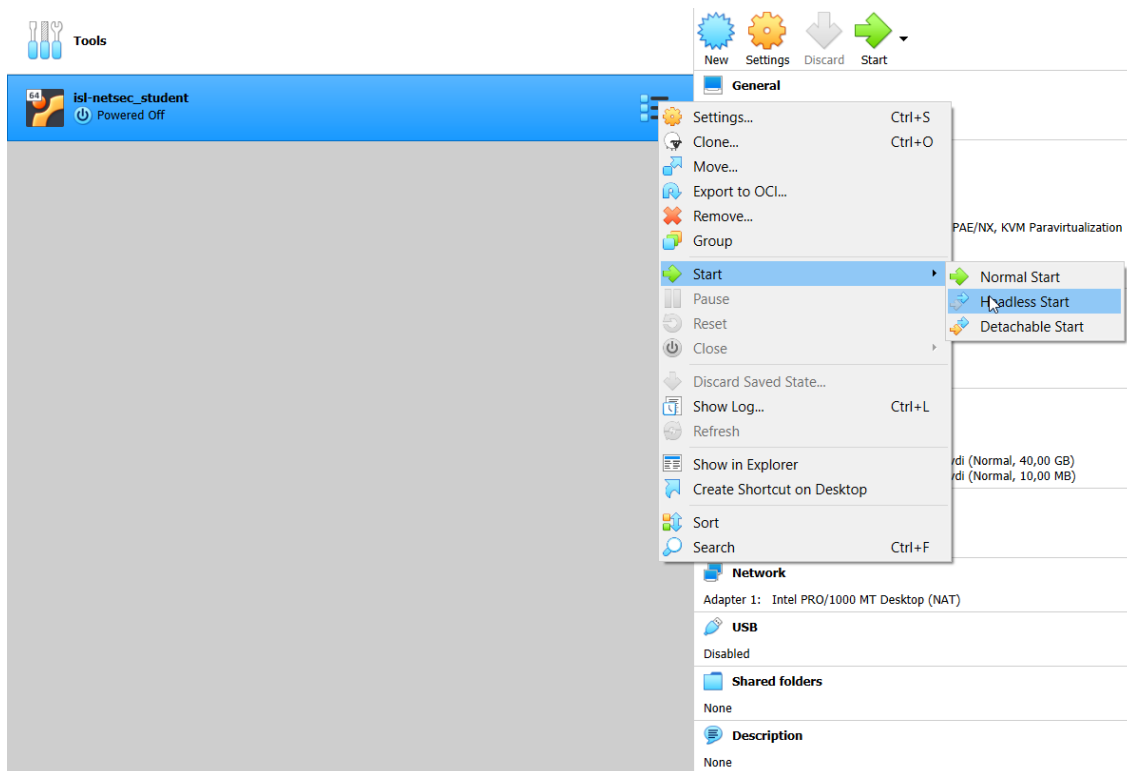


Figure 2.3: Running the VM in headless mode. You can later log into it using a terminal of your choice.

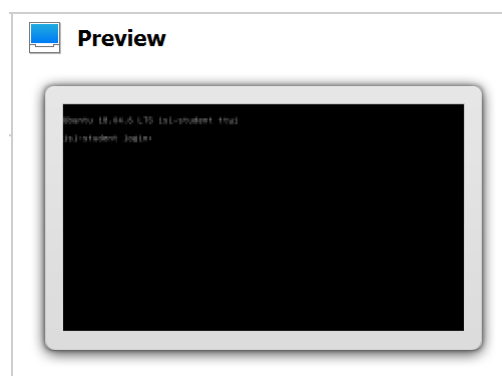


Figure 2.4: If the text in the preview looks like 2 lines and is no longer changing, the VM has booted.

modules on your machine, and the error can simply be fixed using the instructions provided in the message.

2.3 Configuring the Lab VM

Note: If you are using a **cloud VM** instead of a local instance, please refer to the “cloud_setup.md” document which replaces section 2.3 of this sheet.

Note: If you do not want to set up a shared folder, you may alternatively use, for example, VSCode’s “Remote - SSH” plugin (or any other equivalent). This will come at a slight performance degradation but may be less error prone. Please refer to the instruction video posted in the module 05 section on moodle if you’d like to set up the VSCode plugin instead of a shared folder.

Shared folder Setting up a shared folder allows you to seamlessly access the code you are working on in your host from your guest: *Note: You **do not** need to shut down your VM to edit the shared folder settings.*

1. Open the VirtualBox GUI and navigate to the VM settings.
2. Select “Shared Folders” \mapsto “Add New Shared Folder” (top-right Folder icon with a green plus)
3. Browse to the **root** of the repository you cloned and select this as the “Folder Path”.
4. **Important:** Set the “Folder Name” to `src` (it is already set to whatever the folder was called automatically when you select it, you **must** edit this) and `/home/vagrant/src` to be the “Mount Point”.
5. Check the box “Make Permanent” and click “OK” twice (the previous steps can be seen in Fig. 2.5).
6. From your guest: mount the shared folder. We have preconfigured the appropriate command in your VM, it differs depending on your host operating system:

Windows: `win_mount_src`

Linux or Mac: `unix_mount_src`

Note: This will only work if you have followed the naming convention while creating the shared folder. You will need to re-run the mount command after every reboot of your VM.

7. `cd` into the `~/src` directory to check that your code is now also accessible in the VM (your `IPs.yml` file should now be located at `~/src/IP.yml` in your VM).

If you were located in the `~/src` directory when calling the mount command, you

will have to change to a different directory and move back into the ~/src directory to see the mounting take effect.

Setting up your network and connectivity to SCIONLab. After following the previous steps, you can now run the provided configuration script which will personalize your VM. Simply run the following command on the guest:

```
source configure_isl
```

On success, the script will create a file `isl_configured` in the vagrant home directory. This file is checked by certain services at runtime. **Do not delete this file unless you intend to reconfigure your VM.** If the script runs through and does not report any errors, you are now all set up to start solving the task.

Note: Do not change any of the configurations on your VM, especially those related to networking. We will be grading your code on a VM with the exact same configuration as

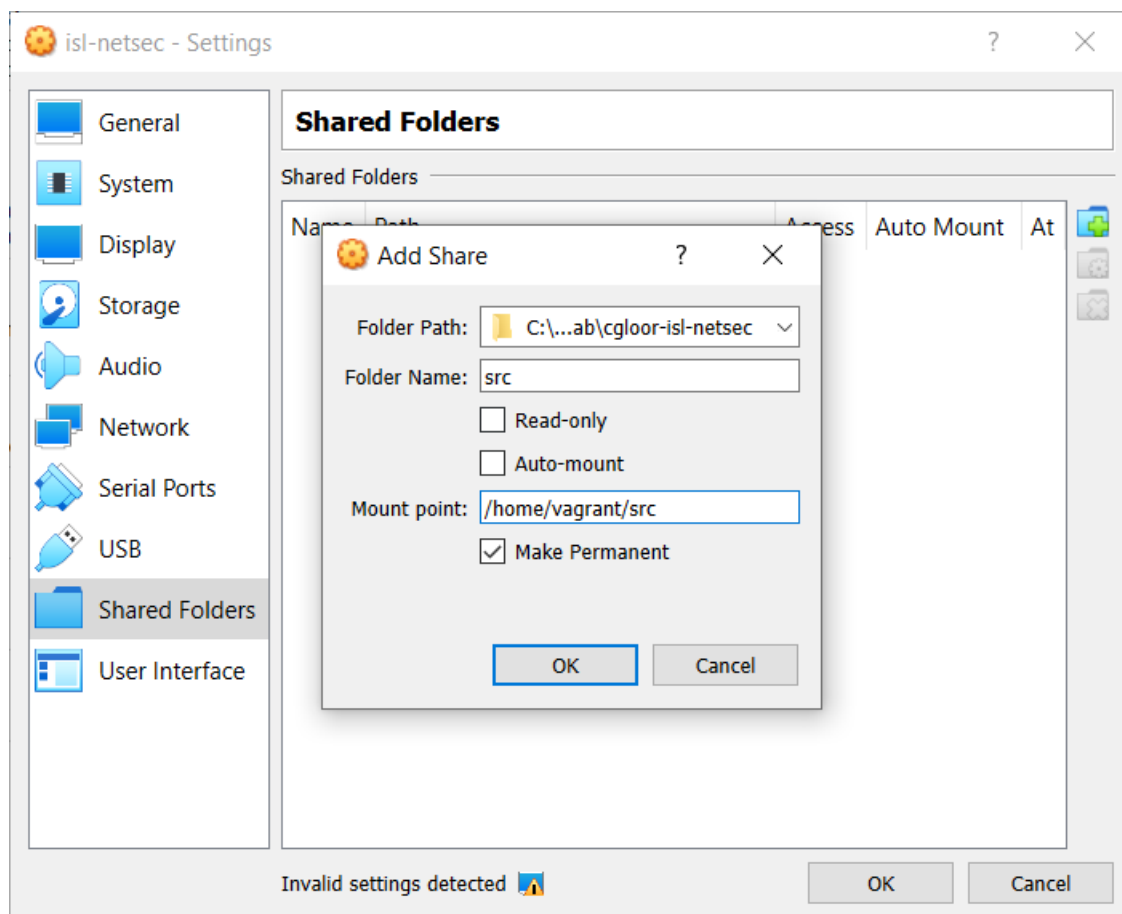


Figure 2.5: While enabling the shared folder, point its root at your cloned repository. Make sure to give it the name `src` or the mount scripts we provide for you inside your VM will not work.

the one distributed to you. Fiddling with the configurations will distort the feedback you get locally compared to the final grading environment.

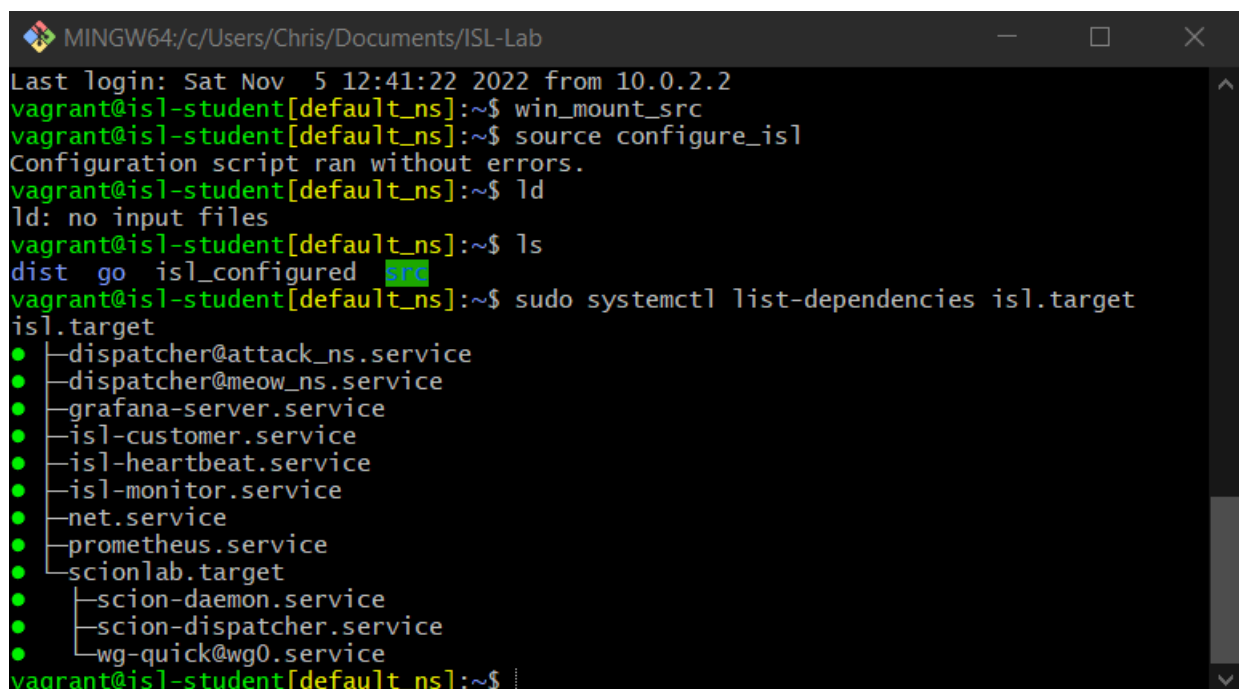
3 A Tour of Your VM

3.1 General Setup

The VM contains the setup shown in Fig. 1.1. It includes the virtualized network, some helper services responsible for visualizing your traffic, and all the SCION services of the end-host stack needed to communicate between the different parties and out to SCIONLab. They are implemented as systemd [18] 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 12 different services running, indicated by the green dot to the left of their listing (Figure 3.1)



```
MINGW64:/c:/Users/Chris/Documents/ISL-Lab
Last login: Sat Nov  5 12:41:22 2022 from 10.0.2.2
vagrant@isl-student[default_ns]:~$ win_mount_src
vagrant@isl-student[default_ns]:~$ source configure_isl
Configuration script ran without errors.
vagrant@isl-student[default_ns]:~$ ld
ld: no input files
vagrant@isl-student[default_ns]:~$ ls
dist go isl_configured src
vagrant@isl-student[default_ns]:~$ sudo systemctl list-dependencies isl.target
isl.target
● | dispatcher@attack_ns.service
● | dispatcher@meow_ns.service
● | grafana-server.service
● | isl-customer.service
● | isl-heartbeat.service
● | isl-monitor.service
● | net.service
● | prometheus.service
● | scionlab.target
● | scion-daemon.service
● | scion-dispatcher.service
● | wg-quick@wg0.service
vagrant@isl-student[default_ns]:~$
```

Figure 3.1: A successfully configured fresh VM in a Windows host. The shared folder was mounted to the `/home/vagrant/src` directory successfully, the configuration ran without errors, and all the systemd services have a green dot to their left.

If you experience issues during the lab, you should first run `isl status` on the guest to perform a series of health checks. The script will automatically attempt to repair minor problems (e.g., a service that is not running). If you experience persistent issues, please refer to the “Troubleshooting” [12] document on Moodle and include the relevant output when filing an issue report.

The virtual network comprises three different network namespaces. When you first log into your VM, you will be in the default namespace. The *meow* namespace is where you must start the server from and the *attack* namespace is where you must start your attack. We provide you with simple commands to switch your view from the default namespace:

```
goto-attack, goto-meow
```

Note: Do not nest these commands. If you want to switch namespaces, always call `exit` first (or press <Ctrl+D>) to return to the default namespace before switching.

Switching namespaces provides you with the view (and bandwidth conditions) of the server or the attacker.

Note that you also have a folder called `dist` in the vagrant home directory. It contains a fresh copy of the task skeletons for your reference.

3.2 Interacting with the SCION Network

The `scion` command ports some of the typical network diagnostic applications to the SCION protocol and adds some of its own. You can display all the possibilities by running:

```
scion help
```

The most useful applications for this task will be `ping`, `showpaths`, and `traceroute`. For example, you can ping the remote victim by running:

```
scion ping 17-ffaa:0:1115,100.0.0.1
```

You can browse through the code if you are interested to see how these applications were ported to the SCION protocol stack [3, 4, 5].

You may also be familiar with the network protocol analyzer Wireshark [14] and its command-line interface `tshark` [13]. We have pre-installed the `tshark` command line utility with a SCION dissector plugin. This enables you to capture and analyze SCION traffic on your WireGuard VPN interface:

```
sudo tshark -i wg0 -Y 'scion'
```

You can also narrow down the traffic by filtering for, e.g., traffic arriving from the

victim AS (17-ffaa:0:1115):

```
sudo tshark -i wg0 -Y 'scion.src_isd == 17 && scion.src_as == "ffaa:0:1115"'
```

The commands will print a warning since we are using `sudo` (which is not optimal, but simpler to set up), but you can safely ignore it. *Note: You will not see anything beyond that the capturing has started on `wg0` after running this command and nothing else, since no traffic from this AS is directed towards you in the idle state.* We recommend browsing through the dissector file to see what options are available to you. The file is located in the following directory on your VM:

`/usr/lib/x86_64-linux-gnu/wireshark/plugins`

3.3 Quirks of working with VMs

While you work on the tasks throughout the next few weeks, there will be periods where you will not want to run your VM. You can either shut down or pause your VM in Virtualbox. As already mentioned previously, **rebooting** your VM may require you to re-run the command to mount your shared folder.

If you instead decide to **pause** your VM, the system time on the guest might become out of sync. This will affect SCION as well as your Grafana dashboards. After unpausing your VM, check the time with this command:

```
timedatectl
```

If the time is off, you can reset it by running:

```
sudo systemctl restart systemd-timesyncd
```

3.4 Code Skeleton

General Some of the files in the root of your repository are used for **both weeks** of our module:

- `IPs.yml` contains your personalized configuration set by the `configure_isl` command. **This file is accessed in various places by the code skeleton. If you change or delete it, your setup will break.**
- `go.mod` is a Go-specific file that contains dependencies. Note that when you run your code for the first time or add imports to it, Go will make modifications to it, which should be committed to Git. However, it will not be necessary to modify this file manually.

- `privatekey` is your cryptographic key used for the VPN connection into SCIONLab. If you delete or modify it, your connection to SCIONLab will break. As the name suggests, this file must be kept secret.

Attack task The code you will be working with for **the first week** is located in the attack folder.

- `server/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 consist of an **ID**, a **query**, and some **flags**.
- `client/example_client.go` contains a simple User Datagram Protocol (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 its implementation in order to experiment with different payloads.
- `client/attack.go` is the **code skeleton for your attack and will be your deliverable for the first week**. Make sure not to change the two function signatures! *If you do by accident, you can always revert your change using Git or refer to the fresh skeleton stored in the `dist` folder.*
- `client/help.go` contains some constants and helper methods related to the lab needed by the various startup scripts, among other things.
You must use these functions in your solution as well. **Do not hardcode anything you can generate with the helpers!** If your code breaks on the grader because you did not follow this instruction, we will not adjust your code in retrospect, even if your solution would have otherwise achieved full points.
- `client.go` is the main file, which allows you to easily start either the example client or your attack. It already takes care of choosing the right parameters and passing them to the client implementations. Feel free to have a look at it to familiarize yourself with Go, and to see how the client and attack are invoked. During grading, we will use this entry point to run your submissions. Also try:

```
go run client.go -help
```

Make sure to start your *attack* from the *attack* namespace!¹

Defense task The code you will be working with for **the second week** is located in the defense folder. The second lab sheet will contain more information about these files, you will not need to touch them for now.

Binary executables In addition to the Go code in your repository, you will work with a few binaries that come preinstalled in your VM:

- **meow** is the UDP server you will be abusing. We recommend passing the **verbose** flag when you start it at first.

`meow -verbose`

This will make the server log to the command line, which can be helpful for debugging the early stages of your reflection attack. Once your reflection succeeds, invoke the program without this flag to speed it up.

Make sure to start the *server* from the *meow* namespace!²

- **victim** is a binary that behaves similarly to the remote victim application, but you can start it in your local network.

Make sure to start the *victim* from the *default* namespace!³

Note: The `-remote` flag of the client application determines if you are targeting the local or remote victim.

Navigating the terminal In order to start the binaries in the different namespaces at the same time, you could open multiple instances of your terminal of choice in your host and log into your guest multiple times.

Alternatively, we have installed and pre-configured *tmux* in your VM. *tmux* is a popular terminal multiplexer that allows you to run multiple shell sessions in a single window. We recommend checking a beginner's guide [15] to learn about the most important commands.

*Note: We have written some shell scripts which automatically install certain environment variables and change the appearance of your command prompt, such that you can navigate the virtualized network more easily. The **tmux** binary specifically is wrapped, which might lead to some unexpected behaviour if you are already familiar with the terminal multiplexer. They are located at `/usr/local/isl/shell` in your guest. Feel free to have a look if you*

¹You will still be able to invoke the attack from the default namespace, but the routing and bandwidth limitations will not be correct, and therefore, the feedback from the victims will be inaccurate.

²The binary will happily start from anywhere, but since the IP address assigned to it only exists in the meow namespace, you will not be able to communicate with the server if it is started outside the meow namespace.

³Again, the binary will happily start from anywhere, but you might not be able to reach it or run into IP address clashes if you start it from within the meow or attack namespace.

are curious. If you feel like these scripts are in your way, and you know what you are doing with the shell, you are free to modify or delete them. But beware that some things might break if you are not careful.

4 Attack Task

4.1 Reflection

In order to implement the reflection, you will have to gain an understanding of what happens under the hood when communicating using the SCION libraries. The example client uses the `pan` library, which is a high-level wrapper on top of the SCION stack, which abstracts away many of the details that are not relevant to simple applications.

In order to implement the reflection, you will need finer access to the internals than `pan` provides. There are multiple ways to familiarize yourself with a codebase.

Command-line debugging We recommend using `delve` [9] to trace the execution of the example client. `delve` is a Go debugger, similar to `gdb` [16] for C, and it uses similar commands. Tracing the example client will allow you to see the concrete function calls used when sending SCION packet, which can be more efficient than blindly reading an unfamiliar repository.

To start debugging, navigate to the attack namespace and start the example client with `delve`:

```
dlv debug client.go
```

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

```
b main.main
c
```

the execution will stop 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 `client.go` by hitting `n` a number of times until I arrive at line 24, 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 will find a description of where you are. First, you will see the function into which you just stepped, in relation to the module path (starting with `ethz.ch/netsec/is1...`). The second entry will be the path and line number. The second entry can be copied and used to set breakpoints. For example, after restarting your `delve` session, you

can do this:

```
b ./client/example_client.go:19
c
```

to directly jump to the beginning of the `GenerateClientPayload` function without having to step through every line of the main function first. 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:

```
dlv debug client.go -- -sploof -remote
```

These basic instructions should be sufficient for you to solve the exercise, but there are plenty of online resources if you want to learn more on how to use delve [17] and command line debuggers in general.

Documentation and source code You can browse through the documentation of the SCION codebase [6]. *Note: the Code deployed on SCIONLab [20] may differ slightly from the upstream SCION codebase. If you encounter unexpected behaviour, it is a good idea to double check if the functionality you find in the upstream repository is identical on SCIONLab.*

Much of the SCION codebase follows the networking conventions in IP networking libraries. If you cannot find much documentation for a particular function, it might be useful to search for an equivalent in Go libraries [10].

4.2 Amplification

As mentioned before, start the clients from the attack namespace and the server from the meow namespace. In a first step, you can invoke the example client with different requests and inspect the answers from the server that will be printed to the terminal. Once you find a payload which yields a satisfactory amplification, add this code to `GenerateAttackPayload()` in the attack skeleton.

4.3 Testing

4.3.1 Local Tests

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:8002` when your VM is running. Navigate to the dashboards (sidebar on the left) and the *Attack Task* dashboard. The Grafana interface can be a bit peculiar, so when selecting the dashboard for the first time, you may need to navigate to the “General”

folder found in the “Browse” tab of the “Dashboards” menu item (4 squares on the left side of the page) to find the boards for the specific tasks initially. After opening the dashboard, you also will have to set the refresh rate (top right circular pair of arrows in the dashboard) to continuously scrape updates from your guest.

You must implement the reflection in two steps: AS-local, and remote. To trigger the local reflection, start `victim` in the default namespace and pass only the `-spoof` flag to `client.go`. If you are successful, the local victim will report the traffic it picked up to your Grafana dashboard. Additionally, it will print a score on the command line, depending on your achieved amplification.

The remote victim is already set up in our infrastructure and listening for your traffic. To start a remote reflection attack, pass both the `-spoof` and `-remote` flags to `client.go`. **Don’t forget to set your personalized port, which can be loaded with the helpers, when you are spoofing the return address!** If you are successful, the remote victim will report back to your Grafana dashboard.

To separate multiple runs of your attack, the victims have a **cooldown** period per student. Both victims will sleep for 10 seconds after an attack burst. Make sure not to start subsequent attacks without waiting for this cooldown to expire, or the feedback might only capture your attack partially and distort the measurements.

Keep in mind that the local traffic is expected to achieve higher bandwidth than your remote attack, since your packets never leave your machine and are not subject to Internet quality of service. Therefore, a local test will give you more realistic feedback on your choice of payload with respect to the amplification. We would like to stress that the remote reflection is the main part of this task and you should focus on completing this objective. If you manage to receive feedback to your Grafana interface from the remote victim, you are on a good track.

You may test any of the subtasks of both weeks on your personal machine as often as you like.

4.3.2 Tests on the Grader

The amplification task and the points returned by the local victim will be dependent on individual machine performance. For this reason, we offer the possibility to test your code on the grading machine in order to obtain accurate feedback on your performance. You may trigger such tests via the GitLab CI/CD web interface.

From the root of your personal GitLab repository, navigate to “CI/CD” → “Pipelines”. To the right of your commits, by expanding the “play” button, you can start a run of any subtask on the grader (Fig. 4.1). This will add your job to the FIFO queue of our grader (Fig. 4.1). To inspect the status and output of jobs, click the button left of the commit, or navigate to the “Jobs” page.

Since there are ~300 students in this course, there will naturally be contention for compute time on the grader. You will be granted 100 total credits independent of the subtask you are testing on the grading machine, in order to discourage monopolizing the grader with an excessive number of jobs.

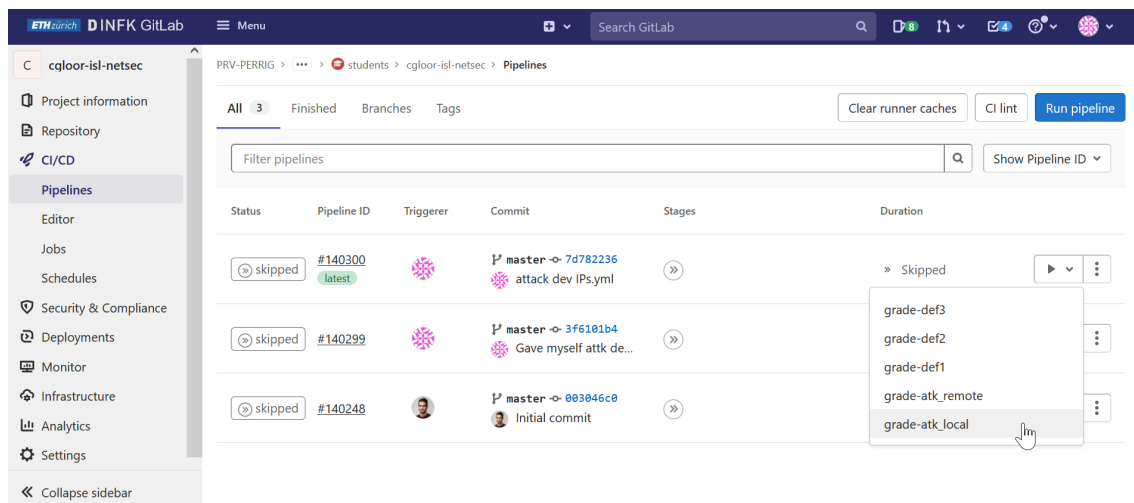


Figure 4.1: This is where you can start a run of your code on the grading machine.

Every test you run after you have used up your credits will result in **one point being subtracted from the final score** achieved for this module.

We recommend starting to solve the module early, as the contention on the grading machine is likely to grow as the deadline approaches. There are **no guarantees** about receiving feedback from the grading machine if you submit your code tightly before the deadline.

The output of your job (Fig. 4.2) will provide you with the score achieved by your code. For your reference, before every test on the grader, the grader will report back the number of credits you have left, or how many points you have already lost through overspending.

4.4 Grading

You can obtain a total of 50 points for the attack task:

- **22 points** are awarded based on your amplification.
- **22 points** will be granted if you reach the remote victim through your reflection, and
- **6 points** will be granted if you manage to clog the remote victim communication. Since all students are sharing the remote victim, it calculates this on a per-student basis, with the assumption that the victim's total outbound bandwidth is equal to the bandwidth achieved by your attack. These 6 points are therefore independent of your amplification. The gauge on your Grafana dashboard gives you an indication of how well you are doing—the closer to 100% you can get, the better.

```

1 Running with gitlab-runner 15.1.0 (76984217)
2   on netsec-hpc-zapdos VSX_kodE
3 Preparing the "ssh" executor
4 Using SSH executor...
5
6 Preparing environment
7 Running on isl-student via netsec-hpc-zapdos...
8
9 Getting source from Git repository
10 Fetching changes with git depth set to 20...
11 warning: templates not found builds/VSX_kodE/0/PRV-PERRIG/netsec-hpc-zapdos
12 Reinitialized existing Git repository in /home/vagrant/builds/VSX_kodE/0/PRV-PERRIG/netsec-hpc-zapdos
13 Checking out 57bead5d as master...
14 Removing client
15 Removing go.sum
16 Skipping Git submodules setup
17
18 Executing "step_script" stage of the job script
19 $ /bin/bash --login isl-grade atk local
20 [1] Performing CI file integrity check... OK
21 [2] Checking Go code... OK
22 =====
23 Deducting a credit from your allowance...
24 97 credits remaining.
25 =====
26 [3] Starting grade run
27 VICTIM: Listening on 10.57.0.132:60000
28 VICTIM: Attack no 1
29 VICTIM: Number of packets: 2133
30 Number of bytes: 1845045
31 Achieved bandwidth: 1476.036000 KBps
32 0% of victim communication was blocked.
33 =====
34 Score: 16/22 (73%)
35 =====
36 Local attack, waiting another 15s before exiting.
37 Cleaning up project directory and file based variables
38 Job succeeded

```

Figure 4.2: The output of a local attack run on the grader. Be aware that even if no attack is detected, and no points are awarded, the job will still succeed without an error. Such a run would yield 0 points during grading.

Your most recent commit on the master branch of this repository will be considered for the grading. Since there is a certain amount of nondeterminism expected for certain tasks, we will rerun your solutions in the week after the lab in a more controlled environment. We will execute 6 runs of your solutions for each task, discard the lowest run and take the mean of the remaining 5 to be your final score.

Bibliography

- [1] Git tutorial. <https://git-scm.com/docs/gittutorial>.
- [2] GitLab basics. <https://docs.gitlab.com/ee/gitlab-basics/>.
- [3] scion ping code and documentation. <https://github.com/netsec-ethz/scion/tree/scionlab/go/pkg/ping>.
- [4] scion showpaths code and documentation. <https://github.com/netsec-ethz/scion/tree/scionlab/go/pkg/showpaths>.
- [5] scion traceroute code and documentation. <https://github.com/netsec-ethz/scion/tree/scionlab/go/pkg/traceroute>.
- [6] Go documentation scion. <https://pkg.go.dev/github.com/scionproto/scion#section-readme>, 2020.
- [7] A Tour of Go. <https://tour.golang.org/>, 2021.
- [8] bash manual. <https://man7.org/linux/man-pages/man1/bash.1.html>, 2021.
- [9] dlv usage. <https://github.com/go-delve/delve/blob/master/Documentation/usage/dlv.md>, 2021.
- [10] Go networking package documentation. <https://pkg.go.dev/net>, 2021.
- [11] SSH manual. <https://www.man7.org/linux/man-pages/man1/ssh.1.html>, 2021.
- [12] Troubleshooting. https://moodle-app2.let.ethz.ch/pluginfile.php/1449820/mod_resource/content/2/troubleshooting.html, 2021.
- [13] tshark manual. <http://manpages.org/tshark/1>, 2021.
- [14] Wireshark manual. <https://www.wireshark.org/>, 2021.
- [15] Chris Kuehl. tmux – a very simple beginner’s guide. <https://www.ocf.berkeley.edu/~ckuehl/tmux/>, 2013.
- [16] F. S. Foundation. GDB manual. <http://manpages.org/gdb>, 2016.

- [17] 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/>, 2018.
- [18] M. Kerrisk. systemd manual. <https://www.man7.org/linux/man-pages/man1/systemd.1.html>, 2021.
- [19] Network Security Group, ETH Zurich. SCIONLab. <https://www.scionlab.org/>, 2021.
- [20] Network Security Group ETHZ. SCIONLab repository. <https://github.com/netsec-ethz/scion>.