Module 04: DDoS Attacks and Defences

Week 2: Defending against DDoS attacks

Joel Wanner, Christelle Gloor, Adrian Perrig

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

November 2020

Problem Overview

In this exercise, you are tasked to provide DDoS protection as a service to a customer. This will require you to implement a firewall that can defend against increasingly sophisticated attacks by cleverly utilizing the network layer of the SCION architecture to block malicious traffic, while still providing good quality of service to legitimate clients of your customer.

Technical Setup

You will continue working on the same VM as last week. The relevant components for this task are shown in Figure 1.

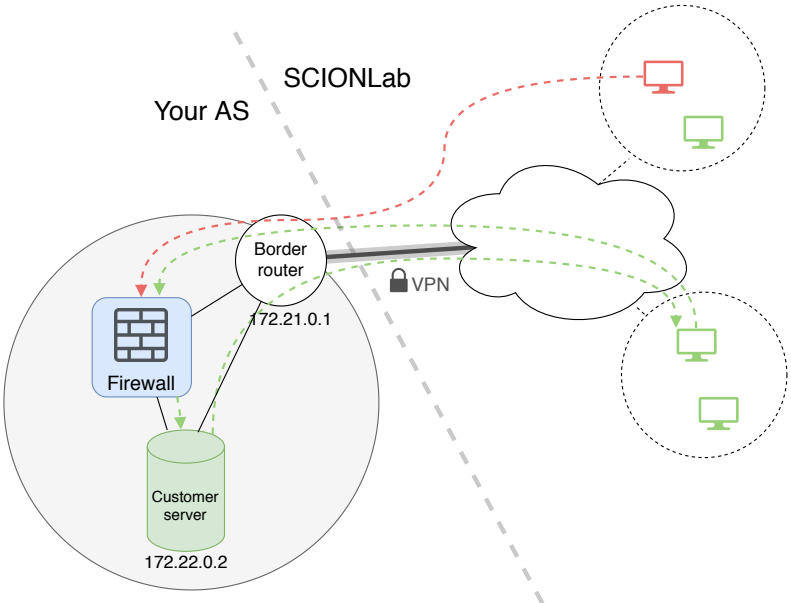

Figure 1: The relevant network components. The customer server is publicly reachable to clients in the SCION network, and your task is to filter incoming requests.

We have set up a code skeleton for the firewall that intercepts all incoming traffic to the customer, which is listening on the SCION address `17-ffaa:0:x,[172.22.0.2]` (where x stands for your AS number). While your firewall is not running, traffic is delivered to the customer unfiltered. Replies from the customer server are always sent back out of the AS directly.

Your task is to implement decision logic for each packet that determines whether

to forward it to the customer or drop it. Your code may keep state to make these decisions.

Attacks

The problem is split into three separate attacks, which grow increasingly sophisticated. These will be referred to as *Level 1*, *Level 2*, and *Level 3*, and each provides you with a number of points toward your final score (as defined in the grading scheme below).

A challenge for this week is that you are not allowed to submit separate solutions for each of these subtasks. Instead, the goal is to build **a single solution** that is able to defend against different classes of attacks.

In this sheet, we will not provide any description of the attacks you will be defending against. It is part of the task—just like in real-world DDoS mitigation—to inspect the attack traffic before designing effective measures against it.

Customer Service

The customer you are protecting exposes a public service that runs the following simple protocol:

$$\begin{aligned} A \leftarrow B: &\quad n_B \\ A \rightarrow B: &\quad n_A, \text{data}, \{\text{data}\}_{\text{sk}_A} \end{aligned}$$

where n_B, n_A are randomly generated nonces, and sk_A refers to the private key of A. In essence, upon receiving a request from a client B, the server A computes some data and sends it back to the client.

These computations on the server side are quite time-intensive (which is simulated in this lab by an artificial processing delay of around 200 ms). Therefore, it is important that as few malicious requests as possible are forwarded to the server to ensure that it has enough resources left to serve legitimate customers.

Since you are playing the role of an external DDoS defense service with potentially many customers, you do not have the ability to change the target protocol. Instead, your task is to put network-level defences in place, that prevent the server from being brought down.

Implementation

Inside your VM, you will find a code skeleton for the firewall at `~/src/defense`.

- `main.go` is the **deliverable** for this week. This is the only file you will need to work on.

- `lib` contains some code that intercepts packets and parses the relevant headers (UDP and SCION).
- `common` provides type definitions for the parsed headers. This contains all the necessary information for implementing this task—you do not need to import any of the SCION libraries this week.
- `connector.go` is a piece of code that takes care of loading the firewall library.

Simply run `make` to compile your firewall, and then execute it using the command `firewall`[1].

[1] The binary is linked to `/usr/local/bin` automatically, and you should also execute it from there.The reason for this is that setting capabilities is not supported on most mounted volumes, such as your `src` folder.

You are not allowed to use any libraries except standard Go libraries in the submitted code for this week. However, you are free to use any tools and libraries while working on your solution (e.g., for analyzing packet traces).

Testing

To launch test attacks, which are identical to those we will use for the final grading, you can use the pre-installed `isl` command-line tool:

```
isl test def1
```

This runs a Level 1 test attack for a short duration and provides immediate feedback on your performance. While the attack is running, the *Defense Task* dashboard on your local Grafana server (at `http://localhost:8001` on your host machine) provides additional live insights.

You are allowed to run as many test attacks as you need. Note that **each run is randomized**, so you cannot rely on getting the attacker's locations from past attack traces. Instead, your firewall must be able to make its decisions based on traffic that is observed live.

Grading

Grading Scheme

The points for this part of the lab are distributed as follows:

- **Level 1:** max. **25** points
- **Level 2:** max. **25** points
- **Level 3:** *optional*

For each level, your score will be determined by the quality of service provided to legitimate clients of the customer. This is computed by evaluating the following metrics over the time of the run:

$$\text{score} := \underbrace{\frac{n_{\text{success}}}{n_{\text{success}} + n_{\text{failed}}}}_{\text{success rate}} \cdot \underbrace{\min\{\sqrt{\frac{\ell}{\bar{r}}}, 1\}}_{\text{response speed}}$$

where n refers to the number of requests issued by clients over the period, and $\bar{r}$ is the average response time observed for successful requests. $\ell := 400\,\text{ms}$ is a baseline for the response time under normal (non-attack) conditions (based on RTT latency, server processing time + a margin to accommodate for overhead of the VPN, the customer server, and your firewall).

The final grades for the subtasks are then computed using the score for each task:

$$\text{score}_i \in [0, 1]$$
$$\text{grade}_i := \text{score}_i \cdot \text{maxScore}_i$$

for tasks $i \in \{1, 2, 3\}$.

Thus, the metric is affected directly by three main factors of your firewall implementation:

True positive rate : If attack traffic is not filtered out sufficiently, the customer will serve client requests more slowly or drop them.

False positive rate : When legitimate traffic is dropped by the firewall, the client will report a failure, decreasing your success rate.

Firewall performance : Implementations that do not perform well will incur additional overhead[2].

[2] Note that this does not mean that you are expected to write a highly optimized implementation. We have scaled down the bandwidth such that the firewall will never be the bottleneck unless it performs algorithmically complex tasks.

Submission and Final Grading

For this part of our module, we ask you to submit a single file as the deliverable. Combined with the attack part from last week, this results in two files that you need to upload.

When the task is done, we will run your submissions individually on a dedicated AS that is part of the SCIONLab infrastructure. We will run each task a number of times and take the median for your final score to rule out any remaining network effects.