Cybersecurity Project

Academic Year 2023-2024

Nebula Security Domains

Samuel Vojtáš

December 2023

1 Introduction

Underlying structure of the Internet nowadays, so called Internet Protocol[1], is inherently insecure. This report explores a possibility for implementation of secure overlay on top of Internet – the Nebula network[2] in addition with a concept of security domains. Security domains are sets of logically connected resources in the Nebula network that can freely communicate with each other. Resources that do not share security domains are not able to open connections to one another.

1.1 Nebula Network Overview

Nebula is a networking tool designed to connect computers on the Internet together in a secure, yet still efficient way. It creates a peer-to-peer[3] (P2P) software-defined network that uses concepts like encryption, authentication and tunneling. Peers can then communicate with each other securely using those concepts.

1.2 Goals

Goal for this project is to create an extra functionality to the Nebula network that will be called *security domains*. In Nebula network, resources on the network can by default communicate with each other without any obstacles. Hosts on the network are visible to other hosts. Our objective is to link some of the resource together using these security domains, so only resources that share at least one security domain can send TCP or UDP traffic to each other.

For this, Nebula provides two mechanisms that will be useful while implementing the security domains – *groups* and *firewall rules*. Upon network instantiation, each resource will be assigned Nebula groups for desired security domains. Then, every resource will have specific firewall rule (in their configuration file for Nebula network) that will allow network traffic only to other resources in these groups. Using groups and firewall rules, we can effectively implement the security domains as described previously.

1.3 Project Structure

Project is hosted in Github repository[4] and has following structure:

[1] `https://en.wikipedia.org/wiki/Internet_Protocol`
[2] `https://github.com/slackhq/nebula`
[3] `https://en.wikipedia.org/wiki/Peer-to-peer`
[4] `https://github.com/samuel-vojtas/Cybersecurity-Project`

```
PROJ_DIR                          # Directory of the repository
  |-- bin                         # For Nebula binaries
  |-- src                         # Scripts for network instantiation & testing
  |-- conf                        # Configuration files
  |-- docs                        # Report & presentation
  |-- docker-image                # Directory to create container image for testing
  |-- requirements.txt            # Python modules to install
```

Individual directories will be described below.

1.4 Prerequisites

To properly instantiate the Nebula network and create all necessary files for the hosts, we will need:

- **Nebula binaries**: Binaries `nebula` and `nebula-cert` downloadable from Nebula Github page[5]. Both should be placed in `PROJ_DIR/bin` directory.
- **Python3, `pip` and necessary modules**: Since scripts for instantiating Nebula network are written in Python, we need Python3 interpreter. Modules, that scripts use, are described in `requirements.txt` file. It is recommended to create a Python virtual environment before running the scripts.

  ```
  # Change directory to the project
  $ cd $PROJ_DIR

  # Create the virtual environment
  $ mkdir venv
  $ python3 -m venv ./venv
  $ source ./venv/bin/activate

  # Install necessary modules
  $ pip3 install -r requirements.txt
  ```

- **Docker**: For testing purposes, it is needed to have multiple containers acting as the hosts on the network. Docker will be used to run these containers. Testing containers will use specially crafted image, which can be built using following commands:

  ```
  # Change directory to the docker-image directory
  $ cd $PROJ_DIR/docker-image

  # Build the image
  $ docker build -t nebula-image .
  ```

2 Detailed Description of Nebula Network

Nebula network uses an architecture composed of a lighthouse and resources of the network. When resource wants to communicate with other resource, it needs to first contact the lighthouse that will open the connection between these two resources. Since resources can access the network from behind firewall or NAT, lighthouse needs to be reachable from all network clients. Therefore it is recommended that the lighthouse is public, i.e. it has public IP address and other resources can connect to it.

Communication between resources is encrypted, therefore Nebula provides a mechanism for authenticating the resources on the network. Every resource is in possession of cryptographic key and

[5] `https://github.com/slackhq/nebula/releases`

certificate. This certificate contains information about resource's name, IP address or the groups it is part of. It is signed by trusted certificate authority (CA). When verifying if the resource is really client of the network, certificate of the certificate authority is used.

Moreover, every resource has a configuration file with information such as how to reach the lighthouse or its firewall rules. This configuration file is specified in YAML[6] format.

Before connecting to the network, resource has to own these files:

- `host.key`: cryptographic key of the resource,
- `host.crt`: certificate of the resource signed by CA,
- `ca.crt`: certificate of the CA,
- `config.yaml`: configuration file for the host.

To simplify, we will assume that all of these files will be saved in `/etc/nebula` directory for any given resource.

2.1 Achieving Security Domains in Nebula Network

For now, we only explained how the Nebula network works. In this form, it does not provide any security domains. To create security domains, we will use Nebula groups. Every security domain will have a Nebula group that is specified upon creation of resource's cryptographic key and certificate. For example, to create a resource called `laptop1` with IP address `192.168.100.5/24` and security domain `laptops`, we can do:

```
# This command assumes we have the key for CA stored in ca.crt
$ $PROJ_DIR/bin/nebula-cert sign -name "laptop1" -ip "192.168.100.5/24" -groups "laptops"

# We can print the information about the certificate
$ $PROJ_DIR/bin/nebula-cert print -path "laptop.crt"
NebulaCertificate {
    Details {
        Name: laptop1
        Ips: [
            192.168.100.5/24
        ]
        Subnets: []
        Groups: [
            "laptops"
        ]
        Not before: 2023-12-16 20:26:01 +0100 CET
        Not After: 2024-12-15 18:59:21 +0100 CET
        Is CA: false
        Issuer: c4df36344109f63ccaef8ad96191616d415bc202dbc7d693a3c1724e786c500e
        Public key: df14afdb26f236a88998b29e8c44787c6315844896477e59593ce88398bb3306
        Curve: CURVE25519
    }
    Fingerprint: 0c957f24ce0fcd56089118c65bc579e458368f733ac4bbd28131eeb7af811068
    Signature: 6c2d6b7ce92b32cd17041c0e7cd54f3bb5ddaf86916cd2300fa09a2065c53d9e3a40...
}
```

The command to create the certificate is however encapsulated in the scripts for this project and will be executed as a part of the process of instantiating the network.

[6] `https://en.wikipedia.org/wiki/YAML`

Next, network needs to know that resources can only communicate with other resources from the same security domain. Therefore, for each security domain resource has (i.e. each Nebula group), there will be a rule in the `config.yaml` stating that it should allow connections from these security domains.

```
# Config.yaml: Configuration file of resource belonging to security domains:
#      group1 & group2
...
firewall:
    outboud:
        # Allow all outbound traffic from the resource
        - port: any
          proto: any
          host: any

    inbound:
        # By default, allow all ICMP traffic to the resource
        - port: any
          proto: icmp
          host: any
        # Allow resources from group1 to contact the resource
        - port: any
          proto: any
          group: group1
        # Allow resources from group2 to contact the resource
        - port: any
          proto: any
          group: group2
        ...
...
```

Resource having this configuration file with security domains `group1` and `group2` can only communicate with resources having at least of these security domains. Note that this does not apply for ICMP communication. For purposes of this project, *creating a connection between resources* means to communicate over TCP or UDP.

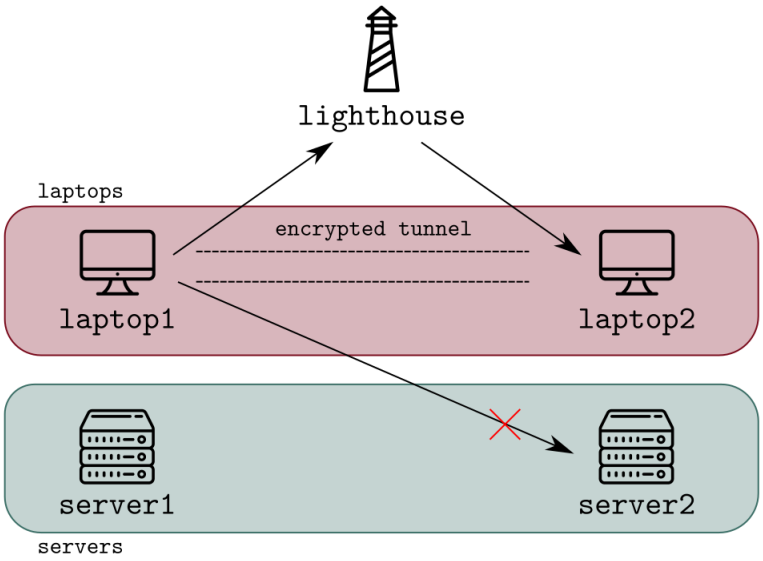
Now, we are ready to show an example of Nebula network with security domains. Suppose, our network contains 4 resources – `laptop1`, `laptop2`, `server1`, `server2`, and 1 lighthouse called `lighthouse`.

There will be two security domains – one for laptops called `laptops` and one for servers called `servers`. Purpose of this is to segment the network in smaller chunks, so that laptops can only communicate with each other, but not servers and vice versa.

Suppose that resource `laptop1` wants to create a connection to `laptop2`. At first, it needs to contact the `lighthouse` which will try to make the connection between the two resources. Resource `laptop2` sees network traffic from `laptop1` which belongs to security domain `laptops`. Based on the firewall rules inside configuration file for `laptop2`, it will decide to open the connection. New encrypted tunnel will be established between these two resources and they can communicate with each other.

On the other hand, if `laptop1` tries to connect to `server2`, the lighthouse will act the same in this situation and tries to make the connection between `laptop1` and `server2`. However, `server2` sees that it does not share a security domain with `laptop1` and decides not to open the connection. Resources `laptop1` and `server2` will not be able to communicate.

We can see this scenario described in the picture below.



This method of creating security domains is used in the script and it is described more into detail in the next section.

3 Implementation

This project automates the method for creating security domains described above. It uses one global configuration file, which contains information about a lighthouse and every resource. It generates needed configuration files, certificates and keys for every resource in `PROJECT_DIR/generated`. They are later distributed among the containers for testing purposes, or can be manually copied to resources of the network.

3.1 Global Configuration File

Global configuration file stored in `PROJECT_DIR/conf/network-config.yaml` contains information about a lighthouse and other resource of the network.

For each of them, configuration file needs to contain the name, IP address that will be used on the network and security domains/groups they are part of. Additionally, for lighthouse, we need also public routable IP address along the port on which the Nebula service runs (this is usually 4242).

Example of the global configuration file describing network as on the previous picture can look like this:

```
# Lighthouse configuration (only 1 lighthouse is allowed)
lighthouse:
    # Name of the lighthouse
    name: "lighthouse"
    # IP address of the lighthouse on Nebula network
    ip: "192.168.100.1/24"
    # Public IP address of the lighthouse
```

```
    routable_ip: "164.90.172.129"
    routable_port: "4242"

# Resource configuration
resources:
    # Name of the resource
  - name: "laptop1"
    # IP address of the resource on Nebula network
    ip: "192.168.100.2/24"
    # Groups/security domains resource is part of
    groups:
      - laptops

  - name: "laptop2"
    ip: "192.168.100.3/24"
    groups:
      - laptops

  - name: "server1"
    ip: "192.168.100.4/24"
    groups:
      - servers

  - name: "server2"
    ip: "192.168.100.5/24"
    groups:
      - servers
```

3.2 Automation Scripts

The main script for this project is `PROJECT_DIR/src/main.py`. It has multiple phases:

1. **Parse arguments**

   This phase is done using `argument_parser.py` module. Main objective of this phase is to gather arguments for the network instantiation such as:

   - `--config-file CONFIG_FILE`: Specify path for the configuration file. By default it is set to `PROJECT_DIR/conf/network-config.yaml`.
   - `--no-containers`: Do not perform the testing of the network and running containers for each resource. By default, testing will be performed.
   - `--no-lighthouse-container`: If testing is performed, it creates also container for the lighthouse. However, in some cases, we want to host the lighthouse on remote server – then, this argument can be specified, so that lighthouse container is not run.

2. **Instantiate the network**

   Network instantiation done by `network_starter.py` is meant to create needed configuration files, keys and certificates in `PROJECT_DIR/generated` directory. If this directory already exists and is not empty, script will try to delete the files stored here.

   Script `network_starter.py` parses global configuration file and creates a certificate authority for Nebula network. Then for each lighthouse and resource it will create a cryptographic key and certificate (signed by newly created CA). Configuration files as described above are also created with special firewall rules based on the resource's security domains. There is a template for all configuration files stored in `PROJECT_DIR/conf/default_config.yaml` – information from global configuration file is added to each indivial configuration file.

   To perform *only* the creating of the needed files for each lighthouse and resource, it is not needed to execute `main.py` script. It can be also done by executing only `network_starter.py`.

3. **Start containers and distribute files among them**

   By default, after parsing the arguments and generating network files, testing phase is followed. Module `container_start.py` runs a container for each resource (optionally also for the lighthouse). Note that container has the same name as the resource specified in the global configuration file. It then copies all necessary files to `/etc/nebula` for given container and starts the Nebula service with:

   ```
   # Python script starts the service
   $ nebula -config /etc/nebula/config.yaml
   ```

   Image for the containers is created using a Dockerfile and it has Nebula installed. After this, containers as well as Nebula network are fully instantiated and resources sharing at least one security domain can communicate together.

4 Testing and Validation

Testing of the scripts for automating the Nebula network instantiation has been performed on Ubuntu 22.04.03 LTS. Testing machine had Docker and Python3 installed. Docker image `nebula-image` for the containers has been built before the testing. Following sections will describe how to test the network.

4.1 Testing with Container-only Setup

Container-only setup means that all resources including the lighthouse will have a dedicated container. The whole testing then will be performed locally on the same network that the containers share.

Warning Routable IP address of the lightouse needs to be specified in the global configuration file. When working with containers and default bridge network, firstly created container (i.e. the lighthouse) will have IP address `172.17.0.2` since `172.17.0.1` is the default gateway. **For this testing scenario, all containers need to be stopped before, so the lighthouse has this IP address**. IP address `172.17.0.2` needs to be specified in global configuration file as `routable_ip` of the lighthouse.

```
# OPTION 1:
# Instantiate the network and test it on container-only setup
$ python3 src/main.py

# OPTION 2:
# Instantiate the network and test it on container-only setup
# Specify also configuration file
$ python3 src/main.py --config-file ~/my_dir/network-config.yaml
```

4.2 Testing with Containers and Remote Lighthouse Setup

In this case, the lighthouse is expected to run on a remote host. For this, testing included virtual private server that acts as a lighthouse. We need to manually copy the file to the lighthouse.

```
$ python3 src/main.py --no-lighthouse-container
...
  [*] Nebula files were distributed among the containers
  [*] Waiting to start Nebula service on remote lighthouse
```

Network files are distributed to the containers, and script is paused, so the user can manually instantiate the lighthouse.

```
$ cd ${PROJECT_DIR}/generated
$ IP=<INSERT IP OF THE LIGHTHOUSE>

$ scp ca.crt root@${IP}:/etc/nebula/ca.crt
$ scp lighthouse.key root@${IP}:/etc/nebula/host.key
$ scp lighthouse.crt root@${IP}:/etc/nebula/host.crt
$ scp lighthouse-config.yaml root@${IP}:/etc/nebula/config.yaml

$ ssh root@${IP}
# WARNING: nebula binary has to be also present at remote lighthouse
$ ./nebula -config /etc/nebula/config.yaml
```

After instantiating the lighthouse, go back to the paused script and press Enter, script will continue and containers will be able to contact the remote lighthouse.

4.3 Testing Network Files Generation

Another case of testing is just to generate network files for lighthouse and resources. Later the generated files can be distributed to the containers.

To generate the network files, issue this command:

```
$ python3 src/network_starter.py
```

Then, after the files are successfully created in `PROJECT_DIR/generated`, we can distribute them to the containers and start the Nebula service manually or with another script:

```
# Script to start the containers, distribute the files and start the Nebula service
$ python3 src/container_starter.py
```

4.4 Verifying the Connections with Netcat

To check if the containers are reachable, we can use Ping utility[7]. Ping uses ICMP datagrams, so all resources should be able to reach each other (ICMP is allowed in between all resources).

For verifying the connection between resources sharing at least one security domain, we used Netcat[8]. On one container, create a Netcat server and connect to it on the other.

```
$ docker attach server1
# server1 at 192.168.100.4
# Start the server
$ nc -lvnp 2222

$ docker attach server2
# server2 at 192.168.100.5
# Connect to the server
$ nc 192.168.100.4 2222
```

5 Conclusion

This project designed an addition to the Nebula network called *security domains*. Automation of the network instantiation has been implemented via Python scripts.

[7] https://en.wikipedia.org/wiki/Ping_(networking_utility)
[8] https://it.wikipedia.org/wiki/Netcat

There is also an option to create few testing scenarios that include verifying the connections between resources that share at least one security domain.

All generated files (i.e. cryptographic keys, certificates and configuration files for network hosts) are stored in the `PROJECT_DIR/generated` directory. Certificates contain the information about the security domains.

Nebula is useful tool when we want to connect network clients that are accessing the network from behind NAT or firewall. Security domains further widens the usage of Nebula because now, the network can be segmented into more chunks, efficiently making the network more secure.

6 References

- `https://github.com/slackhq/nebula`
- `https://nebula.defined.net/docs/`
- `https://docs.docker.com/`