Clusus: a cyber range for network attack simulations

by

R. Flinterman
A.L. Smit
E. Hildebrand
J. Mulder

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Friday July 5, 2019 at 12:00 PM.

Project duration:	April 23, 2019 – July 5, 2019	
Supervisors:	Dr. C. Doerr,	TU Delft
	Dr. S. Roos	TU Delft
Project Coordinators:	Ir. O.W. Visser,	TU Delft
	H. Wang,	TU Delft

TU Delft

Preface

This report concludes the Bachelor Project, which is a requirement for attaining the status of Bachelor Computer Science and Engineering at the Delft University of Technology. The client for this project was the Cybersecurity Department at the Technical University of Delft. The goal of this report is to inform the reader of the stages of research, development, and implementation that this project went through, as well as to inform them of the capabilities of the final product.

In this project we were guided by our Client and our Coach, who have assisted us in many ways. Therefore we would like to thank Christian Doerr for being an amazing client and Stefanie Roos for her valuable advice.

R. Flinterman
A.L. Smit
E. Hildebrand
J. Mulder
Delft, July 2019

Summary

This report documents the design and implementation of Clusus, a cyber range to provide students with a safe isolated environment to learn about cyber security and computer networks. This Bachelor project was proposed by the TU Delft cyber security group.

During a two week research phase currently existing solutions were evaluated and requirements for the system were determined. Based on these requirements and research into the capabilities of cloud providers a design was proposed.

In the second phase of the project was the implementation of the design. The chosen design consists of three major components, a central server that handles the communication between the learning management system and an individual exercise, a containerized exercise, and finally a program that builds these containers. The containerized exercise consists of all virtual machines required for an exercise and a monitoring program that reports progress to the central server.

Contents

1 Introduction

TU Delft offers courses on network security. To make these courses more interactive, a system was to be developed for the cyber security group to allow students to experiment with computer networks. Because a variety of virtualisation technology was available, the first 2 weeks of the project were devoted to doing research on the available options. The remaining 8 weeks were spent on implementing a solution.

A system like this is required, because most students don't have multiple computers they could use to set up a practice environment at home. The capacity to provide resource intensive exercises on campus is also limited. Another issue is that the exercises require students to attack a network, and if a student were to make a mistake they could end up attacking computers unrelated to the exercise. Therefore the goal of this project is to build a system that can automatically set up a safe practice environment on a cloud service and report back to Moodle or another Learning Management System when an exercise has been completed.

1.1. Outline of the Report

The report is structured as follows, in chapter 2 the findings from the research phase of the project are explained. In chapter 3 an overview is given of the system, and how it was designed. The implementation is described after that in chapter 4. How the system was tested is described in chapter 5. In chapter 6 the final product is evaluated and the ethical implications of the product are discussed. Then in chapter 8 the process is evaluated and some recommendations for future work are made. Finally chapter 9 has the conclusion of this report.

The report also has a few appendices. Appendix A has the original project description. The project info sheet is located in appendix B. And the evaluation from SIG can be found in appendix C.

2 Research

In this chapter, the gathered information from the first two weeks of the Bachelor Project will be described. Technologies and libraries that were found to be useful will be detailed, as well as a proposed solution to the problem.

The aim of this project is to allow students to get hands-on practice with computer networks and network attacks in a safe environment. Currently this is an issue, as most students don't have multiple computers at home to practice with and resource intensive exercises can only be provided for a short time on campus. Therefore the goal of this project is to build a system that can automatically set up a safe practice environment on a cloud service, and report back to Moodle when a student has completed the exercise.

In order to make sure the right approach to this problem is taken, the first 2 weeks of this project were dedicated to research, in order for a plan to be set up. After having done this research, a clear problem definition and analysis could be created, This is described in section 2.1. In section 2.2, the design goals identified to be integral to our project are defined. Using these design goals and a list of requirements from the client, clear and objective requirements were formulated, which are laid out in section 2.3. section 2.4 provides details about our work methodology. section 2.5 then lays out the proposed solution design towards solving this problem. Lastly, section 2.6 goes over some of the alternative solutions that were identified.

2.1. Problem Definition and Analysis

Problem Definition

The TU Delft offers courses where students can learn about network security. In these courses, students should be able to perform practical exercises in which real world scenarios pertaining to network security are simulated. However, it should be possible to perform these exercises safely and securely. Current available solutions allow students to direct traffic to addresses outside the network. Another problem lies in the versatility of these solutions. It should be possible to specify an exercise statically, and make the system generate it dynamically.

Problem Analysis

The TU Delft Cyber Security department wishes to have a versatile system that is robust against tampering by students. This system should be capable of generating exercises dynamically and deploying them on the cloud or locally. With such a system, the TU Delft could offer interactive cyber security courses. Simulating such a network can be computationally intensive. Fortunately hosting the network on a cloud service provides a solution to this issue. However, as cloud services can change or be unavailable, the system should be portable and rely as little as possible on the availability of a singular service.

2.2. Design Goals

The main design goals for this project will be explained in this section. These will be taken into account with every step throughout the design and implementation of the project. With these goals in mind, a set of requirements was created. The following design goals have been decided upon: maintainability, security, and scalability. Security is the goal with the highest priority, as network attacks will be simulated within the environment of the system. Secondly maintainability is an important design goal, as it needs to be as modular as possible. Lastly scalability has to be taken into account, as many students will need to access and use the system simultaneously.

2.2.1. Security

Security is an essential part of our system, both internally and externally. Network attacks will be executed within the system as part of exercises, as the students implement protective solutions. Those exercise attacks must not leak out of the system into the cloud server. The students themselves will also have the necessary tools to attack the system from within, so these have to be prevented as well. To make matters more clear, some sub-goals are discussed below.

Testability

It's very important that mistakes in the code are prevented and that the system works exactly as intended. Therefore it must be designed with testability in mind. Tests will be written in order to evaluate the quality and performance of the code. The later-described design goal of modular independence helps with this.

Isolation

To further increase the inherent security of our system, the isolation of key components of our system is necessary. An exercise instance of student x_1 should not interfere in any way with an instance of student x_2, unless interaction between students' instances has been clearly specified as part of the exercise. Components such as a VPN (virtual private network) server should be properly isolated in this manner as well. By prioritising this form of isolation across our system components, it will achieve a higher level of security.

2.2.2. Maintainability

Maintainability is very important for this system as well. The way in which new exercises can be created and set up should be easy to understand and as modular as possible. Maintaining and modifying the code base should be made as easy as possible too. Especially since the system has several use cases for which some parts of the code may need to be altered (e.g. Brightspace integration, modifying the used cloud provider).

The system needs to be maintainable in other aspects as well. Behavior from students needs to be graded and tested, and new exercises should be able to be added, removed, and edited. The exercises will differ greatly in how each attack works and how it is defended against, so modular independence is a must have.

Modular Independence

Modularity is a key priority throughout our system. The configuration of the networks and exercises should be modularly independent in order to make this process as straightforward as possible. Swapping out components of the system (e.g. moodle integration) should be an easy process for other developers unfamiliar with the rest of the code. This will facilitate the adoption and growth of our system, and provide us with an easily understandable and extendable system.

Moodle and Brightspace both allow for the use of the LTI (Learning Tools Interoperability) standard.

Open-source Libraries for LTI are available for Java[1], Python[2], Node.js[3], Ruby[4], PHP[5], .NET[6]. Of these, the Java and PHP libraries are maintained by the creators of the LTI standard.

2.2.3. Scalability

This system will be used in Delft University courses, so many people will need to be able to access and use it at the same time. Teachers and teaching assistants also need to have their own view into the system. The challenge here is that the system is not allowed to crash or cause errors under the pressure of all of those users being active at the same time. Moreover, a singular exercise (e.g. a very large simulation of a Distributed-Denial-of-Service attack) should not be able to bring down the entire system with it.

[1]https://github.com/IMSGlobal/basiclti-util-java
[2]https://github.com/pylti/lti
[3]https://github.com/omsmith/ims-lti
[4]https://github.com/instructure/ims-lti
[5]https://github.com/IMSGlobal/lti-1-3-php-library
[6]https://github.com/andyfmiller/LtiLibrary

2.3. Requirements Analysis

In this section, the requirements for the project are laid out. These requirements have been used to formulate the project plan and form the base of this project's development process.

2.3.1. Requirements

The following requirements have been set based on the problem definition and the design goals. The requirements have been prioritised using the MoSCoW method.

Must Have

- The product must be able to automatically claim resources from a cloud provider.
- The product must be capable of instantiating cyber ranges on at least two cloud providers.
- The product must have a universal standard for describing virtual machines, and only differ in how this information is communicated to the cloud provider.
- The product must be able to switch between the cloud providers used to host the exercise. This must not take more than 10 minutes of manual work.
- The product must automatically shut down once the exercise has been completed (i.e. unclaim the used resources on the cloud provider).
- The product must automatically shut down after a period of inactivity.
- The client virtualisation must be based on Docker/Virtualbox.
- The product must be able to read a configuration from a file. The file lists IP addresses, connections between VMs (Virtual Machines), and which scripts should be run on each host.
- The network must have configurable access points where network traffic can be tapped. These taps should not be able to be moved or changed by the end user.
- The product must be able to configure at least one login via VPN.
- A user must be able to get the VPN login from Moodle.
- The system must be able to control which kinds of network traffic are allowed into and out of the virtual network.
- Any exercise must be able to run locally on the student's computer.
- An exercise must be able to support 30 virtual machines running simultaneously.
- A TA (Teaching Assistant) can view the status of the exercises as they are running (CPU load, etc.)
- A student or TA must be able to reset/shutdown the exercise.
- The network supports the injection of IP (Internet Protocol) and MAC (Media Access Control) address spoofed traffic.
- The product must be able to be showcased for a small example exercise.

Should Have

- The system should have the ability to install a specific version of software packages.
- The system should be able to start from a frozen VM.
- Network configurations must be able to be combined and reused for later (larger) scenarios.
- A TA should be able to view the network as the student is working on it.
- A machine should be able to shut down automatically when a student exceeds a certain amount of activity.

Could Have

- "Multiplayer": The product could provide interaction between users, where one person sends a network attack to another user, who protects their network against it.

Won't Have

- The product won't be integrated with Brightspace

2.3.2. Success Criteria

Before starting the project, it is important to make sure the objective is crystal clear. To this end success criteria have been defined to gauge the "doneness" of the delivered solution. The requirements from subsection 2.3.1 give clear functional criteria to objectively measure progress. Especially, when the requirements pertaining to the design goals 'Security' and 'Maintainability' are met, a capable system for setting up cyber ranges has been reached. Scalability is not as important, as its inherently less of an issue for this particular use case. This is because it's trivial to rent more computing power at a cloud service. Thus, if at least the first two design goals are met, the system could be called a success.

2.4. Methodology

Throughout the project scrum will be used to optimise the productivity in the software development process. Scrum works well, as specific requirements have already been clearly defined, which makes sprint composition straightforward. There will two meetings per week with the client to make sure the product is developing in accordance with their wishes. These meetings integrate well with the Scrum methodology, due to its agile nature, and ability to adopt feedback from clients rapidly.

During the course of the project, the Insyght Lab in Building 28 of the TU Delft will be used. In this room, a workstation has been set up, which can be used for testing purposes. The development team will mostly be present in this room every day of the week, from 9 a.m. till 5 p.m. This further conforms to the guidelines for utilising Scrum, as daily meetings and evaluations are core to its methodology.

2.5. Proposed Solution

This section will give an overview of the proposed solution. A visual representation is given in Figure 2.1.

Overall Structure

Since the portability of the system is an important requirement, it was chosen for the machines defined in an exercise configuration file to exist as VMs that are nested inside a host VM, this host VM is provided by the cloud provider. There are several advantages to this as opposed to having each machine in an exercise be a separate machine at a cloud provider.

- Cloud providers often block certain kinds of traffic for security reasons, but these kinds of traffic are necessary for some of the exercises.

- Cloud providers do not allow one to spin up a frozen VM state, only the state of the disk is frozen. Spinning up from a frozen VM state is a desired feature as some exercises may rely on an exercise that is set up manually.

These circumstances are not true for all cloud providers examined, but they are generally true.

However, a disadvantage may be decreased performance due to nesting VMs. Without nesting VMs the system would need a lot more resources locally at TU Delft because a lot of exercise types could not be run in the cloud. Nesting VMs also allows the system to simulate Windows machines in exercises.

The nested VMs will use VirtualBox box for virtualisation as per the requirements. VirtualBox can be fully configured and run using the VBoxManage API (Application Programming Interface), allowing it to be integrated into the system.

Learning Management System (LMS) Integration

Though the product is meant for use with Brightspace, the client declared actual integration with Brightspace to be out-of-scope of this project, and a demonstration using a substitute would suffice. This is because Brightspace is proprietary software and there is no testing environment available. Instead, Moodle shall be used, as its interface is similar to Brightspace, both using the LTI standard for Learning Management Systems.

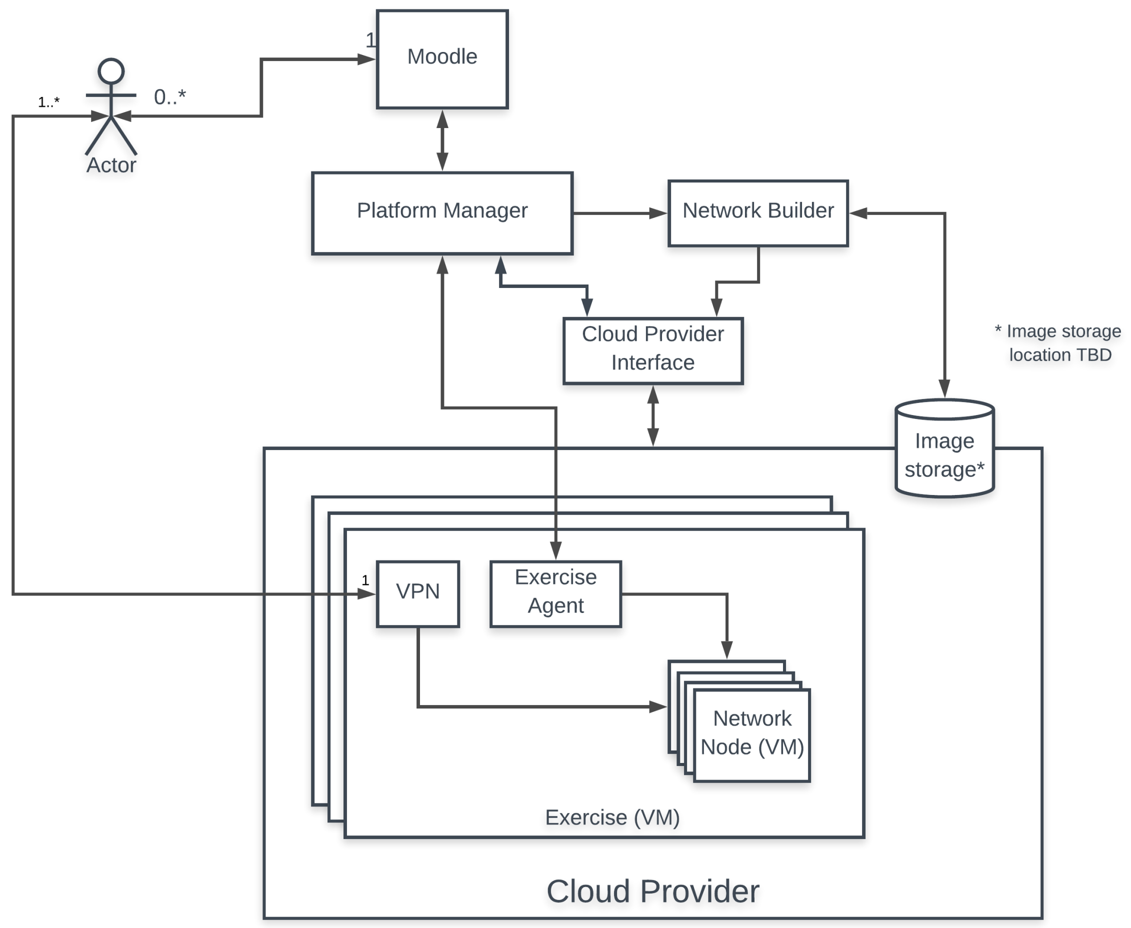

Figure 2.1: Proposed Solution Design Overview

In addition, Moodle is open-source. This means that the developers can have full control over the learning environment.

Platform Manager
The Platform Manager will be the backbone of the system. It communicates with the LMS (i.e. Moodle) to provide the students with the necessary info for their exercise. The Platform Manager has the following responsibilities:

- Listen for requests done by students, spin up their exercise, and monitor it.
- Monitoring the overall state of the exercise (e.g. number of students currently in an exercise, used resources on cloud, etc.).
- Allow teachers to specify and build exercises in advance, saving them to the image storage.
- Notify Moodle when a student has finished an exercise.

Network Builder
The Network Builder is responsible for creating the images stored in the image storage. It will generate a VirtualBox VM for each machine specified in the exercise configuration. These VMs will be connected to each other according to the provided configuration. VMs that should be accessible over the VPN tunnel will have a second network card on a separate subnet to connect them to the VM hosting the openVPN server. The Network Builder will then put all these virtual machines on a single virtual machine together with the exercise agent so it can be stored in the image storage ready to deploy on a cloud provider or locally. The

image builder will also setup the firewall so no data can leave the machine unless specifically allowed by the exercise configuration.

Cloud Provider Interface
As the system will be using two cloud providers to ensure exercises can be spun up at all times, it will need an interface capable of spinning up exercises on both providers, given the same exercise specification. Again, this part of the system needs to be as modular as possible, so that adding or switching out cloud providers is as easy and straightforward as possible. Switching between cloud providers in the interface should be seamless. If for example a cloud provider stops working during a practical, switching to the alternate cloud provider should be quick and not take too much time away from the practical.

Furthermore, the interface should also be able to spin up exercises locally on resources from TU Delft. This can be done when the Platform Manager indicates that utilising the cloud providers is not yet necessary, due to little expected load.

Image Storage
From a user perspective, starting an exercise should not take too long. To expedite this process, exercise images are stored. When a particular exercise is requested by a user, the Network Builder recognises that it does not have to re-generate it and can just serve the pre-loaded image. However, having to upload/download VM images when they are not locally available nullifies the advantages of having stored exercise images. Thus, the system will need to have such an image store on both the cloud providers and the local server.

Exercise Agent
This program runs on the host VM and has two tasks, it monitors the output of the milestone scripts defined in the exercise configuration and it monitors resource usage of the VMs and shuts down the exercise after a period of inactivity.

The milestone scripts put their output in a shared folder so it can be accessed on the host VM. The agent reads these files and updates the platform manager, which notifies Moodle when milestones are reached.

The program also monitors the resource usage so the exercise can be shutdown after a period of inactivity to avoid incurring high costs at the cloud providers by keeping unnecessary machines active.

Cloud Providers
A requirement for the system is that it needs be able to concurrently handle several students. To make sure the system is capable of doing this, it will be utilising the cloud providers Microsoft Azure and Google Cloud. These two were chosen because they support nested virtualisation, which is required for the system's architecture. A more in depth comparison between the reviewed cloud providers is given by Table 2.1. This architecture was chosen because an initial survey of cloud providers showed that their network architecture blocks some of the traffic the system needs to support for security reasons. By virtualising the network inside the system this traffic never reaches the cloud providers network. However this requires the cloud provider to support nested virtualisation so VirtualBox can be run on the machine provided by the cloud provider.

Implementation Details
Communication to the chosen cloud providers and Moodle is supported by Java libraries. Moreover, VirtualBox provides an API written in Java as well. This is why the solution will also be implemented in Java. Some parts of the program will be using bash scripts to setup the virtual machines and install the right software on them, as the API does not.

2.6. Alternative Solutions
While creating the proposed solution, alternative solutions that are already available were examined. By doing this additional insights were gained as to what is necessary when designing a cyber range. Furthermore, by looking into these alternative solutions, common pitfalls and challenges in designing such a cyber range were identified.

KYPO
KYPO, a cyber range that has been in development since 2013, *"provides a virtualised environment for performing complex cyber attacks against simulated cyber environments"* [46]. In their paper, their architecture

Table 2.1: Cloud Provider Comparison

Cloud Provider	**Network Capturing**	**ARP Poisoning**	**MAC Spoofing**	**IP Spoofing**	**Free Trial**	**Nested Virtualization**	**Docker Support**
AWS	Very limited[44]	No[4]	No[4]	No[4]	Yes[2]	only on metal instances [3]	Yes[1]
Microsoft Azure	Yes[17]	No[42]	Possibly[5]	Not for Azure Cloud IP addresses[42]	Yes[16]	Yes [19]	Yes[15]
Google Cloud	Yes[13]	Unknown	Unknown	Possibly[10]	Yes[12]	Yes[14]	Yes [11]
Vmware	Yes[33]	Yes[32]	Possibly, no conclusive source	Possibly, prevention not defined	30 min trial	Yes[31]	Yes[30]
Oracle Ravello	Yes[24]	Yes[25]	Yes[25]	Unknown	Yes[20]	yes[21]	Unknown
Rackspace	Unknown	No[23]	No[23]	No[23]	No		
Red Hat	Unknown	Yes[27]	Unknown	Yes[26]	Yes[28]	Unknown	Yes [22]

design is described. Their prioritised requirements have quite some overlap with this project, as they also aim for flexibility, modularity, isolation and built-in monitoring. A high-level overview is given in Figure 2.2. As can be seen from this figure, the "sandbox" environment resides on Computing Infrastructure (e.g. a cloud provider).

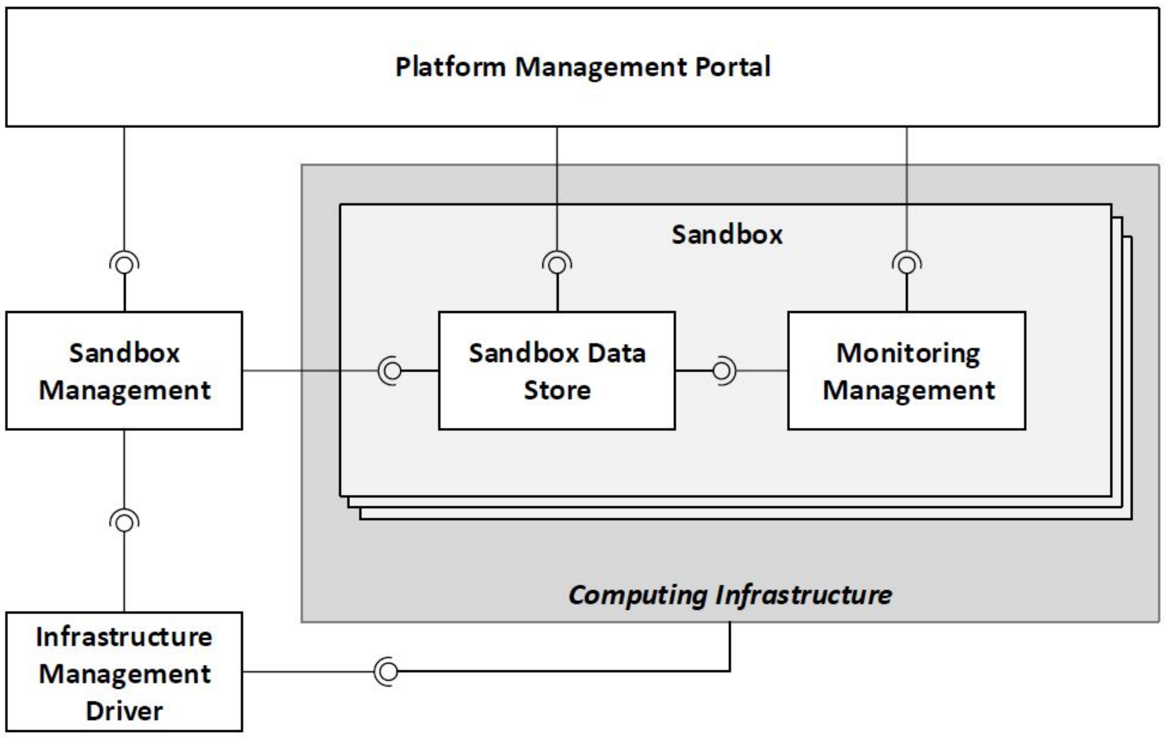

Figure 2.2: KYPO platform high-level architecture overview, image copied from [46]

According to Vykopal et al. KYPO has multiple use cases, namely: cyber research and development, digital forensic analysis and educational/training purposes. The first and last of these are similar to the use cases for this project. Vykopal et al. claim to have found the education and training use case to be the most challenging. One reason for this was the need for many additional features for the platform, mainly pertaining to user interaction. A second reason is the amount of customised content that was needed to provide students with a valuable learning opportunity. One solution would be to make the exercises as modular as possible, making it easier to design them. Exercises would need to be dynamic (i.e. the solution is not the same every time), so that the exercises are reusable for later students.

KYPO aims to achieve much of the same as this project, and thus reusing their solution would be an option, but KYPO is not open-source and is provided as a service. Therefore the KYPO solution cannot be reused for this project.

DETERLab

DETERLab, a facility created for cyber-security experimental science, originates from early work done by Dr. Douglas Maughan in 2003-2004. The objective at the time was to create a test bed for large-scale DDoS attacks.

The DETERLab design is rooted in Emulab [9], a testbed for networks. DETERLab can be accessed through a web portal. In this portal experiments can be started, after which the user has access to a set of machines provided by DETERLab via Secure Shell (SSH). The OS, applications, and network topology can be specified beforehand. Afterwards, users have privileged access and can modify the OS, install or modify applications, and modify system configurations [41].

Benzel's work [36] lists several lessons that were learned from the early work that was done with DETERLab. The first lesson relates to the construction of the experiments done in DETERLab. This construction was found to be difficult and limiting the learning rate of researchers using the lab. The experiment definition methodology lacked abstraction and re-use. Thus, a great deal of structure and details needed to be specified before an experiment could be simulated.

> *Acceleration of the pace of cyber-security research was blocked by the necessity of each experimenter needing to specify a great deal of structure, much of which was not critical to their needs, and without recourse to others' work.* [36]

Another lesson which relates to the requirements for this project pertains to experiment isolation. Early on in the development stage, this isolation was found to be limiting. This was due to the non-deterministic nature of some types of malware. The simulated malware in the isolated test bed proved to have very low fidelity to the one found "in the wild". To combat this, development was started on Risky Experiment Management. In this mode, the inherent constraints of an experiment can be used to make the test bed less constraining. More details on this can be found in [47].

Lastly, as stated before, DETERLab is built upon Emulab. The use of a library or software package to handle a portion of the application logic can be efficient and quick. However, it does impose some restrictions on the design possibilities. This was also the root cause for the difficulty of experiment construction [36].

DETERLab provides students and teachers with a platform meeting many of this project's requirements. But DETERLab too, is provided as a service. This platform, however, has been in development since 2003. This has resulted in many publications on DETERLab, containing valuable lessons for this project as well.

CyRIS

CyRIS [43], or Cyber Range Instantiation System, tries to provide a mechanism to automatically prepare and manage cyber ranges for cybersecurity education and training. It is implemented in Python, and facilitates both installing content on a large environment with many nodes, while also configuring the network service among them. Moreover, their project is open-source. An overview of the workflow for CyRIS is given in Figure 2.3. An overview of the architecture is given in Figure 2.4.

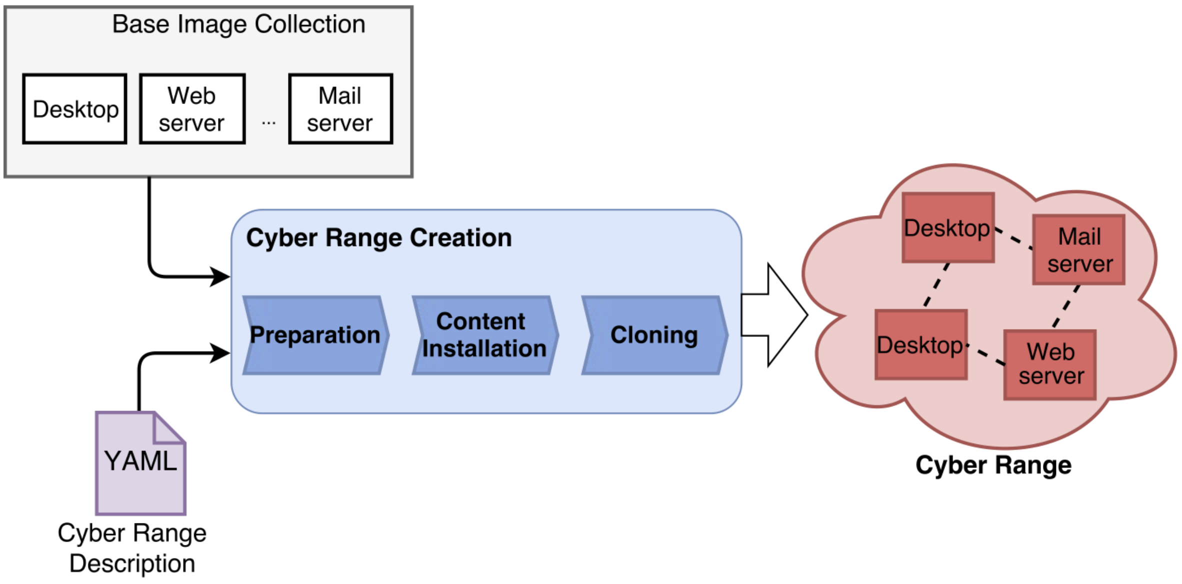

Figure 2.3: CyRIS platform high-level workflow overview, image copied from [6]

CyRIS conforms to many of this project's requirements. It provides the user with the following:

- the specification of the configuration of the network and underlying systems (including the installation of tools and packages)
- the emulation of incidents and malware
- the capture of network traffic

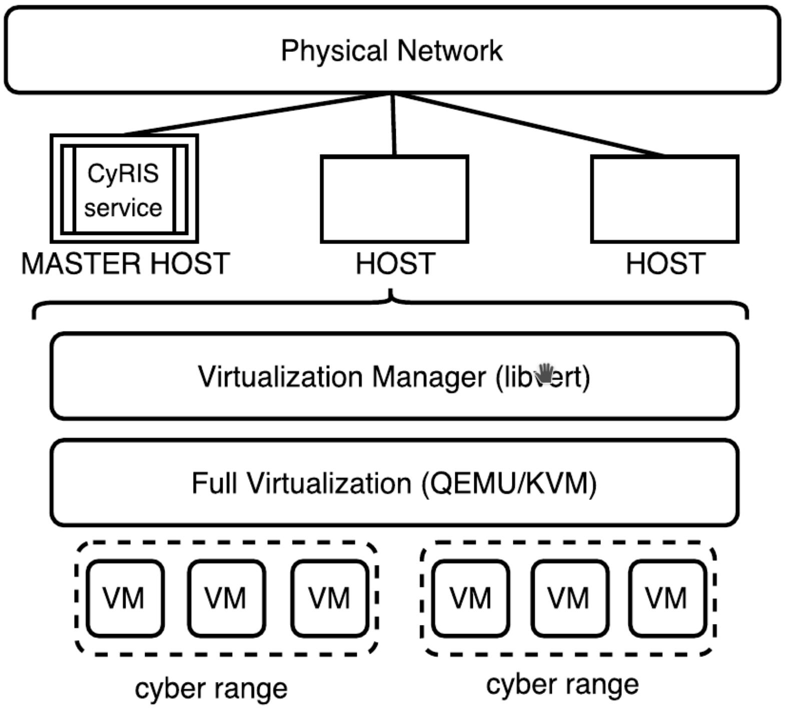

Figure 2.4: CyRIS platform high-level architecture overview, image copied from [43]

Capturing traffic is done using the software tool tcpdump [29], which could also be utilised in this project for similar ends. One drawback to the CyRIS implementation pertains to the possible topologies of the generated networks. According to their paper [43], CyRIS is only capable of producing networks with a ring structure (i.e. all the network nodes are accessible from one central node). Because of this inflexibility, the CyRIS system configuration code should not be reused for this project, as it would decrease the modularity of the system, and possibly restrict some other use cases as well.

3 Design

3.1. System Overview

An important part of the system is that it is very modular. The different parts should be able to run on the same or different computers without too much change in the workings. Furthermore, the system should be maintainable, as future developers should be able to add new functionality. Designing the system came with a variety of challenges.

First, a way for students to connect to the exercises should be provided. Since the TU Delft already has a central application that connects students to their course material, this application should also be compatible with that system. This will be expanded upon in section 3.2. Secondly, this application should have a central program that is responsible for managing students' connections to the exercises, and is capable of spinning up new exercises once a student requests access. This central server is the subject of section 3.3. Another important part was picking the correct tools for simulating the network environment. Choices for the simulation of the network will be discussed in section 3.4.

3.2. Learning Management System

The TU Delft uses Brightspace as its Learning Management System. It is desirable that the final product is also able to communicate with Brightspace. However, as Brightspace provides no free testing environment it was decided with the client that the open-source software Moodle was to be used for this project instead. Because Brightspace and Moodle both implement the LTI standard, it should be possible to embed this tool into Brightspace with minimal changes. Furthermore, as other popular Learning Management Systems also implement the LTI standard, it should be possible to migrate easily in the future as well.

3.3. Server

The server is the central point of control when the network is running. It receives a request from the Learning Management System when a student wants to start an exercise. It will spin up the exercise for the student in the cloud and give them keys to log into the allowed VMs. Then, when a student finishes an exercise, the server will receive feedback and pass this back to the Learning Management System. When the server updates the Learning Management System, the student will be able to observe that their grade has been updated. This process can be observed in fig. 3.3.

3.4. Simulating the Network

There are different possible ways to simulate a computer, but the most important ones are Virtual Machines and Containers. The main difference between them is that Virtual Machines also simulate hardware (and therefore are more heavy-weight), while containers are lightweight.

In the end, it was chosen that a docker container should contain all the virtual machines. There were a few reasons for this:

- By containerizing the application, it could be made more portable.

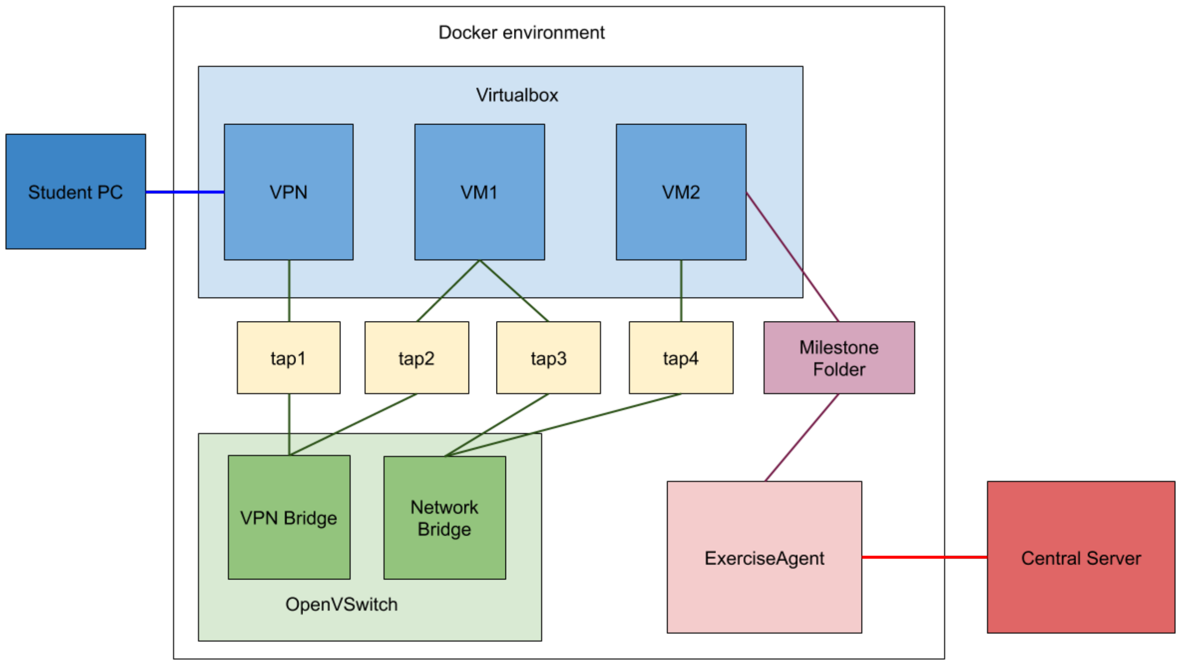

Figure 3.1: Layout of the exercise environment, with connections to a student's PC and the Central Server

- Containerizing the application means the application can be run without altering the computer that runs it.
- With a containerized exercise there is more control over which networks are allowed. When running machines directly in the cloud communications between these machines are heavily restricted to prevent abuse.
- When running the exercises on a local cloud server it is possible to run multiple instances of the docker image on the same computer.

The networks between the machines are constructed using the open-source digitl switch solution OpenVSwitch[34]. OpenVSwitch allows the user to create network bridges, as well as configure which machines are able to access the network with rules. The reason OpenVSwitch is used instead of VirtualBox' inbuilt networking is because Virtualbox does not allow blocking network traffic while giving machines access to the internet [45]. Virtualbox also does not support capturing network traffic. However, for the purposes of this system it is necessary that the networks can be captured, as certain network behavior signifies a completed exercise. While this can be done by running a script on a VM by using OpenVSwitch this detection can run on the host, isolated from students. Additionally, these internal networks only allow for virtualbox VMs to communicate with other virtualbox VMs, while it may be desirable for these VMs to be able to communicate with non-virtualbox applications.

3.5. Distributing the Exercise

As the exercise is contained in 1 file, which is to be built on a computer of the teacher's choice. The teacher can then run the docker image and inspect that it is behaving correctly. If they are satisfied, they can upload this exercise to the docker repository in the cloud service used for the exercises. Most cloud providers that were considered for this project have their own docker repository, as seen in table 2.1. This process is described in fig. 3.2.

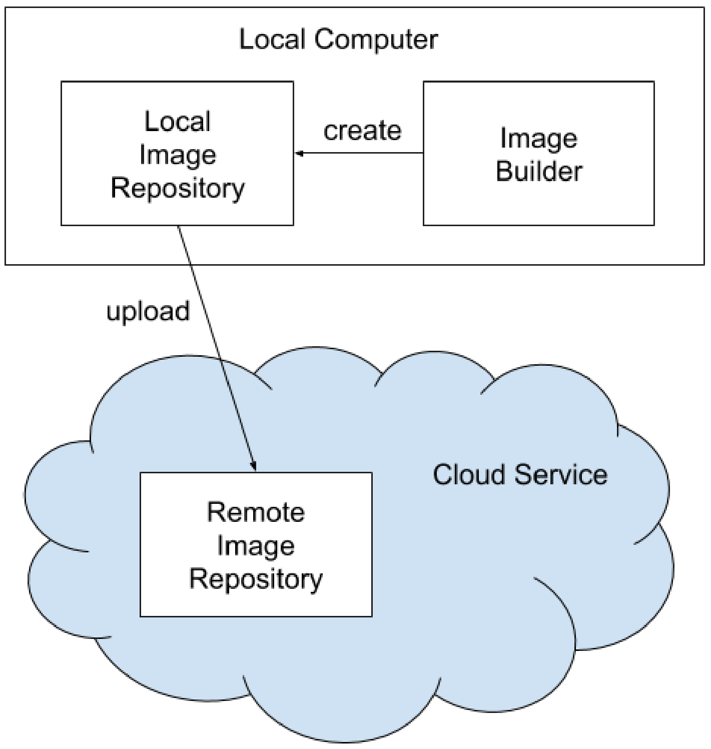

Figure 3.2: A local computer creates a new docker image. This image can then be uploaded to the cloud service.

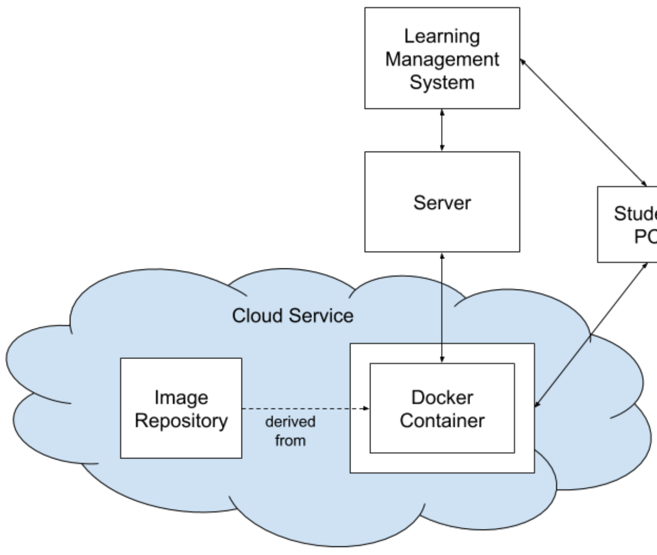

Figure 3.3: A student asks to access an exercise. The server spins it up from the template stored in the docker repository, then hands the student access to the exercise.

4 Implementation

4.1. Overview

Once the initial design of the system was finished, work on the implementation started. Several design goals needed to be accounted for as the system was developed. For example, to keep the system maintainable and secure, a modular structure was necessary, and testability, while it wasn't always possible to satisfy due to some outside services not being possible to simulate, was an important requirement as well.
In section 4.2 the implementation will be addressed from a high level perspective. After that, the inner workings of the system will be explained in more detail for every module.

4.2. Overall Structure

The system is divided in three main modules (section 4.4) that function independently and communicate with each other:

- The **ExerciseAgent** (section 4.4.1) sets up the exercise VMs inside the docker container, then monitors the exercise to see whether it is done. It behaves as the ExerciseAgent program as represented in fig. 3.1.
- The **Docker** (section 4.4.2) modules builds a docker image that contains all the exercise machines. It takes the role of the imagebuilder as represented in fig. 3.2.
- The **Server** (section 4.4.3) module provides a connection between the online learning environment and the cloud server that hosts the network of virtual machines, from user to platform. It behaves as the Server as represented in fig. 3.3.

However, before the details of these main modules are discussed, first the modules they are dependent on (section 4.3) are discussed as their details are necessary to understand the main modules. These are:

- The **Core** (section 4.3.1) module contains classes necessary for every module, in the form of data classes and utility methods.
- The **VMHandler** (section 4.3.2) boots up virtual machines and allows for communication between the many VMs and outside sources.
- The **NetworkBuilder** (section 4.3.3) tells the VMhandler to set up VMs, which it will connect with a network.

4.3. Dependency Modules

4.3.1. Core Module

The core module is a module that contains all classes that are used by multiple modules across the system. These are mainly simple classes containing data rather than functionality, and utility classes. It has three packages of classes that can be grouped together: exercise, server, and utility.

Exercise:
The Exercise package has all the classes that work with the Exercise class. Exercise Objects are the java representation of the YAML configuration files used by the rest of the system. Exercise objects have a list of Milestones, Hosts, and NetworkSegments, and the Firewall options. The YAMLLoader class takes a YAML file and extracts the exercise from it, in the structure of the Exercise class.

A Host is a class with data for a Host VM that needs to be created in the network, and it contains a string for the VM image required, a boolean for enabling vrdp support, a list of strings for packages that need to be installed, two strings for the paths to scripts that need to be executed before and after installing packages, two booleans that determine whether or not vpn dialin is required for students and for teachers, a list of Networks that are available to the host VM, and a list of files that need to be copied to the VM and their destination in the VM.

The Milestone class has info on a milestone in an exercise, and includes the string `host`, which is the name of the Host VM in which the milestone needs to be checked, the string `script`, which is the path to the script that checks whether or not a milestone has been reached, and the string `checkresult` which is the path to the checkresult file.

A NetworkSegment class contains the information on a network segment that needs to be created within the network, most importantly a list of Participant objects `participants` that are the nodes connected in the network. A Participant class has the name of a Host VM and the name of the network adapter. All machines on a network segment will be connected to each other using a switch.

A Firewall object simply contains a list of FirewallRules. A FirewallRule object contains a string `protocol` for the internet protocol to allow, a string `srcIP` for the source IP, an integer `srcPort` for the port on the side of the source, and string `destIP` with integer `destPort` for the destination.

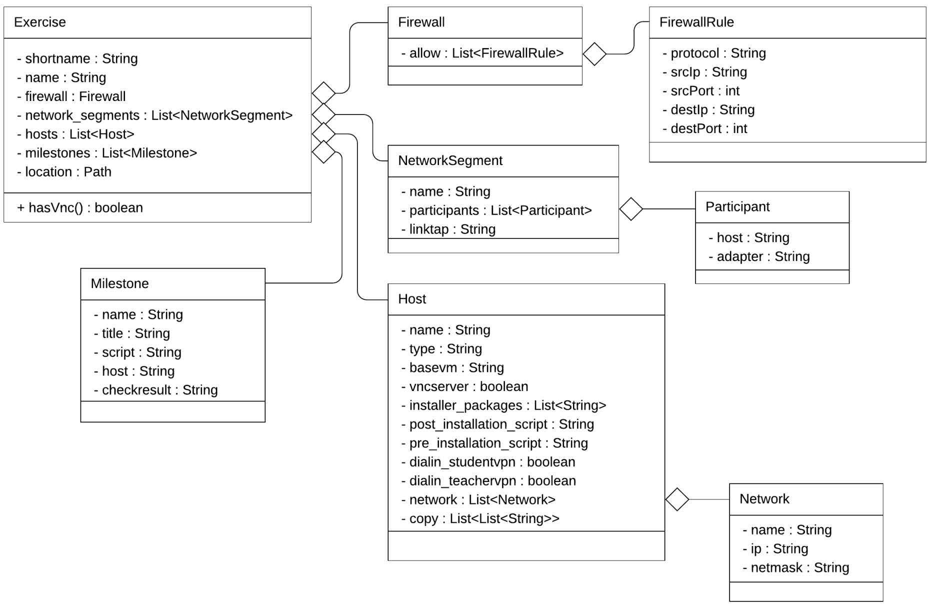

Figure 4.1: A UML Class Diagram of the Exercise class and the classes related to it.

Server:
The server.model.exercise package contains all classes with data required to show students their login credentials and exercise progress, as well as payloads sent to and from the server. The StudentList class contains the Exercise object that the students are working on, as well as the map of students with their IDs and progress as StudentExercise object. It has a function `getStudentExercise`, which retrieves an StudentExercise object by getting it from the map with a studentID, as well as an `addStudentExercise` function, which adds it to

the map by getting the StudentID from it.

The StudentExercise class stores all data the server needs to show a student their progress, report completion of the exercise back to the LMS and control their VMs. The class contains the credentials of a student for the VPN server, for each machine with RDP, and for each machine with ssh access. A Credential object has strings for a username, password, host name, IP address, and network port. It also contains a map with the names of milestones and the progress in them for that student. Its `getMilestone` function retrieves a MilestoneProgress object by looking into the `milestoneProgressList` map using the milestone's name.

The MilestoneProgress class contains a string for a name and for a full title, as well as a MilestoneStatus object. The MilestoneStatus class contains boolean `completed`, a float `percent` between `0` in `1` and with -1.0F for a not yet instantiated percentage, and a string `message` for an optional message.

The payloads sent to and from the server are two classes. The ExerciseCredentials class contains a list of rdp passwords (Credential class), a list of ssh passwords (Credential class), and a string for the url to the vpn file on the server. And the JsonResponse class has a string for the status of the api call, an Object representing the data returned from the api call, which is serializable to json, and an optional string for an error message.

Utility:
The Utility package contains utility classes, which are classes that just have static functions and cannot be instantiated. The FMWriter contains the `writeFromTemplate` function, which takes configurations in a map and uses FreeMarker to write them into an .ftl file. FolderUtility has the `getTempFolder` function, which creates an absolute path to a `/tmp` folder using the string prefix that is passed along with it. The LoggerUtils class contains the `loggerPrintStream` functions, allowing other parts of the code to make use of the logger. The PasswordUtility class has the function `generatePassword`, which creates a random string based on the given parameters, as well as the `hashPassword256` function, which hashes the password using SHA 256.

4.3.2. VMHandler Module

Since the VMHandler is used by multiple classes over multiple modules, the VMHandler class itself is a singleton class. This way every class can communicate the same single VMHandler and get access to the same set of virtual machines. The module can launch and shut down VMs and set up guest sessions with them for access into the VMs from outside. The FileHandler class can copy files and directories to the VM and create a shared folder between the VM and the system it's running on, which is necessary for the ExerciseAgent to keep track of progress in the exercises. The module also contains the ProgramRunner class which takes care of running programs on a virtual machine from outside, and the User class, which is a simple class representing a user, with username and password.

4.3.3. NetworkBuilder Module

The NetworkBuilder module's function is to set up the VMs and their specified network interfaces, so that students are be able to work on their exercise in that network.

The NetworkBuilder class sets up the VMs in the network. It takes the Exercise by reading a config file and then creates a VM for every host included in that exercise. It creates a new instance, adds a root user, copies necessary files for the exercise using the FileCopier class, installs packages given by the host, and shuts down the VM before moving on to the next host included in the Exercise. It also makes an object of the class MilestoneSetupHelper, which sets up a list of MilestoneSetup objects, then executes the `setUpMilestone` function on all of them. The MilestoneSetup class launches a VM and uses the FileHandler class from the VMHandler module to copy over the milestone files, make them executable and launch at startup. The ProgramInstaller class, used to install packages, also makes use of the FileHandler to copy the necessary files over and run the script to install it. ProgramInstaller is an abstract class extended by three classes: one for packages that require apt install commands, one for packages that require apt-get install commands, and one for OSes that don't support installs.

The IntNetBuilder sets up the connections between the VMs, it takes the exercise and all included hosts, shuts down all VMs with the names from the hosts, then it sets all network segments in place and it adds an network configuration file to every host.

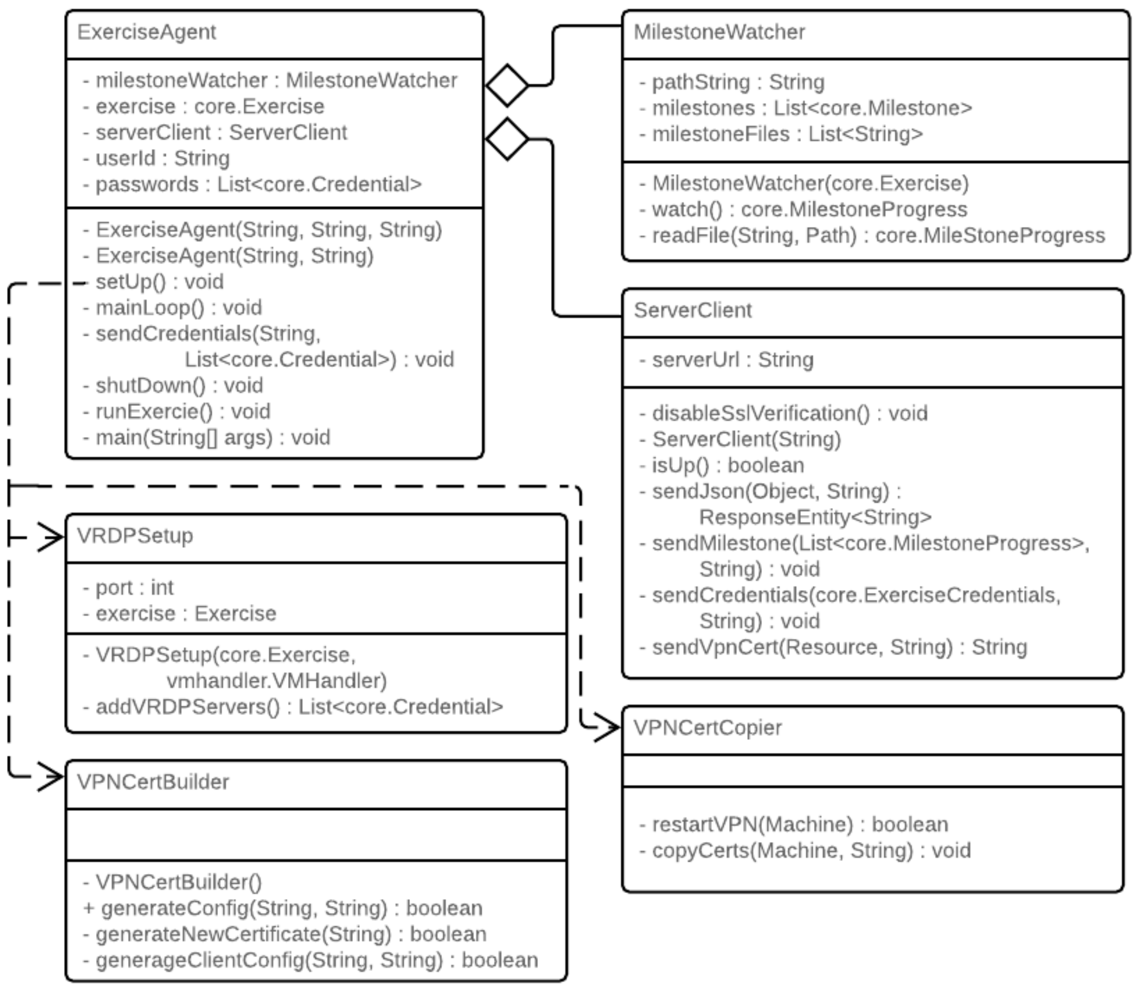
ExerciseAgent
- milestoneWatcher : MilestoneWatcher
- exercise : core.Exercise
- serverClient : ServerClient
- userId : String
- passwords : List<core.Credential>
- ExerciseAgent(String, String, String)
- ExerciseAgent(String, String)
- setUp() : void
- mainLoop() : void
- sendCredentials(String, List<core.Credential>) : void
- shutDown() : void
- runExercie() : void
- main(String[] args) : void
MilestoneWatcher
- pathString : String
- milestones : List<core.Milestone>
- milestoneFiles : List<String>
- MilestoneWatcher(core.Exercise)
- watch() : core.MilestoneProgress
- readFile(String, Path) : core.MileStoneProgress
ServerClient
- serverUrl : String
- disableSslVerification() : void
- ServerClient(String)
- isUp() : boolean
- sendJson(Object, String) : ResponseEntity<String>
- sendMilestone(List<core.MilestoneProgress>, String) : void
- sendCredentials(core.ExerciseCredentials, String) : void
- sendVpnCert(Resource, String) : String
VRDPSetup
- port : int
- exercise : Exercise
- VRDPSetup(core.Exercise, vmhandler.VMHandler)
- addVRDPServers() : List<core.Credential>
VPNCertCopier
- restartVPN(Machine) : boolean
- copyCerts(Machine, String) : void
VPNCertBuilder
- VPNCertBuilder()
+ generateConfig(String, String) : boolean
- generateNewCertificate(String) : boolean
- generageClientConfig(String, String) : boolean

Figure 4.2: A UML Class Diagram of the exerciseagent module

The NetworkBuilder also has a separate package containing two classes that establish connections. The GCConnection class builds a connection with Google Cloud, and the SSHConnection class for making SSH connections with the host as a user.

4.4. Main Modules

4.4.1. ExerciseAgent Module

The ExerciseAgent module's executable ExerciseAgent class first creates an instance of itself and calls `runExercise` on it. This function calls the `setUp` function, which sets up the whole exercise, including the VMHandler, the necessary credentials and the machines for each host in the exercise. setUp creates a VRDPSetup object and calls its `addVRDPServers` function, which enables vrdp support for all the hosts that need it and then returns the list of passwords. It then takes the path to a temporary folder, generates VPN keys within that folder through the VPNCertBuilder's `generateConfig` function, and copies all keys in that folder over to the virtual machine with the VPCertCopier's `copyCerts` function. The `runExercise` function then continues by starting a while loop in which the shared folder with milestone progress files is watched. If one of the milestone json files changes, the MilestoneWatcher class will pass the json data onto it, after which it sends it over to the ServerClient, which turns it into a post request for the server.

4.4.2. Docker Module

The main purpose of the Docker module is to create a full docker image for an exercise. This image can then be uploaded to the cloud or run locally to provide an exercise environment for the student.

The Docker module consists of 3 classes:

- PackageBuilder: Runs a task to package the ExerciseAgent module into a jar
- DockerBuildFolder: Prepares a temporary folder with all the resources necessary to build the docker image. Calls PackageBuilder.
- DockerBuilder: The run point of this module. Calls DockerBuildFolder to prepare a temporary folder, then executes a docker build task to build the image.

To run the DockerBuilder module, the user must have a working version of Docker installed. Once the DockerBuilder class has finished running, the created image is locally registered with a random id and the user is given a docker image id. Using this id, the docker image can be added to a remote docker repository, for example the one provided by a cloud provider, as described in section 3.5.

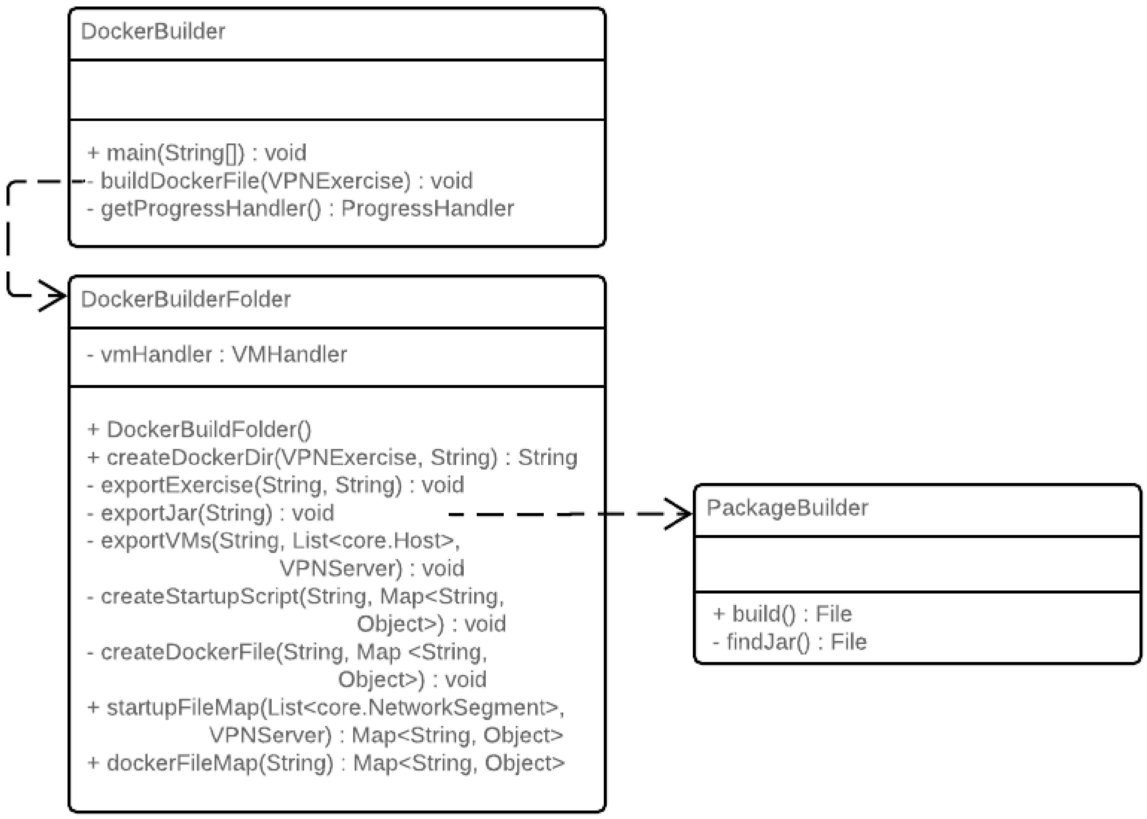
DockerBuilder
+ main(String[]) : void
- buildDockerFile(VPNExercise) : void
- getProgressHandler() : ProgressHandler
DockerBuilderFolder
- vmHandler : VMHandler
+ DockerBuildFolder()
+ createDockerDir(VPNExercise, String) : String
- exportExercise(String, String) : void
- exportJar(String) : void
- exportVMs(String, List<core.Host>, VPNServer) : void
- createStartupScript(String, Map<String, Object>) : void
- createDockerFile(String, Map <String, Object>) : void
+ startupFileMap(List<core.NetworkSegment>, VPNServer) : Map<String, Object>
+ dockerFileMap(String) : Map<String, Object>
PackageBuilder
+ build() : File
- findJar() : File

Figure 4.3: An UML Class Diagram of the docker module

4.4.3. Server Module

The server module contains the main server of the system which handles requests coming from the LTI and any exercise agents belonging to a particular exercise. This module also contains the cloud interface package containing the necessary functionality to communicate with the cloud providers.

Server

The server module contains a Spring application, which means there is an executable Application class that boots it up, and Controller classes that execute functions when the application receives a request.

The AppController class has a function called `launch`, which retrieves the user data from the LtiVerificationResult and creates a StudentExercise for the user when a post request is received at `/launch`. The function then redirects the user to `/index`, which the MoodleController listens to. This handles the launch requests from the LMS.

The MoodleController has the `index` function, which is triggered when a post request is received at `/index`. It takes all of the user information from the StudentExercise (name, ID, milestones, etc.) and adds them to the model of the html page shown to the user. This page is shown to a user and shows them the login information they need to connect to the exercise and their progress in the exercise, the page updates automatically using AJAX.

AgentController has the function `processVpnCert` when receiving a post request at `/vpn` and lets the ExerciseAgent upload VPN certficates. After this the ExerciseAgent module will send a post request to `/rdp`, to pass credentials along with the VPN file, which is caught by the `processCredentials` function.
When a post request is received at `/ex/milestone/status`, the `reportMilestone` function is triggered. The JSON data of the milestones completion values is read by the function, which then updates the data for each MilestoneProgress on the webpage for that student.

The Config and DataConfiguration classes are Spring configuration classes, which contain Spring Beans. These are functions that are instantiated at the startup of the application. The Config class has the `servletContainer` bean, which allows for the system to request a key from the keystore for a secure connection, and redirects the connection to a different outgoing port for HTTPS. The DataConfiguration class has the beans `getExercise`, to load an exercise from the given .yaml file, and `getList`, which creates a single student list for the default exercise.

Cloud Interface
The cloud interface package is responsible for all communication with the implemented cloud providers. A summarized overview of the main components is given in fig. 4.4.

ContainerHandler is an universal class capable of performing all docker registry related actions. Our system uses private docker registries to host and store the base exercise images used to spin up exercises. When instantiating a ContainerHandler object, a RegistryAuthSupplier object is passed. This supplier contains all the necessary authentication details for the specific private docker registry.

ContainerRunner is an abstract class defining the necessary methods needed to spin up an exercise on the cloud, or locally. Extending classes are forced to implement the runContainer and closeContainer methods. How these methods are implemented of course depends on the chosen cloud provider. In this project, this functionality is implemented for Google Cloud and Microsoft Azure. ContainerRunner depends on a ContainerHandler object to push the locally generated exercise images to the relevant docker registry.

To make the system as modular as possible, any instantiation of ContainerHandler will work, but optimally you would want to instantiate a ContainerRunner with a ContainerHandler storing the images as close as possible to this runner. The system automatically registers the optimal ContainerHandler to the specified ContainerRunner, but the option to specify a custom handler is there if necessary.

In the auth package, all functionality needed to authenticate with the docker registries and cloud providers is provided.

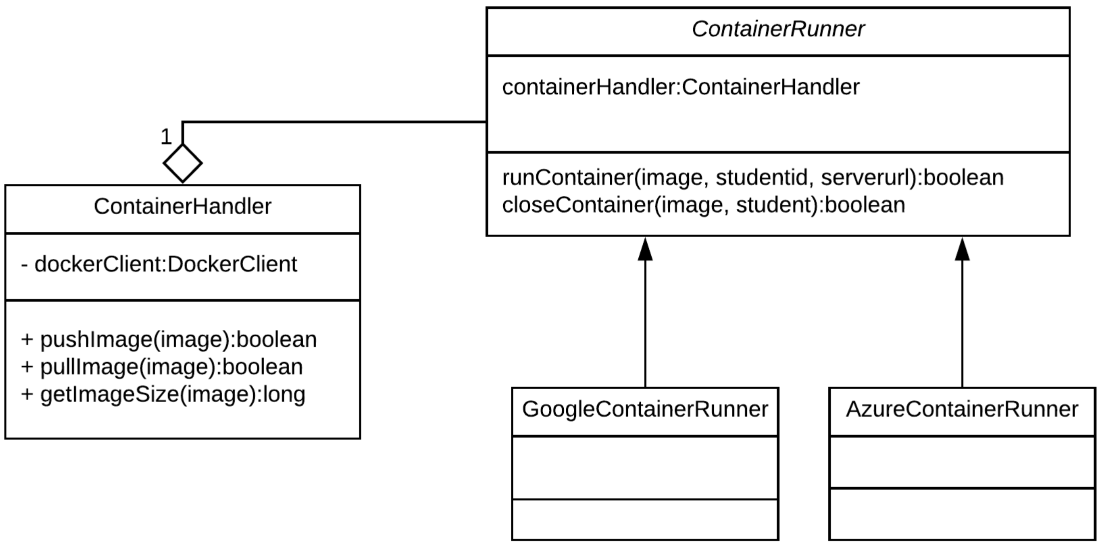
ContainerRunner
containerHandler:ContainerHandler
runContainer(image, studentid, serverurl):boolean
closeContainer(image, student):boolean
1
ContainerHandler
- dockerClient:DockerClient
+ pushImage(image):boolean
+ pullImage(image):boolean
+ getImageSize(image):long
GoogleContainerRunner
AzureContainerRunner

Figure 4.4: An UML Class Diagram of the cloud interface main components

5 Testing

An important part of software development is testing. The system was tested in multiple ways, first during development unit tests were written for the code, and at the end of the project the system was tested to see if requirements were met.

5.1. Unit testing

During development Unit tests were written for the system. These tests were automatically run by the Continuous Integration(CI) server on every push and merge. These tests also generated coverage reports by using Jacoco. All trivial code, like getters and setters, automatically generated by Lombok was ignored in these reports.

5.1.1. core

This module has a test coverage of 33%, but a branch coverage of 100%, because the untested code is all in the utility package which mostly contains helper methods to make the rest of the code easier to read.

5.1.2. vmHandler

The vmhandler module has an instruction coverage of 55% and a branch coverage of 41%. Testing this module required a lot of mocking since the automated tests in the CI environment could not use the virtualbox API which this module relies on. As a result the coverage for this module is lower.

5.1.3. networkbuilder

This module has a 38% instruction coverage and 41% branch coverage. Most of the untested code in this module is in the connection to the cloud providers and the NetworkBuilder class itself, this class uses the other classes in the module to create the virtual machines.

5.1.4. server

This module has an instruction coverage of 33% and a branch coverage of 38%, a part of the untested code consists of fairly simple methods to return some data when requested by the other components of the system, and the methods that return the webpages that are displayed to students.

5.1.5. exerciseagent

This module has a low test coverage with 7% instruction coverage and 10% of branches being covered by automated tests. Most of this module was not tested automatically but only tested manually while developing the program,

5.1.6. docker

This module has no automated tests and was only tested manually.

5.2. Testing requirements

To test if the system met the requirements multiple tests were designed, the first one to see if the system could handle setting up and running larger networks, a second one to test the system when working with multiple students at the same time.

5.2.1. large networks

To test if the the system can handle generating and running larger and more complicated networks a test exercise was created. This exercise has 30 computers, the computers were all part of the same simple network and ran simple scripts to simulate activity. This test consists of two parts, first building the exercise and then running it.

This test has not been run yet, the system should be able to generate an image like this without issues since only one VM runs at a time during image generation, and since the VMs would mostly be idle running them all should also be possible with the system, the size of the image might be an issue for the system because each VM currently uses multiple gigabytes on disk.

5.2.2. multiple students

Testing if the system could handle multiple students at once was done by letting the system setup 10 instances of the port scanning exercise, this exercise has three virtual machines, one of them runs the vpn server, a second one is used by the student and the third one runs multiple services and should be scanned by the student to discover the open ports. No student activity was simulated since the instances constantly contact the server with progress reports and this does not change with activity.

While running this test some issues with running multiple instances on the same machine, and a lack of logging of errors were discovered. However when these were fixed and the test was run again the server was able to handle 10 exercise agents and 10 pages without it affecting the response time of the server.

6
Final Product Evaluation

In this chapter, the final product shall be discussed, and how far it has matched the design goals and project requirements.

6.1. Evaluation of Functional Requirements

In this section the list of functional requirements shall be given. For each requirement it shall be explain whether the requirement has been met, in which way it has been met (with a reference if it has been mentioned earlier in the report), and if it has not been met, why.

6.1.1. Must Haves

Description	**Complete**	**Justification**
The product must be able to automatically claim resources from a cloud provider.	Yes	The server has a connector to run containers on the cloud (section 4.4.3)
The product must be capable of instantiating cyber ranges on at least two cloud providers.	Yes	The cyber range is compatible with Azure and Google Cloud (section 4.4.3)
The product must have a universal standard for describing virtual machines, and only differ in how this information is communicated to the cloud provider.	Yes	The exercise environment is described in the form of a yaml file (section 4.3.1), which is interpreted by the docker (section 4.4.2) and network-builder(section 4.3.3) modules respectively.
The product must be able to switch between the cloud providers used to host the exercise. This must not take more than 10 minutes of manual work.	Yes	This can be done by resetting the server with different settings.
The product must automatically shut down once the exercise has been completed (i.e. unclaim the used resources on the cloud provider).	Yes	The server shuts down the instance when all milestones have been completed at 100%.
The product must automatically shut down after a period of inactivity.	Yes	The server shuts down the instance when a student has performed no activity on it for 10 minutes.

Table 6.1: Functional requirements for the Cloud Provider

Description	Complete	Justification
The client virtualisation must be based on Docker/Virtualbox.	Yes	The machines in the network are simulated using Virtualbox.
The product must be able to read a configuration from a file. The file lists IP addresses, connections between VMs (Virtual Machines), and which scripts should be run on each host.	Yes	The exercise is defined by a yaml file. (section 4.3.1)
The product must have a universal standard for describing virtual machines, and only differ in how this information is communicated to the cloud provider.	Yes	The exercise is defined by a yaml file (section 4.3.1)
The network must have configurable access points where network traffic can be tapped. These taps should not be able to be moved or changed by the end user.	Yes	Taps are safe from tampering as long as the student is not given root access on the computer they log in to with the VPN.
The product must be able to configure at least one login via VPN.	Yes	New VPN keys are generated for each exercise instance (section 4.4.1)
A user must be able to get the VPN login from Moodle.	Yes	The server passes the VPN keys on to moodle (section 4.4.3)
The system must be able to control which kinds of network traffic are allowed into and out of the virtual network.	Yes	Firewall rules can be defined in the yaml file, which are then applied in the docker image.

Table 6.2: Functional requirements for the Network

Description	Complete	Justification
Any exercise must be able to run locally on the student's computer.	No	This requirement was adjusted as it was deemed unrealistic by developers and client. However, any exercise can be run on a local TU Server.
An exercise must be able to support 30 virtual machines running simultaneously.	No	Untested
A TA (Teaching Assistant) can view the status of the exercises as they are running (CPU load, etc.)	No	Memory/CPU use can be viewed from the cloud dashboard, and on Moodle the TA can view the grade.
A student or TA must be able to reset/shutdown the exercise.	No	The TA is able to shut down the exercise from the cloud dashboard, however it is not graceful.

Table 6.3: Functional requirements for the Exercises

Description	Complete	Justification
The network supports the injection of IP (Internet Protocol) and MAC (Media Access Control) address spoofed traffic.	No	Though it is possible to run a script inside the docker container that injects traffic into the bridges, there is no way to define this in the exercise.
The product must be able to be showcased for a small example exercise.	Yes	Exercises that are built can be run locally for a showcase as long as there is a running server.

Table 6.4: Global product requirements

6.1.2. Should haves

Description	**Complete**	**Justification**
The system should have the ability to install a specific version of software packages.	Yes	The system can pass version arguments to the package manager
The system should be able to start from a frozen VM.	Yes	The system is capable of loading existing VM images as they are.
Network configurations must be able to be combined and reused for later (larger) scenarios.	No	This would severely affect the design of how networks would be built. It was decided with the client to drop this requirement.

Table 6.5: Network Configuration requirements

Description	**Complete**	**Justification**
A TA should be able to view the network as the student is working on it.	Yes	It is possible, as a TA, to view the student's VPN keys and log in using them.
A machine should be able to shut down automatically when a student exceeds a certain amount of activity.	No	This would require being able to monitor all the individual VMs in an exercise, which the server is currently incapable of.

Table 6.6: Global product requirements

6.1.3. Could Haves

Description	**Complete**	**Justification**
"Multiplayer": The product could provide interaction between users, where one person sends a network attack to another user, who protects their network against it.	No	Deemed out-of-scope for the current project.

Table 6.7: Could haves

6.1.4. Won't Haves

Description	**Complete**	**Justification**
The product won't be integrated with Brightspace.	Yes	Product was integrated with Moodle instead.

Table 6.8: Won't haves

6.2. Evaluation of Design Goals

In section 2.2, three main design goals were defined. If these design goals could be reached, the project could be called a success. In this section, the design goals shall be examined and for each it shall be decided whether the goal has been reached adequately.

6.2.1. Security

Security as a design goal is achieved by testing the code, and by isolation of exercises.

Testability
As the project progressed, full testability of code became difficult. This is because the product requires many different kinds of software. For example, it was not possible to write tests to guarantee that a command sent to a virtual machine would execute correctly. To ensure maintainability nonetheless the team adjusted the log messages to be very informative, so that a new developer would be able to react to a problem as it appeared.

Isolation
Isolation of the exercises was achieved through having the exercises run in containers. One of the things containers ensure is that only permitted traffic is able to leave the container. This allows exercises not to interfere, even if they run on the same computer. Additionally, this prevents traffic that is not supposed to leave the container from leaving, as long as the exercise is well-defined.

6.2.2. Maintainability

As discussed in section 2.2.2, the code should be modular and easy to extend. Especially since use cases may change (the university may start using a different Learning Management System, or a new cloud service should be compatible with the system). This has been achieved by making as little as possible in the code depend on one specific architecture. For Learning Management Systems, it was decided to implement the LTI standard, making the application compatible with many types of LMS.

For cloud compatibility, it was decided to have the connector to the cloud be concentrated into a single class. Therefore, as much of the code as possible could be shared, and only the part of the code that actually communicates with the cloud service needs a different connector. Then, a teacher must to upload the docker images of the exercises into the cloud service repository to be able to use the cloud service.

6.2.3. Scalability

As most of the heavy work of the system is concentrated in the cloud, the server is only responsible for the communications between the Learning Management System and the docker container, which are simple HTTP requests. Therefore, the work of the server will not increase by much. The cloud instances, meanwhile, are able to work independently from one another. One scalability problem is that the exercise images can take up quite a bit of disk space (one of the simple exercises took up 14GB with 3 virtual machines). Therefore storage of many exercises may be a problem if the size of the virtual machines cannot be reduced.

6.3. Evaluation of Success Criteria

The goal of maintainability has been met, as the system has successfully designed to be very modular.

Security is somewhat more difficult to assess, due to the many surfaces of attack which are possible on the system. This also differs depending on the chosen interpretation of security. From a student's perspective, the system is quite secure. It should not be possible for a student to interfere with the exercise of another student in any way. This was the main focus point regarding security for our project. As such, the system has achieved this design goal as well.

Lastly, the goal of scalability has been met too, as the system utilizes several cloud providers to deploy and host exercises. This means that all of the design goals have been met, and as such, all of the success criteria.

7 Ethical Implications

In this section, ethical issues regarding this product and it's uses will be discussed. Of course, we do not condone malicious use of the product but we cannot control the usage either.

First of all, though the system is intended to be used to teach students about cybersecurity, it is also possible to use this system to instruct many people at once in how to abuse the flaws in a computer network. It is not possible for us to mitigate this as one of the reasons this system was built is to allow students to experiment in an environment where breaking a machine in the isolated network has no consequences on the outside world.

Another problem that is that an exercise that is not very well-configured may lead to unintended side-effects. Though the privileges in an environment are as limited as possible by default, it is still possible that the configuration allows for abberant behavior. For example, if an exercise involves a DDOS-attack, and the teacher opened up the ports of some machines in the network to allow them to access the internet, the student may be able to redirect the DDOS-traffic to an uninvolved party outside the exercise environment.

A third problem is regarding the use of cloud services. Though the system does not send any personal information to the cloud service (students are identified by a number), it is still possible for the cloud service to identify the student's computer as they connect to the exercise. Not unnecessarily sharing a student's data is one of the reasons why the exercise should also be able to run locally on a machine belonging to the TU Delft.

Even though we attempted to prevent misuse, there are still some problems with this system we cannot prevent. Therefore, we would like to ask users to be careful when using this system and not to abuse it.

8 Process evaluation and recommendations

This chapter will evaluate the process used during the project and how the project can be expanded upon in the future.

8.1. Process Evaluation

During this project, the scrum methodology was used to satisfaction of the group members, using 1-week sprints. It was flexible enough to be adaptable as priorities in the project shifted.

The code was hosted on Gitlab, with project management and documentation being done using Gitlab's inbuilt issue board and wiki.

Every week, 2 meetings were held with the project's client, Christian Doerr, who worked in the same building. Aside from these face-to-face meetings, there was also frequent digital communication over the digital application Rocket. This high level communication allowed the team to get feedback on new features quickly, and enabled the team and the client to have closely aligned expectations for the project.

However, there were also some problems. The level of communication inside the group was rather low, and teammates would rarely work on the same feature. This caused team members who were stuck to spend longer than necessary on their features. It also caused some problems as separately built features had to be integrated, leading to some redundant code. Because teammates would not often use each other's features until later this meant that instances of hardcoding were not corrected as soon as they should have, making the product less modular.

In the future, aside from planned meetings with the supervisor there should be planned meetings with the group and more focus on integrating the created features with one another.

In conclusion, though the group was able to have a satisfactory relationship with the supervisor there should have been a focus on communication inside the group.

8.2. Future work and Recommendations

There are some extensions to this project that have not been implemented in this project due to time constraints or being out of the scope of this project, but that are still valuable additions.

An extension that would greatly improve the flexibility of the project is the possibility to spin up docker containers to simulate machines in the network. Virtual Machines take up relatively much disk space (for this project, even simple 3-machine exercises would cost 14 GB of disk space) and the full capabilities they offer are not always necessary. For example, if one would like to have 20 machines performing a simple task such as sending HTTP requests to a server running on one of the machines, it would be preferable to have these 20 machines represented as docker containers. It would cost minimal effort to implement this, as OpenVSwitch is agnostic to the exact devices it connects, and merely passes messages between IP addresses. Another method to reduce disk usage could be to use linked clones for virtual machines to reduce duplicated data.

Another extension that would improve the flexibility of this product is the ability to have multiple students connect to the same exercise. Currently, only a single student can connect to an exercise environment, but this could be extended by generating a second pair of VPN keys and a way to designate which student will be allowed to connect to which machine. This also opens the possibility for cooperative and adversarial exercises.

9 Conclusion

In this project, we set out to create a cyber range to allow students to learn practicals skills related to cybersecurity. First, we did 2 weeks of research to scope out existing solutions and gather the necessary info to come up with an effective design. Having done this research, the design laid out in section 2.5 was built.

Our chosen design relies on several independent APIs, and can be seen as several independent modules. Familiarizing ourselves with these APIs, as well as making sure their relevant components were integrated well, proved to be challenge at first. For some APIs, available documentation is sparse, and in many cases our use cases for them were quite specialized. After a few weeks, however, our knowledge in using them was sufficient, and the main components of our system could be built.

The built system solves the core of our problem definition laid out in section 2.1. Some minor requirements might still be missing. But the base functionality necessary to facilitate network security exercises on the cloud exists. Moreover, the system is built to be as modular and extendable as possible. Adding features, or improving existing ones should not prove very difficult.

In all, this means that the design goals laid out in section 2.2 have been met. Furthermore, the system has sufficiently solved the problem laid out in the problem definition. Of course, due to time limitations, and some setbacks caused by minor miscommunications (described in chapter 8), the system is not as complete as initially hoped. Nevertheless, all of the above leads us to believe that the project can be called a success.

Appendices

A
Original Project Description

A secure environment for hands-on networking exercises

Obtaining hands-on practice with computer networks is difficult. Most students do not own multiple computers to set up and play with networks at home, while on campus such resource-intensive exercises can only be provided for a short time. The goal of this context project is to build a system where students can "own" and practice in their own private network, based on the resources leased from a cloud provider. For this environment, the team will need to develop a software that can spin up and configure a set of virtual machines based on a machine-readable configuration file at at least two cloud providers, and configure a VPN access so that students can log in. The software also monitors the virtual network so that the resources can be released as soon as the exercise objective is accomplished or the network is no longer accessed. Finally, the team also integrates an interface using the Learning Management Systems standard (for which a software library is available), through which the hands-on environment communicates with Canvas. This means that information how to log into the hands-on environment is shown in the e-learning environment, and the cloud reports back when the exercise is completed.

Required features:

- configurations can be specified in a configuration file, which lists network configuration (IP addresses, connections between VMs), and scripts which should be run on each host
- cloud independent: system is compatible with at least 2 cloud providers, and able to realize the environment through virtual machine images, private/public VM networks and docker images (configurable by scenario depending on required resources/capabilities, but definition should be general enough to allow transparent switching)
- configurations can be stored in "libraries", so that these building blocks can be recycled as part of larger scenarios
- demo networks are isolated from the Internet, can configure if no traffic may leave the virtual network, or whitelist some parts of the traffic (e.g. DNS and HTTP only)
- support the injection of IP and MAC address spoofed traffic
- user dials in by VPN (openVPN?) to specific, configurable point into the network
- system can collect pcap traces at configurable interfaces or locations in the network, realized such that it is outside of the scope of influence of the users of the exercise
- connection via the LMS standard, via this interface progression updates (exercise started, exercise solved) can be pushed to be shown in brightspace ; information about how the access the VPN (created username/password, VPN endpoint) is exported via this to brightspace
- monitors whether users are logged in, spins down network environment if no longer used (configurable timeout)

- definition and validation of "unit tests": specific milestones that should be reached as part of the exercise and that the system can monitor for (when provided with an external code snipplet); for example on an ARP poisoning scenario: milestone 0: packet injection tool compiles and starts up; milestone 1: injection of an arbitrary, spoofed packet, milestone 2: packet that will add an entry to the ARP table; milestone 3: ARP poisoning attack working; these are defined in the sense of unit tests
- showcase the system through a small test network with vulnerable machines

  Nice to have:

  - ability to install software packages from rpm / deb source by version, for example apache@2.0.33 or openssl@1.0.1f

B
Infosheet

General Information:

Title of the Project: Clusus
Name of the client organisation: TU Delft Cyber Security Group
Date of the Final Presentation: 05-07-2019

Description:

The students of TU Delft have access to cybersecurity courses. For the students to be able to gain more practical experience with cybersecurity issues, a new environment is necessary for students to practise in. Clusus is a system for teachers to create, configure and deploy networks of virtual machines, and for students to access these machines. The main challenge of this project was deciding how the network of machines should be simulated, since this would affect the entire design. During the research phase of the project much attention was directed towards this subject. We examined at various existing tools and their solution to the problem before deciding to implement our own product using existing VM and networking software.
To allow the team more flexibility to adjust the course of action, the agile Scrum methodology was used. Due to this, it was possible to adjust the plan quickly if a problem was discovered.
The final product consists of multiple subprojects, namely: 1. An application that builds a virtual environment from a specification file. This virtual environment can be hosted on a cloud or on a local machine. 2. A server that connects to a Learning Management System. 3. An application that runs inside the virtual environment to monitor the student's actions.
In the final version, certain features have been omitted due to time constraints. However, the report details upgrades to the current functionality and how these may be achieved.

Members of the Project Team:

Name: Esmee Hildebrand
Interests: Algorithmics, Software testing, Project management
Contribution and role: Virtual Machine Networking, Docker

Name: Joas Mulder
Interests: Data Analytics, Machine learning, Language Design
Contribution and role: Cloud interfacing, Virtual machine setup, Exercise monitoring

Name: Albert Smit
Interests: Software engineering, Robotics
Contribution and role: Virtual machine setup, Server and Webpages, Exercise monitoring

Name: Remi Flinterman
Interests: Artificial Intelligence, Game Development, Robotics
Contribution and role: Server and integration with learning management system

Client:

Name: Dr. Christian Doerr
Affiliation: TU Delft Cyber Security Group

Coach:

Name: Dr. Stefanie Roos
Affiliation: TU Delft Distributed Systems Group

Contact:

Esmee Hildebrand, 34hildebrand@gmail.com
Joas Mulder, joasmulder@hotmail.com
Albert Smit, albertsmit93@gmail.com
Remi Flinterman, remiflint@live.nl

The final report for this project can be found at: http://repository.tudelft.nl

C
Software Improvement Group evaluation

In week 5 and week 9 of the project, it was required by the TU Delft that the code be sent to the Software Improvement Group[1] for a code quality review. This chapter lists the reviews (in Dutch) and explains what was done to improve the software after the week 5 review.

C.1. Week 5 evaluation

[Feedback]
De code van het systeem scoort 3.3 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code marktgemiddeld onderhoudbaar is. We zien Unit Interfacing en Unit Size vanwege de lagere deelscores als mogelijke verbeterpunten.

Voor Unit Interfacing wordt er gekeken naar het percentage code in units met een bovengemiddeld aantal parameters. Doorgaans duidt een bovengemiddeld aantal parameters op een gebrek aan abstractie. Daarnaast leidt een groot aantal parameters nogal eens tot verwarring in het aanroepen van de methode en in de meeste gevallen ook tot langere en complexere methoden. Dit kan worden opgelost door parameter-objecten te introduceren, waarbij een aantal logischerwijs bij elkaar horende parameters in een nieuw object wordt ondergebracht. Dit geldt ook voor constructors met een groot aantal parameters, dit kan een reden zijn om de datastructuur op te splitsen in een aantal datastructuren. Als een constructor bijvoorbeeld acht parameters heeft die logischerwijs in twee groepen van vier parameters bestaan, is het logisch om twee nieuwe objecten te introduceren.

Voorbeelden in jullie project:

- MilestoneSetup.MilestoneSetup(FileHandler,ProgramRunner,Milestone,Machine,String)

Bij Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Dit kan verschillende redenen hebben, maar de meest voorkomende is dat een methode te veel functionaliteit bevat. Vaak was de methode oorspronkelijk kleiner, maar is deze in de loop van tijd steeds verder uitgebreid. De aanwezigheid van commentaar die stukken code van elkaar scheiden is meestal een indicator dat de methode meerdere verantwoordelijkheden bevat. Het opsplitsen van dit soort methodes zorgt er voor dat elke methode een duidelijke en specifieke functionele scope heeft. Daarnaast wordt de functionaliteit op deze manier vanzelf gedocumenteerd via methodenamen.

Voorbeelden in jullie project:

- SSHConnection.SSHConnection(String,String,String,String)
- Machine.shutDownVM()
- MilestoneSetup.setUpMilestone()
- NetworkBuilder.createVM(Host,Exercise)

[1] https://www.softwareimprovementgroup.com/

De aanwezigheid van testcode is in ieder geval veelbelovend. De hoeveelheid testcode ziet er ook goed uit, hopelijk lukt het om naast toevoegen van nieuwe productiecode ook nieuwe tests te blijven schrijven.

Over het algemeen is er dus nog wat verbetering mogelijk, hopelijk lukt het om dit tijdens de rest van de ontwikkelfase te realiseren.

C.2. How this feedback was addressed

The feedback given by the Software Improvement Group was fairly straightforward. As such, making the necessary fixes was quite straightforward as well. The example classes/methods given by SIG were refactored into smaller, more logical units. This lead to an easier to understand system, as well better testable units. The SSHConnection class which was mentioned was refactored too, but eventually removed from our system, as its functionalities were no longer needed. This mainly due to switching to dockerized exercise instances.

In all, the feedback was taken seriously, and helped us recognize some of the more weaker components of our system. Afterwards, more care was taken into making sure system units were interfaced well, whilst keeping their size manageable.

C.3. Week 9 evaluation

In de tweede upload zien we dat het codevolume is gegroeid, terwijl de score voor onderhoudbaarheid is gestegen.

We zien dat de verbeterpunten uit de feedback op de eerste upload zijn aangepast, en op deze gebieden is dan ook een verbetering in de deelscores te zien. Het verschil is overigens niet heel groot, maar er is wel degelijk activiteit op deze gebieden te zien.

Het is goed om te zien dat er naast nieuwe productiecode ook nieuwe testcode is geschreven.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject.

Bibliography

[1] What is docker? https://aws.amazon.com/docker/, . Accessed: 2019-06-25.

[2] Aws free tier. https://aws.amazon.com/free/, . Accessed: 2019-06-25.

[3] Running hyper-v on amazon ec2 bare metal instances. https://aws.amazon.com/blogs/compute/running-hyper-v-on-amazon-ec2-bare-metal-instances/, . Accessed: 2019-06-25.

[4] Aws answers: Vpc security capabilities. https://aws.amazon.com/answers/networking/vpc-security-capabilities/, . Accessed: 2019-06-25.

[5] Nested virtualization in azure. https://azure.microsoft.com/de-de/blog/nested-virtualization-in-azure/. Accessed: 2019-06-25.

[6] Github - cyris. https://github.com/crond-jaist/cyris. Accessed: 2019-04-30.

[7] The deter project. http://deter-project.org/, . Accessed: 2019-04-24.

[8] About deterlab. https://deter-project.org/about_deterlab, . Accessed: 2019-04-24.

[9] Emulab. https://www.emulab.net/portal/frontpage.php. Accessed: 2019-04-24.

[10] Alias ip ranges overview. https://cloud.google.com/vpc/docs/alias-ip, . Accessed: 2019-06-25.

[11] Containers on compute engine. https://cloud.google.com/compute/docs/containers/, . Accessed: 2019-06-25.

[12] Google cloud platform free tier. https://cloud.google.com/free/, . Accessed: 2019-06-25.

[13] Using vpc flow logs. https://cloud.google.com/vpc/docs/using-flow-logs, . Accessed: 2019-06-25.

[14] Enabling nested virtualization for vm instances. https://cloud.google.com/compute/docs/instances/enable-nested-virtualization-vm-instances, . Accessed: 2019-06-25.

[15] Docker on azure. https://azure.microsoft.com/en-us/services/kubernetes-service/docker/, . Accessed: 2019-06-25.

[16] Create your azure free account today. https://azure.microsoft.com/en-us/free/, . Accessed: 2019-06-25.

[17] Manage packet captures with azure network watcher using the azure cli. https://docs.microsoft.com/en-us/azure/network-watcher/network-watcher-packet-capture-manage-cli, . Accessed: 2019-06-25.

[18] Moodle. https://moodle.org/. Accessed: 2019-04-24.

[19] Nested virtualization in azure. https://azure.microsoft.com/de-de/blog/nested-virtualization-in-azure/. Accessed: 2019-06-25.

[20] Try for free. https://cloud.oracle.com/tryit, . Accessed: 2019-06-25.

[21] High performance nested virtualization. https://cloud.oracle.com/ravello/technology/virtualization, . Accessed: 2019-06-25.

[22] Rackspace. https://docs.docker.com/machine/drivers/rackspace/, . Accessed: 2019-06-25.

[23] Network security in the cloud. https://blog.rackspace.com/network-security-in-the-cloud, . Accessed: 2019-06-25.

[24] How to use promiscuous mode on aws: using port mirroring for packet capture in the cloud. https://blogs.oracle.com/ravello/packet-capture-on-aws,.

[25] Man-in-the-middle network security testing on enterprise environment replicas in aws google cloud. https://blogs.oracle.com/ravello/mitm-cyber-security-lab-on-cloud, . Accessed: 2019-06-25.

[26] Malicious software and spoofed ip addresses. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/sect-security_guide-firewalls-malicious_software_and_spoofed_ip_addresses, . Accessed: 2019-06-25.

[27] Hardening infrastructure and virtualization. https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/14/html/security_and_hardening_guide/hardening_infrastructure_and_virtualization, . Accessed: 2019-06-25.

[28] Evaluate red hat openstack platform. https://access.redhat.com/products/red-hat-openstack-platform/evaluation?extIdCarryOver=true&sc_cid=701f2000001OH7EAAW, . Accessed: 2019-06-25.

[29] Tcpdump manpage. https://www.tcpdump.org/manpages/tcpdump.1.html. Accessed: 2019-04-30.

[30] Vmware vcloud air. https://docs.docker.com/machine/drivers/vm-cloud/. Accessed: 2019-06-25.

[31] Running nested vms. https://communities.vmware.com/docs/DOC-8970. Accessed: 2019-06-25.

[32] Any feature to help block/mitigate arp poisoning attacks? https://communities.vmware.com/thread/471193, . Accessed: 2019-06-25.

[33] Capturing and tracing network packets by using the pktcap-uw utility. https://docs.vmware.com/en/VMware-vSphere/6.0/com.vmware.vsphere.networking.doc/GUID-5CE50870-81A9-457E-BE56-C3FCEEF3D0D5.html, . Accessed: 2019-06-25.

[34] Open vswitch. https://www.openvswitch.org/.

[35] Anatoliy Balyk, Mikolaj Karpinski, Artur Naglik, Gulmira Shangytbayeva, and Ihor Romanets. Using graphic network simulator 3 for ddos attacks simulation. *International Journal of Computing*, 16(4): 219–225, 2017.

[36] Terry Benzel. The science of cyber security experimentation: the deter project. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 137–148. ACM, 2011.

[37] Stefan Boesen, Richard Weiss, James Sullivan, Michael E Locasto, Jens Mache, and Erik Nilsen. Edurange: meeting the pedagogical challenges of student participation in cybertraining environments. In *7th Workshop on Cyber Security Experimentation and Test ({CSET} 14)*, 2014.

[38] Sung-Do Chi, Jong Sou Park, Ki-Chan Jung, and Jang-Se Lee. Network security modeling and cyber attack simulation methodology. In *Australasian Conference on Information Security and Privacy*, pages 320–333. Springer, 2001.

[39] Bernard Ferguson, Anne Tall, and Denise Olsen. National cyber range overview. In *2014 IEEE Military Communications Conference*, pages 123–128. IEEE, 2014.

[40] Mohammed Humayun Kabir, Syful Islam, Md Javed Hossain, and Sazzad Hossain. Detail comparison of network simulators. *International Journal of Scientific & Engineering Research*, 5(10):203–218, 2014.

[41] Jelena Mirkovic and Terry Benzel. Teaching cybersecurity with deterlab. *IEEE Security & Privacy*, 10(1): 73–76, 2012.

[42] Ashwin Palekar. Microsoft azure network security. Nov 2013.

[43] Cuong Pham, Dat Tang, Ken-ichi Chinen, and Razvan Beuran. Cyris: A cyber range instantiation system for facilitating security training. In *Proceedings of the Seventh Symposium on Information and Communication Technology*, pages 251–258. ACM, 2016.

[44] Teri Radichel. Packet capture on aws. In *SANS Institute Reading Room*. SANS Institute, 2017.

[45] virtualbox.org. Virtual networking. `https://www.virtualbox.org/manual/ch06.html`. Accessed: 2019-06-23.

[46] Jan Vykopal, Radek Ošlejšek, Pavel Čeleda, Martin Vizvary, and Daniel Tovarňák. Kypo cyber range: Design and use cases. 2017.

[47] John Wroclawski, Jelena Mirkovic, Ted Faber, and Stephen Schwab. A two-constraint approach to risky cybersecurity experiment management. In *Invited paper at the Sarnoff Symposium*, 2008.