Towards Scalable & Secure Virtual Laboratory for Cybersecurity e-Learning

Dissertation

zur Erlangung des akademischen Grades
des Doktors der Ingenieurwissenschaften (Dr.-Ing.)
am Fachgebiet Internet-Technologien und -Systeme
des Hasso-Plattner-Instituts

eingereicht an der
Hasso-Plattner-Institut, Digital Engineering Fakultät
der Universität Potsdam

vorgelegt von M.Sc.

Johannes Harungguan Sianipar

Potsdam, June 2019

Dissertation Reviewers:
Prof. Dr. Christoph Meinel, Hasso-Plattner-Institut,
Prof. Dr. Benhard Sitohang, Institut Teknologi Bandung,
Prof. Dr. Matthew Adigun, University of Zululand

Examination Committee:
Prof. Dr. Andreas Polze, (Chairman)
Prof. Dr. Felix Naumann,
Prof. Dr. Robert Hirschfeld

Published online on the
Publication Server of the University of Potsdam:
https://doi.org/10.25932/publishup-50279
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-502793

To my parents, my wife Silvia and my son Nathan

Declaration

I herewith declare that I have produced this thesis without the prohibited assistance of third parties and without making use of aids other than those specified. Notions taken over directly or indirectly from other sources have been identified as such. All data and findings in the work have not been falsified or embellished. This thesis has not previously been presented in identical or similar form to any other German or foreign examination board.

The thesis work was conducted from March 2013 to May 2019 under the supervision of Prof. Dr. Christoph Meinel.

Johannes Harungguan Sianipar
Potsdam, Germany

Abstract

Distance Education or e-Learning platform should be able to provide a virtual laboratory to let the participants have hands-on exercise experiences in practicing their skill remotely. Especially in Cybersecurity e-Learning where the participants need to be able to attack or defend the IT System. To have a hands-on exercise, the virtual laboratory environment must be similar to the real operational environment, where an attack or a victim is represented by a node in a virtual laboratory environment. A node is usually represented by a Virtual Machine (VM). Scalability has become a primary issue in the virtual laboratory for cybersecurity e-Learning because a VM needs a significant and fix allocation of resources. Available resources limit the number of simultaneous users. Scalability can be increased by increasing the efficiency of using available resources and by providing more resources. Increasing scalability means increasing the number of simultaneous users.

In this thesis, we propose two approaches to increase the efficiency of using the available resources. The first approach in increasing efficiency is by replacing virtual machines (VMs) with containers whenever it is possible. The second approach is sharing the load with the user-on-premise machine, where the user-on-premise machine represents one of the nodes in a virtual laboratory scenario. We also propose two approaches in providing more resources. One way to provide more resources is by using public cloud services. Another way to provide more resources is by gathering resources from the crowd, which is referred to as Crowd-resourcing Virtual Laboratory (CRVL).

In CRVL, the crowd can contribute their unused resources in the form of a VM, a bare metal system, an account in a public cloud, a private cloud and an isolated group of VMs, but in this thesis, we focus on a VM. The contributor must give the credential of the VM admin or root user to the CRVL system. We propose an architecture and methods to integrate or dis-integrate VMs from the CRVL system automatically. A Team placement algorithm must also be investigated to optimize the usage of resources and at the same time giving the best service to the user. Because the CRVL system does not manage the contributor host machine, the CRVL system must be able to make sure that the VM integration

will not harm their system and that the training material will be stored securely in the contributor sides, so that no one is able to take the training material away without permission. We are investigating ways to handle this kind of threats.

We propose three approaches to strengthen the VM from a malicious host admin. To verify the integrity of a VM before integration to the CRVL system, we propose a remote verification method without using any additional hardware such as the Trusted Platform Module chip. As the owner of the host machine, the host admins could have access to the VM's data via Random Access Memory (RAM) by doing live memory dumping, Spectre and Meltdown attacks. To make it harder for the malicious host admin in getting the sensitive data from RAM, we propose a method that continually moves sensitive data in RAM. We also propose a method to monitor the host machine by installing an agent on it. The agent monitors the hypervisor configurations and the host admin activities.

To evaluate our approaches, we conduct extensive experiments with different settings. The use case in our approach is Tele-Lab, a Virtual Laboratory platform for Cyber Security e-Learning. We use this platform as a basis for designing and developing our approaches. The results show that our approaches are practical and provides enhanced security.

Zusammenfassung

Die Fernunterrichts- oder E-Learning-Plattform sollte ein virtuelles Labor bieten, in dem die Teilnehmer praktische Übungserfahrungen sammeln können, um ihre Fähigkeiten aus der Ferne zu üben. Insbesondere im Bereich Cybersicherheit E-Learning, wo die Teilnehmer in der Lage sein müssen, das IT-System anzugreifen oder zu verteidigen. Um eine praktische Übung durchzuführen, muss die virtuelle Laborumgebung der realen Betriebsumgebung ähnlich sein, in der ein Angriff oder ein Opfer durch einen Knoten in einer virtuellen Laborumgebung repräsentiert wird. Ein Knoten wird normalerweise durch eine virtuelle Maschine (VM) repräsentiert. Die Skalierbarkeit ist zu einem Hauptproblem des virtuellen Labors für E-Learning im Bereich Cybersicherheit geworden, da für eine VM eine erhebliche und feste Zuweisung von Ressourcen erforderlich ist. Die Verfügbare Ressourcen begrenzen die Anzahl der gleichzeitigen Benutzer. Die Skalierbarkeit kann erhöht werden, indem die verfügbaren Ressourcen effizienter genutzt und mehr Ressourcen bereitgestellt werden. Die Erhöhung der Skalierbarkeit bedeutet die Erhöhung der Anzahl gleichzeitiger Benutzer.

In dieser Arbeit schlagen wir zwei Ansätze vor, um die Effizienz der Nutzung der verfügbaren Ressourcen zu erhöhen. Der erste Ansatz zur Erhöhung der Effizienz besteht darin, virtuelle Maschinen (VMs) durch Container zu ersetzen, wann immer dies möglich ist. Der zweite Ansatz besteht darin, die Last auf den Benutzer vor-ort-maschine zu verteilen, wobei der Benutzer vor-ort-maschine einen der Knoten in einem virtuellen Laborszenario repräsentiert. Wir schlagen auch zwei Ansätze vor, um mehr Ressourcen bereitzustellen. Eine Möglichkeit, mehr Ressourcen bereitzustellen, ist die Nutzung von Public Cloud Services. Eine andere Möglichkeit, mehr Ressourcen bereitzustellen, besteht darin, Ressourcen aus der Menge zu sammeln, die als Crowd-Resourcing Virtual Laboratory (CRVL) bezeichnet wird.

In CRVL, kann die Menge ihre ungenutzten Ressourcen in Form einer VM, eines Bare-Metal-Systems, eines Accounts in einer Public Cloud, einer Private Cloud und einer isolierten Gruppe von VMs einbringen, aber in dieser Arbeit konzentrieren wir uns auf eine VM. Der Mitwirkende muss dem CRVL-System den Berechtigungsnachweis des VM-Administrators oder des Root-Benutzers geben.

Wir schlagen eine Architektur und Methoden vor, um VMs automatisch in das CRVL-System zu integrieren oder daraus zu entfernen. Ein Team-Placement-Algorithmus muss ebenfalls untersucht werden, um die Ressourcennutzung zu optimieren und gleichzeitig den besten Service für den Benutzer zu bieten. Da das CRVL-System den Beitragsgeber-Hostcomputer nicht verwaltet, muss das CRVL-System in der Lage sein, sicherzustellen, dass die VM-Integration ihr System nicht beeinträchtigt und das Schulungsmaterial sicher auf den Beitragsgeberseiten aufbewahrt wird, damit niemand das Trainingsmaterial ohne Erlaubnis wegnehmen kann. Wir untersuchen Möglichkeiten, um mit dieser Art von Bedrohungen umzugehen.

Wir schlagen drei Ansätze vor, um die VM von einem bösartigen Host Administrator zu stärken. Um die Integrität einer VM vor der Integration in das CRVL-System zu überprüfen, schlagen wir eine Remote-Verifikationsmethode ohne zusätzliche Hardware wie den Trusted Platform Module-Chip vor. Als Besitzer des Host-Rechners können die Host-Administratoren über Random Access Memory (RAM) auf die Daten der VM zugreifen, indem sie Live Memory Dumping, Spectre- und Meltdown-Angriffe durchführen. Um es dem bösartigen Host-Administrator zu erschweren, die sensiblen Daten aus dem RAM zu erhalten, schlagen wir eine Methode vor, die kontinuierlich sensible Daten im RAM bewegt. Wir schlagen auch eine Methode zur Überwachung des Host-Rechners vor, indem ein Agent darauf installiert wird. Der Agent überwacht die Hypervisor-Konfigurationen und die Aktivitäten des Hostadministrators.

Um unsere Ansätze zu bewerten, führen wir umfangreiche Experimente mit unterschiedlichen Einstellungen durch. Der Anwendungsfall in unserem Ansatz ist Tele-Lab, eine virtuelle Laborplattform für Cybersicherheit E-Learning. Wir nutzen diese Plattform als Grundlage für die Gestaltung und Entwicklung unserer Ansätze. Die Ergebnisse zeigen, dass unsere Ansätze praktisch sind und mehr Sicherheit bieten.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my Ph.D. supervisor, Prof. Dr. Christoph Meinel, for his wisdom, patience, and continues support. I sincerely thank him for providing the opportunity and financial support for me to pursue my PhD in Germany. The thanks are extended to Christian Willems for sharing research ideas in Tele-Lab, for being the co-authors of my papers, and for involving me in teaching the "Linux for the masses" course.

I would like to thank Prof. Andreas Polze for willing to become my second supervisor. I would also like to thank the following HPI staff: Michaela Schmitz, Daniela Roick and Sabine Wagner, especially Michaela for helping me in providing many supporting documents of Visa Extensions and thesis submission. I want to thank Dr Nemeth Sharon for helping me in improving my English and in proofreading one of my papers. I also want to thank Matthias Bauer for allocating a time slot in our research seminar whenever I need it.

My colleagues and co-authors, Dr Eyad Saleh and Muhammad Ihsan Sukmana, thank you for supporting me in writing the papers. I want to acknowledge Nuhad Shaabani for the discussion on a research topic and for encouraging me to submit the paper. My friends and office mates Ihsan and Kennedy, it was a nice time that we spent together. I am thankful for every conversation and discussion that we had. I want to thank the rest of my colleagues in HPI, especially Xiaoyin, Nuhad, Aragats, Harry, Haojin, Mina, Tatiana, for the chat and the time we spent together mostly during lunch.

My gratitude to Prof. Benhard Sitohang and Prof. Matthew Adigun, for willing to become my thesis reviewers. I am grateful to Del Institute of Technology and Del Foundation for their support, especially Bapak Luhut Binsar Pandjaitan for understanding, motivation and his support to my family.

Last but not least, I would like to express deepest gratitude to my lovely mom Pasti Pardede and my sincere wife Silvia for their continuous support and encouragement during my PhD journey.

Contents

List of Figures

List of Tables

LIST OF TABLES

Chapter 1

Introduction

The need for Cybersecurity professionals is still very high [10][23][87][88]. Cybersecurity Ventures predicts that there will be 3.5 million unfilled cyber security positions by 2021 [24]. In October 2018, $(ISC)^2$ report said that the cybersecurity workforce supply and demand gap is more than 2.9 million [47]. At the same time, there were a lot of cyber attacks and data breaches have been reported [23][50][113][133]. According to Cisco[1], Asia-Pacific companies receive 6 cyber threats every minute [17]. Cybersecurity Ventures predicts that cybercrime damages will cost the world around $6 trillion annually by 2021 [23]. To fill the gap, the current and the future generations of Cybersecurity professionals must be trained through a learning platform that could be accessible and affordable by a large number of users.

Online Education or e-learning comes with a lot of number of ways to teach and learn outside of traditional classrooms, by providing access to the learning material from anywhere and anytime via Internet connection. It can include text, audio, video, animations, virtual laboratory environments and live chats with professors. Using e-Learning platform, students could interact with professors regardless their geolocation. This is one benefit that we can get from the rapid development of computer science and communication technology. Some researches have been done to find the best learning method to support the students or learners in online Education or e-Learning [106][100]. E-Learning in Computer

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

science is the most rapid developed, where the learners could also have hands-on exercises to implement or practice their knowledge. Some researches have been done to use the technology to provide more features for e-Learning, such as providing computer laboratory for an e-Learning user to learn programming or computer networking.

Massive Open Online Courses (MOOCs) is a form of Online Education which usually delivered in lecture form to online "classrooms" with an unlimited number of participants. The term "MOOC" was first mention in 2008 [65] and it has grown incredibly fast, especially in 2012 [79]. By now, there are a lot of universities and educational institutions participate in the wave of MOOCs, which attract millions of learner into using MOOC platforms, regardless their ages, nationalities and educational backgrounds. MOOC platform should also be able to provide virtual laboratory environment for the participants to practice their knowledge and skills.

Hands-on exercises are essential in cybersecurity learning and training. Knowing is not enough, the participants need to be able to practice the knowledge in a real environment to have adequate skills in cybersecurity. The exercises are usually in various forms of course laboratory, but they could also in the types of treasure hunt or capture the flag competition. The laboratory settings of those exercises could be simple consists of a VM or very complex consists of multiple server systems. Hands-on exercise scenarios need many resources to provide a virtual laboratory environment to thousands of learners. This dissertation is focused on increasing the scalability of a Virtual Laboratory for cybersecurity e-Learning to be able to serve a large number of users as in the MOOC platform. Increasing the scalability should not reduce the security of data confidentiality.

1.1 Research Background

Hasso Plattner Institute (HPI) had an e-Learning platform called Tele-Lab. It is a platform for learning IT Security (Cybersecurity) subjects. In a learning system, especially in cybersecurity subjects, students should be able to do practical hands-on exercises. Tele-Lab tried to fill the gap between e-Learning and practical education in cybersecurity, by providing Virtual Laboratory Environment. In

cybersecurity exercises, each student needs at least one machine to be able to implement an exercise scenario and check the result. More machines are needed for more complex scenarios such as attack and defense scenario, which requires at least one machine as a victim and the other as an attacker.

Telelab provides a virtual laboratory environment using a private cloud platform called OpenNebula [70]. Each machine (victim or attacker) in practical hands-on exercise is represented by a virtual machine (VM). A VM needs a fixed allocation of resources such as memory, processor (virtual CPU) and hard disk. For example, to deliver a Man in the middle attack exercise scenario for a participant, it needs 3 VMs where one VM as an attacker, and two VMs as the victims [126]. For 100 students, they must be provided with 300 VMs. Suppose one VM needs memory around 512 MB, then 300 VMs need 150 GB memory. To serve this number of simultaneous participants in a private cloud, we need to provide a high specification computer hardware or many computers with the low specification. To serve more participants, the virtual laboratory must be able to scale out. The number of participants is limited by the available resources for virtual laboratory. To be able to use Telelab as a MOOC platform that could serve thousands of users, we need to increase the scalability of Telelab.

1.2 Research Questions

Based on the aforementioned research background, the main question of this thesis is "How To Increase the scalability of Virtual laboratory for Cybersecurity e-Learning such as Telelab?". In answering the research question, a part of the virtual laboratory could be running on an untrustworthy environment which creates a security problem. We investigate a set of questions that construct this dissertation storyline. The questions are:

1. How to increase the scalability by increasing efficiency in using the available resources?
2. How to increase the scalability by providing more resources?
3. How to secure the data confidentiality when the VM is running on untrustworthy host machine?

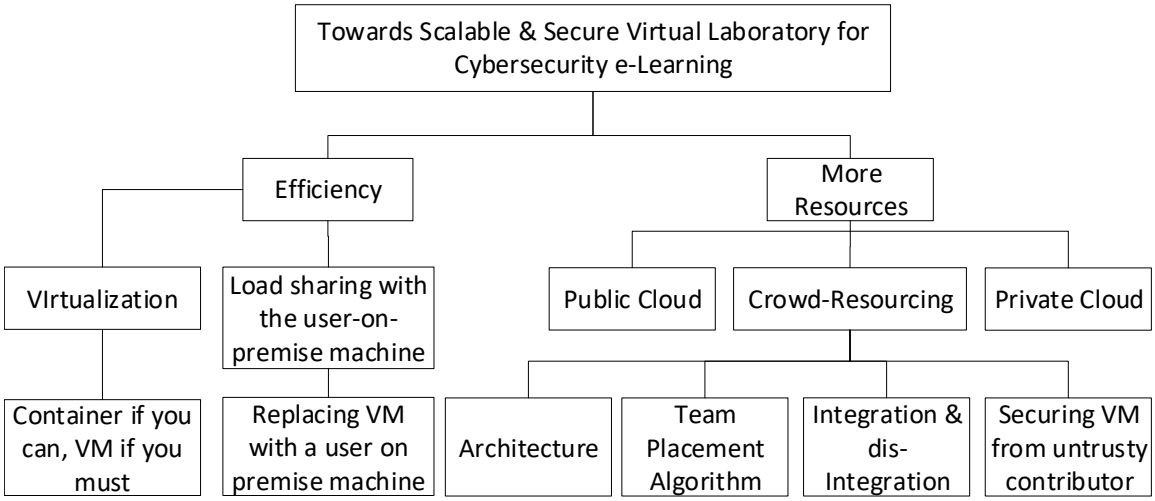

Figure 1.1: Research Structure

In answering the research questions, we follow a research structure shown in Fig. 1.1. To increase the efficiency, we propose to replace VMs with containers whenever it is possible and to replace an attacker VM with a user on-premise machine. To provide more resources, we propose to use public cloud services and Crowd-Resourcing. In crowd-resourcing, the resources are gathered from the crowd (people) which contribute their VMs to the virtual laboratory system. Since the VMs could be anywhere around the world, we propose an architecture, team placement algorithm, integration and dis-integration approaches. We also propose several approaches to strengthen the VMs, since a VM could be running on a un-trustworthy host machine in Crowd-Resourcing virtual laboratory. We use Tele-Lab [70] platform as our study case, but our solutions could also be used to solve the same problem on other platforms.

1.3 Contributions and Publications

The purpose of the approaches developed in this thesis is about how to increase the capability of the Virtual Laboratory system to be able to serve a large number of users to become a part of MOOC platform for Cybersecurity e-Learning. In answering the research questions, we come up with some contributions and

publications. We proposed two approaches as our contributions, to answer the first question of increasing the Virtual Laboratory scalability by utilizing the resources more efficient. The first approach is to replace Virtual Machine (VM) with Container because Container needs much fewer resources compare to VM. This approach is published in the following paper:

- Sianipar, J., Willems, C., Meinel, C.: A container-based virtual laboratory for internet security e-learning. In International Journal of Learning and Teaching. IJLT, vol. 2, no. 2, pp. 121–128. (2016)

The second approach in answering the first research question is by distributing the load to the user on-premise machine. In some exercise scenarios in cybersecurity e-learning, the attacker needs only a web browser to attack the victim. The web browser needs to be running on a VM. Instead of providing a VM for the user, we propose to use the user on-premise machine to run the web browser as the user is the attacker. Because the virtual laboratory is isolated, we need to provide a secure way for the user to attack the victim inside the virtual laboratory from the user on-premise machine. We use signed URL in our approach, which is published in:

- Sianipar, J., Willems, C., Meinel, C.: Signed URL for an Isolated Web Server in a Virtual Laboratory. In Proceedings of the 2017 9th International Conference on Education Technology and Computers, pages 218-222. ACM. (2017)

To answer the second question of providing more resources to increase scalability, we propose to use cloud computing (public or private) and crowd-resourcing. Cloud computing services need an extra budget, while crowd-resourcing does not need an extra budget because the resources are coming from the contributor that voluntarily shared their VMs. There should be some ways to encourage the crowd to contribute. We are more focus on crowd-resourcing, which gathers resources from the people (crowd) because there is still a lack of research in this area. The people as the contributors can contribute resources in the form of a VM. The VM needs to be automatically integrated into the Virtual Laboratory system. To manage the Crowd-Resourcing Virtual Laboratory (CRVL), we propose a CRVL

architecture, team placement algorithm, integration mechanism, and fault recovery mechanism. Publications about CRVL are listed here:

- Sianipar, J., Willems, C., Meinel, C.: Crowdresourcing Virtual Laboratory Architecture On Hybrid Cloud. In INTED2016 Proceedings, 10th International Technology, Education and Development Conference, pages 2940-2949. IATED. (2016)

- Sianipar, J., Willems, C., Meinel, C.: Team Placement in Crowd-Resourcing Virtual Laboratory for IT Security e-Learning. In Proceedings of the 2017 International Conference on Cloud and Big Data Computing, pages 60-66. ACM. (2017)

- Sianipar, J., Willems, C., Meinel, C.: Virtual Machine Integration & Fault Recovery in Crowd-Resourcing Virtual Laboratory. Accepted at the 7th International Conference on Computer and Communications Management (ICCCM 2019)

In the CRVL system, everybody (including a bad guy) could become a contributor. The VM could be running on an untrustworthy host machine. To secure the data confidentiality when the VM is running on the untrustworthy host machine, we propose three approaches as our contributions, i.e. Virtual Machine Integrity Verification, Moving Sensitive Data against live memory dumping, and Agent-based monitoring architecture. These contributions are published in:

- Sianipar, J., Saleh, E., Meinel, C.: Construction of Agent-Based Trust in Cloud Infrastructure. In Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, pages 941-946. IEEE Computer Society. (2014)

- Sianipar, J., Willems, C., Meinel, C.: Virtual Machine Integrity Verification in Crowd-Resourcing Virtual Laboratory. In 2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA), pages 169-176. ACM. (2018)

- Sianipar, J., Sukmana, M., Meinel, C.:Moving Sensitive Data Against Live Memory Dumping, Spectre and Meltdown Attacks. 2018 26th International Conference on Systems Engineering (ICSEng). IEEE. (2018)

There are two other publications during the period of Ph.D. study which not being used in this thesis, i.e.:

- Sianipar, J., Meinel, C.: A verification mechanism for cloud brokerage system. In Proceedings of 2015 Second International Conference on Computer Science, Computer Engineering, and Social Media (CSCESM), pages 143-148. IEEE. (2015)

- Saleh, E., Sianipar, J., Meinel, C.: SecPlace: A Security-Aware Placement Model for Multi-tenant SaaS Environments. In Proceedings of IEEE 11th Intl Conf on Autonomic and Trusted Computing. IEEE ACT. (2014)

1.4 Thesis Organization

The rest of the thesis is organized as follows: In Chapter 2, we introduce a Virtual Laboratory for IT Security e-Learning platform called Tele-Lab. It is at the beginning state before the research to increase the scalability was started. The architecture will be described in this chapter along with the virtualization, isolation, middleware and remote access. We summarize this chapter by explaining the Tele-Lab weaknesses.

Chapter 3 consists of the ways to increase the scalability by efficiency. The available resources must be used efficiently to be able to serve users as much as possible. Efficiency could be reached by replacing VM with a container. The architecture and the evaluation of the container based virtual laboratory are described in a section of this chapter. Another way to increase resources efficiency is to utilize the user on-premise machine. We describe the scenario where the user on-premise machine could be used as the attacker machine of the exercise scenario. The architecture that keeps the exercise environment isolated is explained in this chapter.

Chapter 4 provides a panorama about some works on using the public cloud as a virtual laboratory for IT subjects especially IT Security. The architecture and the middleware are explained in this chapter.

Chapter 5 focuses on Crowd-Resourcing Virtual Laboratory (CRVL). This chapter describes the architecture of CRVL which combines the private cloud. public cloud and resources from the crowd to create virtual laboratory environment. Most of the web-based learning materials are on the private cloud, but the virtual laboratory environment could be on private cloud, public cloud and contributor's resources. Since there are alternatives to run a virtual laboratory exercise, in this chapter we describes a team placement algorithm that could select the best VM based on the geo-location and the load of the VMs. This chapter also explains the mechanism of the VM integration in CRVL which include the VM integrity verification. Since a contributor's VM is not reliable, we describe the VM fault recovery in this chapter.

Chapter 6 presents several ways to increase the security of data confidentiality in Crowd-Resourcing Virtual Laboratory. Because the contributor's host machine could be malicious, the VM must be verified and secured before being integrated into CRVL. In this chapter, we describe three mechanisms that could be used to increase the data confidentiality of a VM. The first mechanism is to remotely verify the integrity of a running VM on the contributor's host machine. The second mechanism is about securing sensitive data from live memory dumping, Spectre and Meltdown. The third mechanism is on how to monitor the host machine by placing an agent inside the host machine.

In Chapter 7, we summarize the main contributions of our work and outline some future research directions. At the end of this thesis, appendices, references, acronyms and acknowledgment can be found.

Chapter 2

Tele-Lab: Virtual Laboratory for IT Security e-Learning

In this chapter, we introduce the Tele-Lab as the base of research in this thesis. Our approaches in the next chapters are designed, implemented and tested based on Tele-Lab. Section 2.1 describes the overview of Tele-Lab. The Tele-Lab evolution or development history is described in section 2.2, starting from using a standalone system until a private cloud system. The architecture and the implementation of Tele-Lab using the private cloud is presented in section 2.3. A summary closes this chapter in section 2.4. In this thesis, IT Security and Cybersecurity are used interchangeably to represent the same meaning, but in this chapter, we use IT Security because Tele-Lab is using IT security term to represent security in information technology (IT) field.

2.1 Tele-Lab Overview

Tele-Lab is an Online Education or an e-Learning platform to learn and practice IT security subjects. It consists of a web-based tutoring system to learn the concept and get the knowledge in the form of text or multimedia. The learning units covered by Tele-Lab include various aspects of cryptography and network security such as encryption, authentication and firewall [126]. Tele-Lab also consists of a training environment to practice and to gain hands-on experiences in a virtual laboratory environment built on VMs. The VM is representing a victim

or an attacker node in an exercise scenario. The VM is running on a private cloud (OpenNebula) platform [70] where the attacker VM is accessible and can be operated via remote desktop. On the private cloud platform, the VM could be reverted to the original state after each usage [70]. The VM needs to be reverted to the original (fresh) state to be used by other students. The student needs a web browser to access the learning material and virtual training in the Tele-Lab system. Figure 2.1. shows the new user interface of Tele-Lab while having an SSH connection to one of the nodes in exercise scenario.

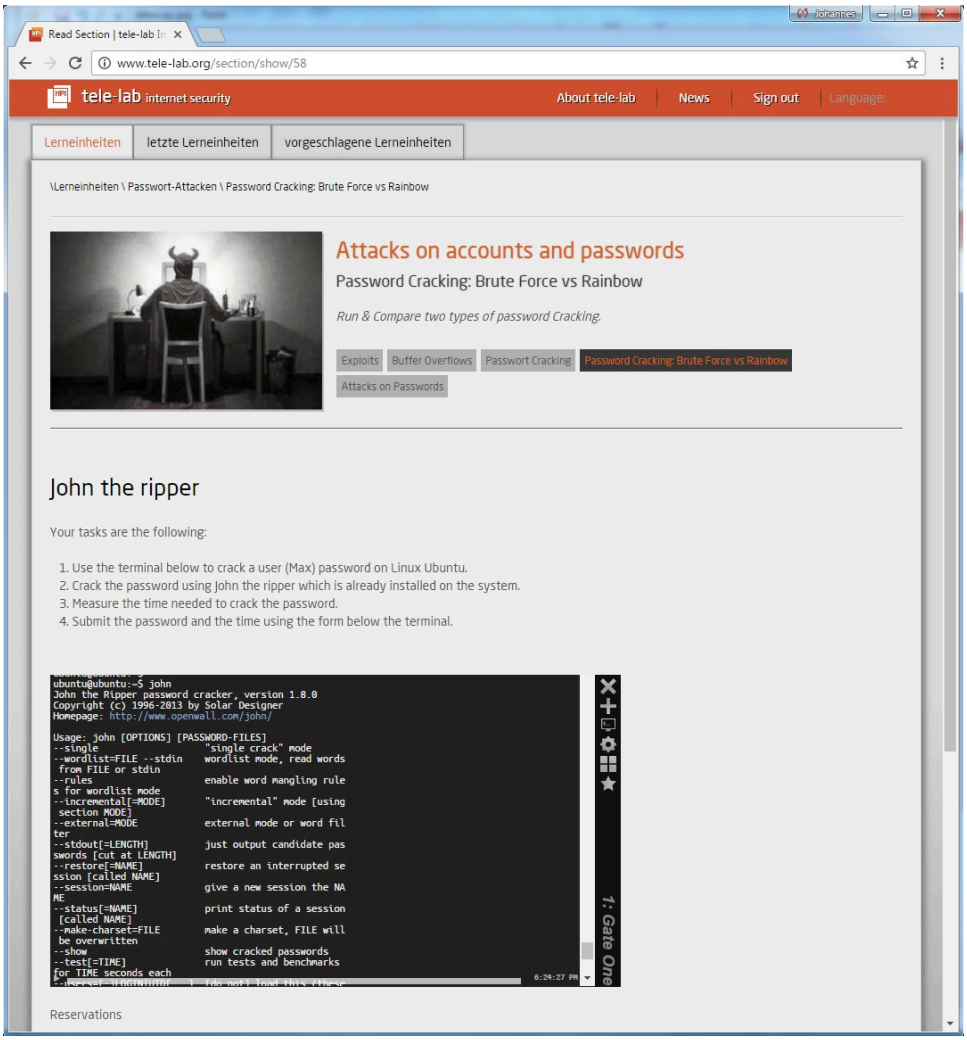

Figure 2.1: Tele-Lab Interface

The Tele-Lab platform could dynamically assign more than one VM to a single

user at the same time. Usually, one student needs at least two VMs, one as the victim and the other as the attacker. A virtual local area network (VLAN) is used to connect and isolate those machines. The combination of those machines and VLAN is known as Team [124][12]. So, a team is a combination of VMs, VLAN, and others that are configured based on an exercise scenario. Within the VLAN, the student as an attacker could perform basic network attacks such as interacting with a victim (e.g. Port scanning). A victim could be a server or a client running all needed services and applications. It could have several scripts that simulate user behaviour. IPTables is used to filter the traffics to and from the VLAN.

2.2 Tele-Lab Evolution

Tele-Lab was started as a tutoring system for IT security education to support the teaching and training process in a laboratory[43]. It was a standalone system where its tutoring system, exercises, and user's working environment were installed and integrated on a single Linux machine. There were two problems of this standalone version. The first was that user errors could easily corrupt the computer machine during exercises. In the event of failure, the exercises are interrupted and terminated, therefore the entire system must be restored using a backup partition. The second problem was that after each usage, the system cannot be easily restored to the original (fresh) state. To get the fresh state, the system needs to undo all the previous user activities on the system.

To handle the problems of the earlier version and provide mobility, Tele-Lab was developed on CD/DVD[41]. All the learning materials and the linux system were embedded on a Knoppix CD/DVD [55]. It is a bootable live CD/DVD which runs a Linux operating system (OS) on a computer without an OS installation on the hard-disk. This way, it is easy to recover from errors or to get the fresh state by only rebooting the system. It is also portable because the students can bring the live CD/DVD with themselves and easily use it on a general computer machine at home or any place they want, as long as the computer has the minimum hardware requirements of the Linux operating system. The weakness of this version is that the live CDs/DVDs need large size of disk space for the operating

system, that makes the space for learning materials and exercises are limited. The live CDs/DVDs is also a standalone system which can be used only by a local user. The tutor needs to visit each computer to check for the user activities or exercise's results on these Tele-Lab versions. Another weakness is that to update the learning materials, the CD/DVD needs to be replaced with the new one.

Placing the Tele-Lab system on a server accessible from the computer network (Internet) could mitigate the weaknesses of the standalone versions. The Tele-Lab server[42][40] is the first e-Learning version of Tele-Lab systems. It transforms the Tele-Lab system from a desktop-based to a web-based system and from standalone to client-server that makes the resources accessible for remote users. The Tele-Lab server could separate the user's working environments from the tutoring system and exercises. The user's working environments are build from VMs created on a host and connected to a network. These VMs and the network can be cloned and running on other hosts. Users could be given privilege rights on the VM to complete security tasks. If a VM was crashed, it could be quickly restored and will not damage the other parts of the Tele-Lab server. Using VMs, a laboratory environment can be economically and conveniently virtualized by software, which is also remotely accessible on the Internet.

Some works had been done to improve the content and the system of the Tele-Lab Server. Cordel et al. added wireless network security learning unit in the Tele-Lab content[19]. Willems et al. in [125], enhanced the Tele-Lab architecture by adding the NX server for proxying the remote desktop connection from the user to each VM. They also added other features such as a virtual machine pool to accelerate the provisioning time[125]. Willems et al. in [123], described several ways to secure the Tele-Lab from several kinds of attack. Willems et al. in [124], introduced a distributed virtual laboratory architecture to be able to share resources. The architecture uses a VPN to connect two Tele-Lab servers. The architecture was implemented to connect two independent instances of Tele-Lab in Germany and Lithuania.

The latest version of Tele-Lab that being used as the base of research in this thesis was introduced by Moritz et al. in [70]. They use OpenNebula private cloud to make Tele-Lab's virtual laboratory environment more scalable with a desire to have IT Security virtual laboratory for MOOCs. Tele-Lab consists of

a frontend and a backend application. The backend application communicates with the OpenNebula via middleware. VMs and virtual networks are provided by OpenNebula. Details about private cloud Tele-Lab is described in the next section.

2.3 Tele-Lab Architecture

The Tele-Lab architecture consists of a frontend web based application, a back-end service, and a VM pool. The frontend application is for tutoring contents (Learning Units) with user and administrator interface. The user interface is for participants (students) to work with the learning units and to access the virtual laboratory. The administrator interface is for administrator to provide content of learning units, create the training scenarios, prepare the VMs, and monitor the system in general. This frontend application was developed on Grails framework which communicates with the back-end service using a XML-RPC client. The Tele-Lab architecture taken from [70] is shown in figure 2.2.

The Tele-Lab Backend service is a middleware to manage the communication between frontend and the cloud controller. The service could start, suspend and stop VMs on OpenNebula. It was developed in ruby and it uses the Ruby OpenNebula Cloud API to communicate with the OpenNebula. This API allows external applications to work with OpenNebula's datatypes, e.g. VM images, virtual networks, and virtual machines. Tele-Lab service implements a XML-RPC server for platform independent access.

OpenNebula is one of open source private cloud frameworks. It has a Host Controller installed on each host machine to manage the VMs running on the physical host. The Host Controller communicates with the hypervisor and with a central cloud component called Cloud Controller to receive or send information. The Cloud Controller collects the host information from each host, and uses the information to determine on which physical host a new VM should be deployed. It is also responsible for virtual network management using virtual network devices such as virtual switching, to connect VMs. To manage VMs, the Host Controller and Cloud Controller use the libvirt library to be independent

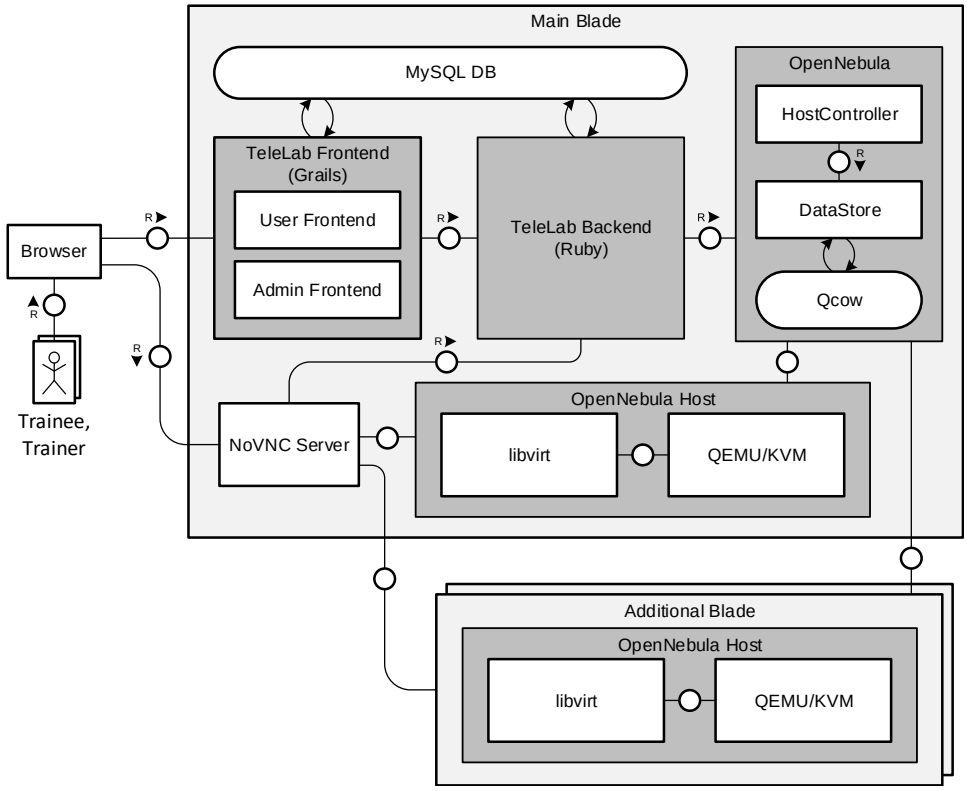

Figure 2.2: Private Cloud Tele-Lab Architecture

of particular hypervisors such as QEMU-KVM[1]. Libvirt is a toolkit to manage virtualization platforms[2].

In cloud computing, a VM disk image is needed to run a VM. A VM disk image is a template for creating new VM instances. All the various VM disk images are stored in a central repository called Datastorage, which is a network directory shared using distributed file system such as Network File System (NFS). In OpenNebula [3], the disk image could be stored in the QCOW2 [4] image format to let multiple VMs using the same image concurrently without copying the image. QCOW2 image format enables a faster creation of similar VMs without the need to create a lot of virtual disk images. When a new VM instance is started, QCOW2 creates a new delta file with differences to the original image without

[1]https://wiki.archlinux.org/index.php/KVM
[2]https://libvirt.org/
[3]https://opennebula.org/
[4]https://people.gnome.org/ markmc/qcow-image-format.html

cloning the image. All common data are provided by the original image which is in a read-only mode. This way, multiple VMs could be started simultaneously without the drawback of copying the original image.

Tele-Lab uses noVNC [1] for remote desktop connection between a user (trainee) and the VMs that should be used by the trainee to do the hands-on exercises. noVNC server is installed on the physical host to provide a remote desktop connection between a user and the VM. noVNC server uses QEMU framebuffer device to get a remote desktop connection to the VMs. On the user side, an HTML5 browser is needed to get a remote desktop connection to one of the VM in the OpenNebula private cloud. No additional software or plugins is required. HTML5 has canvas element and WebSocket features which enable the remote desktop connection to noVNC server in the private cloud.

As shown in figure 2.2, Tele-Lab uses MySQL Database to store not only the frontend information (i.e. user data and learning unit content), but also the backend information such as pools and teams.

2.4 Chapter Summary

In this chapter, we describe Tele-Lab, a virtual laboratory for Cyber security e-Learning which uses a VM to represent a node (host) in an exercise scenario. Tele-Lab uses OpenNebula private cloud to create a Team for a user to do hands-on exercises. A Team consists of VMs and virtual networks. Tele-Lab uses VLAN and iptables to isolate a Team in the private cloud. Tele-Lab uses QCOW2 image format to let multiple VMs use the same image concurrently. The user could use any modern browser to get a remote desktop connection to the VMs without any additional software. noVNC is used for the remote desktop connection.

A VM needs a fix allocation of resources such as memory, vCPU (virtual CPU) and disk space. The available number of resources limits the number of VMs that could be running simultaneously on the private cloud. The number of resources could not be easily increased. Thus, the number of users that could be served concurrently is also limited. This is one of the weaknesses of the Tele-Lab that must be addressed to be able to serve a large number of users or to function as

[1] https://novnc.com/info.html

a MOOCs platform. Tele-Lab needs to be able to rapidly scale-out and scale-in on demand.

Chapter 3

Efficiency to Increase Scalability

In this thesis, we are trying to increase the ability of the virtual laboratory system to scale out to serve a large number of simultaneous users as in a MOOCs platform. Scalability could be increased without providing more resources by increasing the efficiency of the virtual laboratory system, such as by replacing a high resource usage software component with the lower resource usage software component. Virtual Machine (VM) is a high resource usage software component that makes a cybersecurity virtual laboratory uses many resources. In many cases, VMs could be replaced with containers. Efficiency could also be increased by sharing the load to the user on-premise machine, where a node in the exercise scenario is represented by the user on-premise machine. In this chapter, we present two approaches to increase the scalability of cybersecurity virtual laboratory (Tele-Lab) by doing efficiency in using the available resources.

3.1 Container Based Virtual Laboratory

Existing technologies enable us to choose the one that uses the lowest resources but still able to function properly. For example, in the Tele-Lab, a virtual machine is used to represent a host in an exercise scenario. In virtualization technology, besides Virtual Machine, Container is another way to represent a host/node in the context of the training environment. In some cases where the desktop environment is not needed, the VM could be replaced with a container (docker). A container does not need resources as much as a VM, because containers are run-

ning above the same kernel (Operating System), and Containers could only run a text-based application. Running container instance does not install a new operating system, because it uses the same OS as the host. The container does not need a preliminary fixed allocation of resources. It uses namespaces and cgroup to isolate one container from the others and from the host[1]. It is usually used to run services in an isolated environment.

Containers can communicate with each other and act according to the practical exercise scenario. Network tools that are needed in the practical exercise scenario can be installed as needed in each container. A container uses resources just like an application in an operating system. This means that we can have containers as much as the available memory in a host. We can run containers as much as we can as long as the PID (Process Identifier) and resources are available. The number of containers that can be run on a host depends on how much memory is needed for an application inside the container. Besides taking much fewer resources compared to VM, container provision time is much smaller than VM provision time. Felter et al.[30], Seo et al. [98] and Sharma et al.[99] elaborate the performance of a container compares to a VM. In this section, we present the architecture of the container based virtual laboratory.

3.1.1 Architecture and Organization

Moving from a VM based virtual laboratory to a container-based virtual laboratory changes several parts of the Tele-Lab system. Replacing VMs with containers needs to create a new architecture, which is a re-factored enhancement to the infrastructure presented in figure 2.2. Basically, the architecture is almost the same as the private cloud Tele-Lab, except that it uses containers instead of VMs to represent a node in the virtual laboratory scenario. A host of containers is a VM in a private/public Cloud. Additional host (VM) can be provided in the same cloud provider to serve larger participants. Figure 3.1 shows the architecture of Tele-Lab using containers which we call it as Container-Based Virtual Laboratory Architecture.

The Tele-Lab backend needs to be modified to add containers to the architecture. In the new architecture, a node in a virtual training environment could be represented by a VM or Container, depends on the scenario requirement. When a

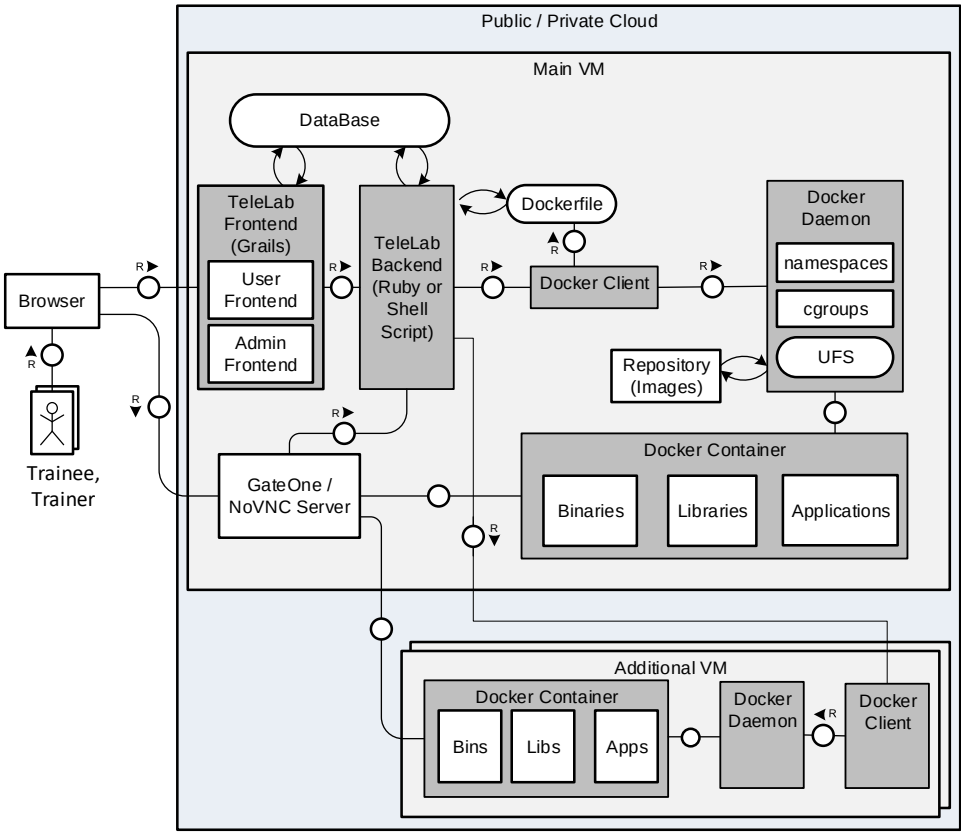

Figure 3.1: Container Based Virtual Laboratory Architecture

scenario requires a Windows or a Linux desktop, a node needs to be represented by a VM. When a scenario does not require a desktop or a GUI application, a node could be represented by a container. In this section, we focus on the container side of the virtual laboratory architecture.

In the container-based virtual laboratory (CBVL), containers are running on a VM to isolate the containers from other containers on other VMs. This way, if a user of a container crashed the host VM, it does not affect the other containers on other VMs. The VM, as the host machine, should not be the main VM where the Tele-Lab application is running. It should be another VM in the cloud. The Tele-Lab Backend still needs to be able to manage VMs in the private cloud platform (OpenNebula) by implementing an XML-RPC server for platform-independent access. It also needs to be able to manage containers using API[1] or shell scripts on

[1]https://docs.docker.com/engine/api/v1.30/

the container platform (docker). It receives requests from the Tele-Lab Frontend to create a Team for a Trainee (student). A Team consists of containers and virtual networks that are configured based on a training scenario. In this thesis, we use shell scripts to manage docker containers.

A Team must be isolated from other networks using VLAN on a Virtual Switch. Each container is provided with several tools which are needed to do the training. Docker is used as a platform to run containers. The Tele-Lab Backend creates a Team by executing shell commands on the host (VM) where Docker Client installed and running. It uses an SSH connection to execute shell commands on a remote host (VM). The shell commands for creating the Team is listed in a shell script. Which scripts to run is depending on the exercise scenario of a learning unit. This information is provided in the database.

Basically, the script consists of docker commands to create containers based on specific images. The images are already built and ready to be used. The script also consists of openVswicth command to create a virtual network for the containers, specify an IP address for each container, and to configure VLAN for the Team isolation. The script has IPTables commands to filter traffic to and from VLAN of the Team. The script also has IPTables command for network address translation (NAT) between VM (host) IP and port address and the container IP and port address. Listing 3.1. shows an example of a shell script.

Docker needs a container image to start a container instance. Docker builds images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands that would be executed to build a Docker image. By calling docker build command from a terminal, Docker builds an image step by step, executing the instructions successively[2]. Dockerfile is created by the Trainer or Admin for each container image, based on the container roles in the Team. Trainer or Admin builds container image by executing a shell command on Docker Client, via TeleLab Backend. The image could be stored locally or on a remote repository.

A container can be run with specific capabilities or even as a privileged container. Capabilities of a container can be specified as needed[1]. Docker Daemon runs and manages a container. Each container is separated by using namespaces, cgroups and UFS[1]. Docker container consists of binaries and libraries that can

Listing 3.1: Shell Script Example.

```
#!/bin/bash
#Shell command arguments:
#Number of scenario instances = $0
#bridge number = $1
#port number = $2

sudo ovs-vsctl add-br $1
for (( i=1; i<=$0; i++))
port = $2 + 1
do
  j=0
  for (( s=1; s<=2;s++))
  do
    let j=j+1
    if [ "${j}" -eq 1 ]
    then
        c[$j]=$(sudo docker run --privileged=true -d -n=false -t -i mitm /bin/bash)
    else
        c[$j]=$(sudo docker run -d -n=false -t -i ftp_client /bin/bash)
    fi
    sudo ./ovs.sh $br_num ${c[$j]} 172.21.$i.$s/24 172.21.$i.255 172.21.$i.254 $i
  done
  sudo ovs-vsctl add-port $br_num "vlan$i" -- set interface "vlan$i" type=internal
  sudo ovs-vsctl set port "vlan$i" tag=$i
  sudo ifconfig vlan$i 172.21.$i.254/24 netmask 255.255.255.0 broadcast 172.21.$i.255
  sudo iptables -t nat -A POSTROUTING -s "172.21.$i.0/24" -o eth0 -j MASQUERADE
  sudo iptables -I FORWARD 1 -i eth0 -d 192.168.56.33/32 -j ACCEPT
  sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport $port -j DNAT
  --to-destination 172.21.$i.1:22
done
```

be used to run an application. The Docker client is the primary user interface to Docker. It accepts commands from the user and communicates back and forth with a Docker daemon[1].

When the needed containers were ready to be used, the CBVL opens a GateOne page which provides an SSH connection to the attacker node in the exercise scenario. To be able to access the container of an exercise node, every

attacker container is provided with an SSH server. A participant could access the virtual host using GateOne and SSH connection to the container. GateOne is text-based remote access. If a desktop based is needed, then the CBVL will show a noVNC connection to the participant. In the context of man in the middle (MITM) attack and Firewall learning unit, text-based remote access is used. A small part of Tele-Lab frontend must be modified to be able to display the GateOne [1] interface on the user browser. Through the GateOne interface, users could have an SSH connection to interact with one of the containers in the virtual training environment. Virtual Training Environment is the place where Teams are created and running.

A Team is created on the fly per user request, and it will be shut down when the user leaves or ends the remote access session. Figure 3.2 shows the message flows in the process of creating a Team. A user clicks a link on a browser to start a Tele-Lab hands-on exercise for a Learning Unit. The Tele-Lab Front End accepts the request. It verifies the request, and if the request is valid, the request will be sent to the Tele-Lab Backend. Tele-Lab Backend checks on the database, whether the images for the requested learning unit has been made or not. If the images are not available, the Tele-Lab backend will choose and run the specific Dockerfile to create images for this specific learning unit. After the images are ready, Tele-Lab backend will execute a script on the Docker Client to create a container with specific capabilities. Configuration setting such as capabilities, Dockerfile, for every learning unit is stored in the database.

Commands or script in Docker Client is sent to Docker Daemon to be executed. Docker Daemon could be in the same VM with Docker Client or in another VM in another Public/Private Cloud. Docker Daemon creates containers based on the script. For example, in the MITM Attack exercise scenario, Docker Daemon creates two containers using script. One container has a role as a victim, which periodically sends credential to an FTP server. The other container is the attacker container where a participant has access to it, and from this container, an attacker can run an attack using tools that have been provided in it. The attacker tries to get the victim's credential and use the credential to access the server and get a specific file from the server to prove that he has successfully made the attack.

[1] http://liftoff.github.io/GateOne/

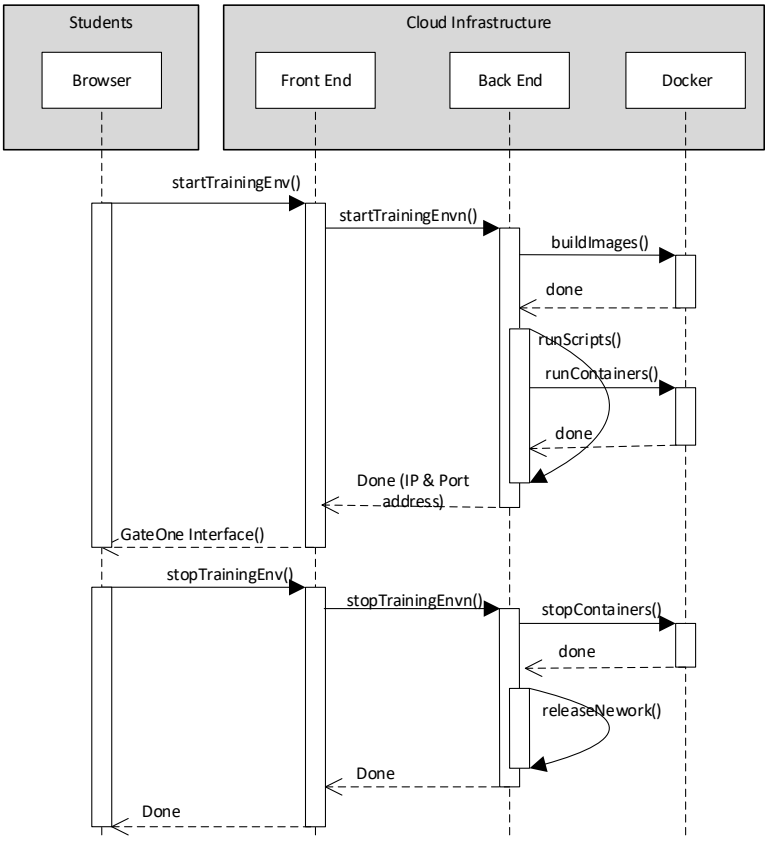

Figure 3.2: Starting and Stopping a Team

After the container was created, run and ready to be used, the CBVL provides remote access to the participant. Remote access is using GateOne to have an SSH connection to the container. CBVL gives a link to the user, whereby clicking this link, the system directs the user to the GateOne with ssh connection to the attacker container where the user can start an exercise. An Example of the link to GateOne SSH connection is https://192.168.56.205/?ssh=ssh://johannes@localhost.

The allocated container instances will be stopped when the user no longer needs it (when the user logs out, or exit from GateOne). When the user logs out from the container instances, the system will stop all the instances related to the user exercise. This way, the allocated memory and CPU for these container instances will be released and ready to be used by others.

3.1.2 Evaluation

We did several experiments to evaluate the container based virtual laboratory. The objectives of our experiments are (1) to prove that the Docker container can be used as a virtual host in a virtual laboratory (Tele-Lab), (2) to find out whether it can increase the scalability or not. We use MITM attack and Firewall learning unit in our experiment, to represent learning units in Tele-Lab. In our experiments, we did not test it using real-life user access. We generated the traffic and ran the tools using scripts and cron job. We do not need to implement all Tele-Lab system, but only the part where containers are used to replace VMs.

For the objective number one, we installed and configured Ettercap for the MITM attack learning unit and IPTables for a firewall learning unit on a docker container. These tools can be run well in the docker container. The MITM attack network topology is shown in figure 3.3. In the CBVL, the nodes (Alice, FTP Server, and Attacker) are represented by containers and the switch is represented by a virtual switch. For objective number two, we run the Team as many as possible on the certain host to find out how many students can simultaneously do the exercise. We create a script and use the cron job to simulate real user activities in doing the exercise.

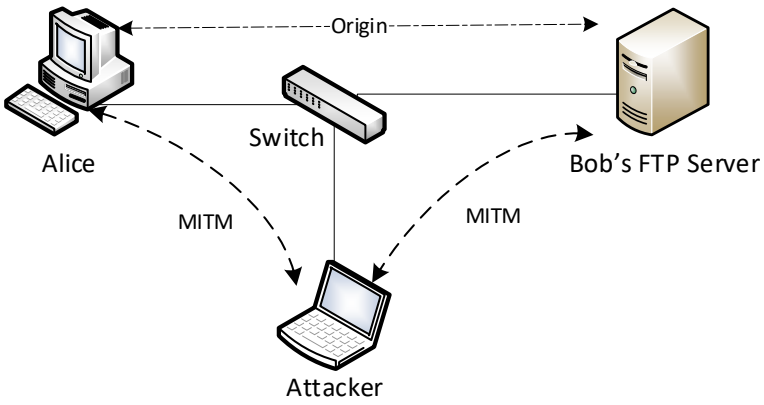

Figure 3.3: MITM Attack Topology

We delivered our experiment on a VM. This VM was running in a Virtual Box hypervisor with resource allocation of 1.5 GBytes memory, 25 GBytes storage and 1 CPU. We use htop to monitor resource consumption. As we can see in

figure 3.3, there are 3 nodes involved in the MITM attack exercise scenario. In our experiment, we use only one FTP server (Bob) for every Team. For Alice container, we create a Dockerfile (Listing 3.2) to create a Docker image that has an FTP client inside. Using crontab script, Alice periodically (every 5 minutes) sends credentials to Bob. For the attacker container, we create a Docker image (Listing 3.3) that has an Ettercap and SSH server installed and configured. The Ettercap commands are executed using the cron job right after the container is started. The Ettercap command is stored in a shell script where we add a random sleep time to execute the command randomly.

Listing 3.2: Dockerfile for Victim Container.

```
1 FROM ubuntu:12.04
2 RUN apt-get update && apt-get install ftp -y
3 RUN apt-get install cron -y
4 ADD ftp_login /var/local/ftp_login
5 ADD crons.conf /var/local/crons.conf
6 RUN crontab /var/local/crons.conf
7 ADD startup.sh /var/local/startup.sh
8 CMD ["/bin/bash", "/var/local/startup.sh"]
```

Listing 3.3: Dockerfile for Attacker Container.

```
1 FROM ubuntu:14.04.2
2 RUN apt-get update && apt-get install ettercap-text-only -y
3 RUN apt-get install cron -y
4 ADD ettercap_start /var/local/ettercap_start
5 ADD crons.conf /var/local/crons.conf
6 RUN crontab /var/local/crons.conf
7 ADD startup.sh /var/local/startup.sh
8 CMD ["/var/local/startup.sh"]
```

To create a MITM attack Team for a student, only Alice and attacker containers that need to be created. We started the experiment by creating 20 Teams for 20 students, and gradually increased by 20 if the performance was still good. The performance is measured by accessing one of the attacker containers and enter attacking shell commands manually and get the response time. More containers

mean higher response time, but as long as the user still feels comfortable and the containers are not crashed, we tried to add more Teams. The performance is still good if there was no error shown up from the system, and we could do the exercise comfortably.

Until 100 Teams for 100 students, the performance was still subjectively acceptable. However, for 120 Teams, the performance is not acceptable. The Linux system showed several errors of killing processes because of "out of memory". Htop showed that the CPU and memory were overloaded. In this condition, we cannot create a new training environment. From this result, we concluded that for the VM of 1.5 GBytes memory, 100 students could be served to do MITM attack exercise. Even though it is subjectively acceptable to run 100 Teams, we suggest to run only 50 Teams (100 containers) on a VM with 1.5 GB memory, to make sure the performance is acceptable. If we used VMs instead of containers, the available resource is enough for 1 Team only. So, running 50 Teams on the same resources is already improved the efficiency of the resources usage significantly.

For Firewall learning unit, we only need two containers for a Team. For the firewall container, we create a Docker image that has IPTables, an SSH server, and an FTP server. IPTables has already configured to allow only SSH connection. For the other container, we create a Docker image that has an ssh client, an FTP client and a Telnet client. These clients are used to test the IPTables configuration. We start the experiment in the same way as in the MITM attack. Until 200 Teams, the performance was still subjectively acceptable without any errors. We stop the experiment because we thought it was enough for the experiment objectives. We suggest to run only 100 Teams on a VM with 1.5 B memory, to ensure that the performance is still good. Figure 3.4 shows the number of teams could be run when using VM and Containers in MITM attack and Firewall scenarios.

From these experiments, we can see that the memory requirement to run a Team plays the main role in the number of Teams that can be run on a virtual training environment. Ettercap needs much more memory compares to IPTables. That is why we could run 200 IPTables Teams, while we could only run 100 Ettercap Teams on the same training environment. To design and allocate resources

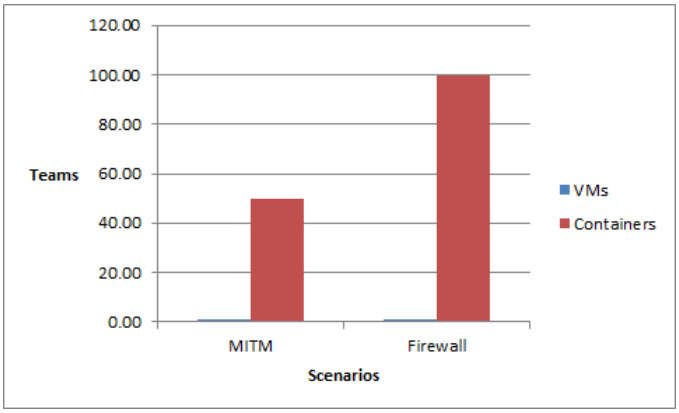

Figure 3.4: Number of Teams

for all Learning Units in Tele-lab, we need to find out the number of Teams that can be run for each Learning unit exercise in the same host.

In the exercise using Ettercap, the CPU is always used 100%, even when there were only 20 training environments. This condition did not generate an error, and the training environments were still working well. The 25 GByte storage capacity was also not an issue in the experiments. The same host can only be used for one Team (one student) if we used VM as a node in the training environment. So, from these experimental results, we can see that containers can be used to replace VMs and can significantly increase the scalability.

Container has some drawbacks in the context of security that the container is more prone to attacks. For example fork bomb could be executed by a user inside the container to shutdown services on the VM. This attack is a DOS attack that could be overcome by limiting the number of processes that could be executed by a user in container and the host machine.

3.2 Load Sharing with User on Premise Machine

As mentioned above, scalability could be increased by reducing the usage of the resources by distributing the load to the user on-premise machine. In the Tele-Lab, all the nodes (VMs) for an exercise scenario is placed in the virtual laboratory server, even though it is used only to run an Internet browser to attack a victim. This kind of nodes could be replaced with the user on-premise machine,

while still keeping the rest of the virtual training environment isolated. As shown in figure 3.5, the user on-premise machine replaces an attacking machine (VM attacker) which using a web browser to do the attack. This attacking machine was a VM because a desktop VM is needed to run a graphical user interface (GUI) application such as browsers. In this research, we focus on a web-based application attack that uses an Internet browser as a tool to attack the victim. We propose an approach that can provide secure access to the web server inside an isolated virtual laboratory, using a signed uniform resource locator (URL) and a reverse proxy. We develop a reverse proxy that able to verify a signed URL, run a shell script and forward the web request to the designated isolated web server.

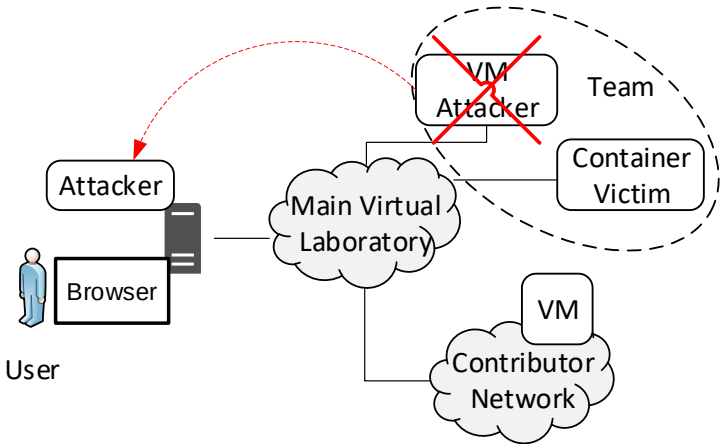

Figure 3.5: Replacing an Attacker VM with User on Premise Machine

3.2.1 Related Work

To replace a node of a Team with user on-premise machine, Signed URL and reverse proxy are used to keep the Team isolated. In this subsection, we describe several related works about Signed URL and reverse proxy. Signed URL is a query string authentication that has been used by most cloud storage providers, as part of access control mechanism. It is referred to as temporary URL in OpenStack[78], Signed URL in Google Cloud Storage[32], preSigned URL in Amazon Web Services[7], and SAS in Microsoft Azure[66].

Graupner et al.[33] proposed to use signed URL in multi-cloud providers architecture. They propose a multi-cloud access control broker scheme suitable for an enterprise use-case. They use Signed URL to share permissions to access a Cloud Storage in multi-cloud service providers. Another use case of a signed URL is proposed by Saleh et al.[92]. They propose SignedQuery, a mechanism designed to enhance the security isolation between tenants in a SaaS environment. SignedQuery uses a signature to sign a tenant's request to prevent any tenant from accessing other tenants' data. Using the signature, the server can recognize the requesting tenant and ensure that the data to be accessed belongs to this tenant.

A reverse proxy is a type of proxy server that retrieves resources on behalf of a client from one or more servers. These resources are then returned to the client as if they originated from the Web server itself. Typical usage of a reverse proxy is to provide Internet users access to a server that is behind a firewall[9]. A reverse proxy can be used as an application firewall against some threats. Wurzinger et al.[127] propose an approach to mitigate cross-site scripting (XSS) attack using a reverse proxy. Lin et al.[61] propose a detection method based on reverse proxy servers to detect the flooding attacks. Valeur et al.[121] propose an approach that composes a web-based anomaly detection system with a reverse proxy to cope with a vulnerable code being deployed and made available to the whole Internet.

3.2.2 Architecture

We aim to replace an attacker VM with a user on-premise machine, while still keeping the virtual laboratory environment isolated. As an attacking machine, a user on-premise machine must be able to attack a victim inside the isolated virtual laboratory environment. In this case, the victim is a web server. To keep the web server isolated, a reverse proxy is used as an intermediary between an attacking machine and the web server. The web server could be installed on a VM or a container inside a VM. The VM could be running on a different location than the Tele-Lab server, such as on another private cloud or a public cloud. The reverse Proxy is installed on the VM where a Team is created. This way, a user on-premise machine could have a direct connection to a Team. It does not have

to go through the main virtual laboratory server to get connected to a Team. Figure 3.6 shows the connection scheme.

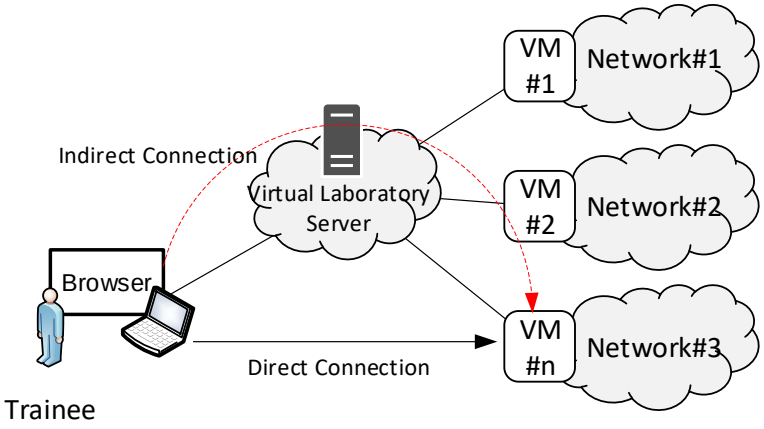

Figure 3.6: User on-Premise Machine Connection to the Virtual Laboratory

We use Signed URL to have a direct connection between user on-premise machine and the Team inside the virtual training environment. Signed URL is created by the virtual laboratory system, used by the user to access the web server, and verified by the reverse proxy server. Figure 3.7 shows the software architecture inside the VM of a virtual training environment. In this case, the web server is installed on a container. The reverse proxy and the containers are running on the same VM. Beside a reverse proxy, a signed URL and a Team creation function, the VM also has a Gateone service. The Gateone is used as an SSH gateway to containers inside the VM.

Signed URL is created by concatenating selected elements (URL path) of the HTTP request to form a string, and use a secret key to create the HMAC (Hash-based message authentication code) of that string. The virtual laboratory system generates the secret key and sends it to a particular VM. This HMAC is the signature used to authenticate the request. It is added to the HTTP request as a parameter of the header section.

The signed URL is sent to the user on-premise machine. The user uses the signed URL to get access to an isolated web server inside a VM. The reverse proxy receives the signed URL and verifies the signature. As shown in figure 3.8, the reverse proxy regenerates the signature using the URL path and the secret

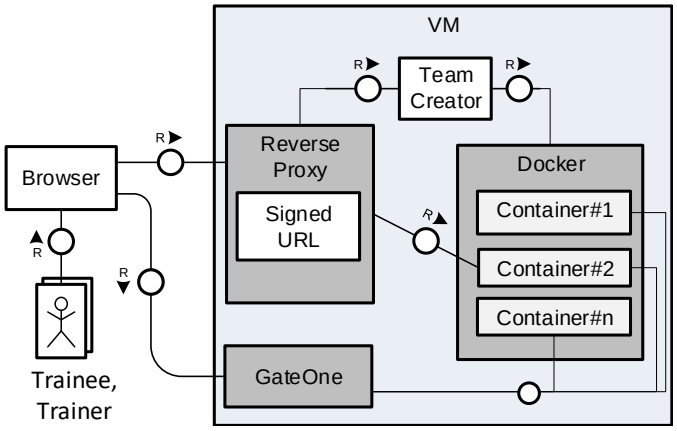

Figure 3.7: Software Architecture on a VM

key. If the signature that was generated in the VM was the same as the one received from the user, the request is valid. If the signatures were not the same, the request was not valid and will be rejected.

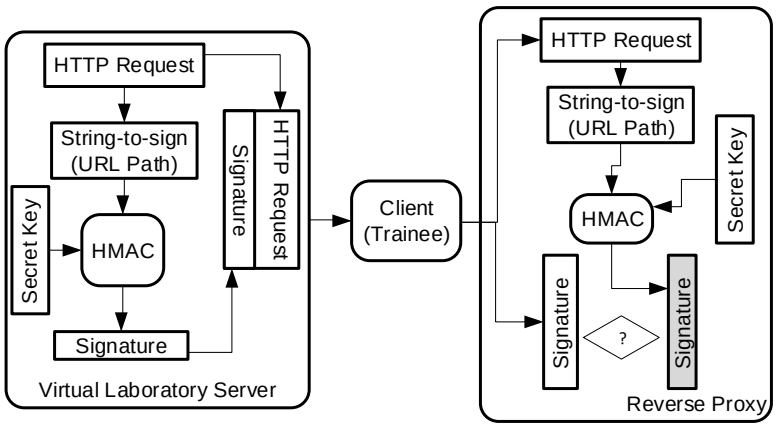

Figure 3.8: Signature Creation and Verification

In Tele-Lab a user needs to login to be able to use the services. After the user login, the user can access a learning unit and request for a practical hands-on exercise. The Virtual Laboratory system needs to create a Team for a practical hands-on exercise. It selects the best VM to run a Team for the user. After selecting the best VM, the virtual laboratory system creates a signed URL. The signed URL contains information about user ID, a learning unit ID, timestamp, and signature. For example: http://192.168.56.205:8080/?userid=10&lu=5&time=

90&signature=edf064dbe83e201baf4b550d2e2d9975d163c2c0. This signed URL is sent to the user. The sequential diagram of this access flow is shown in figure 3.9.

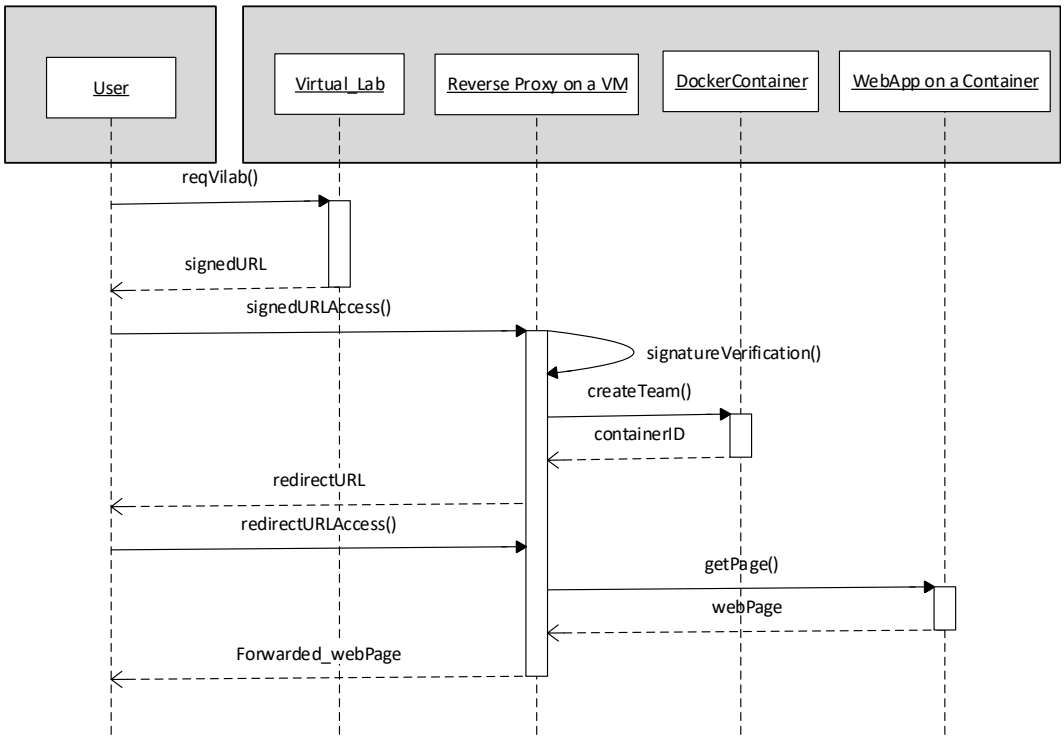

Figure 3.9: Signed URL Sequential Diagram

The user uses the signed URL to request for a Team inside the VM. The reverse proxy on the VM serves the user request if the user was already authenticated and the request was allowed by the virtual laboratory system. The reverse proxy verifies whether the request is valid by checking whether the main virtual laboratory system signs the request or not. If the signed URL is valid, the reverse proxy starts to create the Team as requested by running a script. The reverse proxy gets the container ID and creates a new URL to access the web server inside the container. This new URL is sent to the client, to be used by the client to access the web server. The reverse proxy forwards the user request to the appropriate web server. The URL can only be used in a specific time range.

3.2.3 Implementation

Signed URL is created in the virtual laboratory system, which is in the Front-end, and it is verified in the contributing VM. The implementation is based on the existing Tele-Lab implementation, where the front-end is using Grails, and the back-end is using python and ruby. The signed URL function in the virtual laboratory system is built using groovy because the front end of the virtual laboratory application is using Grails. The front-end creates the Signed URL and send it to the user. The Signed URL consists of user id, learning unit ID, timestamp and signature. The user uses the Signed URL to request for a Team and to get access to the isolated web server, which is part of the Team.

The reverse proxy in a VM receives the request from the user. The reverse proxy server is built using Node.js. We add a function to verify the Signed URL. This function is written using JavaScript. If the Signed URL was not valid, the request is rejected. If the Signed URL was valid, the Team Creator function is called. The Team Creator function is built using a shell script to create a Docker container and an isolated network. The container consists of a web server and an SSH server. After the container is running, the reverse proxy application automatically notices the new container existence, and add the container to the list of signatures and the forwarded URLs. The reverse proxy application creates a new URL to access the new web server. This URL is sent to the user, and it is valid only for three hours. After three hours, the container will be shut down, and the URL will be removed from the list. Behind the scenes, the VM sends reports to the main application about the number of running containers, the list of active users, the log of users activities, etc.

3.2.4 Discussion

Replacing an attacking machine with the user on-premise machine means reducing the resources used on the virtual laboratory system. The amount of the resources that had been reduced is as big as some resources that are needed to run a VM. A Desktop VM usually needs at least 512 MB memory, 20 GB hard disk and one virtual CPU. Besides reducing the number of memory, hard disk and CPU, it also reduces the number of bandwidth usage. When an attacking machine is a VM

inside the virtual laboratory environment, the user accesses the VM using noVNC connection which transfers a VM desktop to a user's browser. This connection needs more bandwidth than the HTTP connection via a reverse proxy because besides forwarding the web page content from a web server, it also needs to send the desktop to the user.

Signed URL is used to keep the web server isolated. The signed URL should be known only to the requesting user. In our approach, we did not verify the ID of the sender of the signed URL, because we assume that the user will not share the signed URL to another user. Although the signed URL can be used multiple times within time range, a user can only have one connection to a VM. If a signed URL is being used by a user, the system rejects another request using the same signed URL. If the signed URL was valid, the reverse proxy creates a Team and gets the URL of a web server which is part of the team. This URL is sent to the user and could be shared with another user. The other user can use this URL to access the web server. Again, we assume that the user does not want to share the URL to another user, because there is no need to do that.

If there is a need to be able to stop another user in using the same URL, we can use cookie-based authentication. The next connection must be able to show the same cookie to get connected to the web server. However, the valid user can also share his cookies to another user to let another user get connected to the web server. As long as the valid user is willing to share his cookies or token to another user, another user can get access to the web server. We could solve this problem by using a javascript function to get the IP Address of the user machine and by using browser fingerprinting.

3.3 Chapter Summary

In this chapter, two different approaches are proposed to increase Tele-Lab scalability by increasing the efficiency of using the resources, i.e. by replacing VMs with containers and by replacing Attacker VM with user on-premise machine. At section 3.1, the first approach is elaborated. It shows that the container could be used to replace VM as a host in Tele-Lab. It is also shown that scalability can be increased. The container-based Tele-Lab system has been successfully

implemented. The future work on this topic will be the extensive evaluation of the container-based Tele-Lab system with real-life user access, to measure the right number of containers on a VM, which is still acceptable by the users. Users satisfaction against the text-based tools should also be measured.

Another future work would be related to the security of the system because we have to use the privileged container. We need to analyze deeply the risk of using a privileged container for a node in a Team and find the best way to secure it. There is a possibility that the user on the privileged container can tamper the host. Providing agent on the host can detect when the privileged container is tampering the host. The agent must monitor every access or any strange activities from the containers to the host. This agent informs the sysadmin whenever it detected an attempt to attack. It can also be used as a measurement of the security of the cloud provider.

At section 3.2, the second approach is described. The paper at hand proposes an approach to increase the scalability of a virtual laboratory for cybersecurity e-Learning, by replacing an attacking machine (a VM in the virtual laboratory environment) with a user on-premise machine. In this research, we focus on an attacking machine that uses an Internet Browser as an attacking tool. This way, the scalability is increased by decreasing the usage of resources. In our approach, a reverse proxy is used as an intermediary between user on-premise machine and the isolated web server. Signed URL is used to validate the user request to a VM where a Team is created. The request is signed by the virtual laboratory system and verified by the reverse proxy on the VM. This scheme is successfully implemented.

For the future work, we are going to investigate on how to replace an attacking machine that uses other desktop application tool to attack the victim. We think that distributing some parts of the load to the client site is a good idea to increase the scalability. We cannot move all things to the client, because of some reasons such as the virtual laboratory system still needs to monitor the learning process.

Chapter 4

Cybersecurity Virtual Laboratory in Public Cloud

A flexible and economical way to provide more resources is using public cloud, where resources can be provided dynamically as needed without downtime. This also applies to Virtual Laboratory (Tele-Lab) where its scalability and flexibility could be increased by using a public cloud provider to get more resources whenever the customer needs it. Of course, this is not without limitation, as for the customer, the most influencing parameter that limits the use of resources is the ability to pay for the cost.

The public cloud virtual laboratory architecture is relatively similar to the private cloud Tele-Lab, where there is a middleware using API to manage VMs on the public cloud provider. The middleware is at the backend part of the Tele-Lab application. The public cloud could be used to run a Team consists of VMs or Containers. The containers team runs on a VM. For the VMs team, the isolation must be configured by implementing Virtual Private Cloud (VPC), and the public cloud provider must be informed to get permission to execute hacking tool software. In this thesis, we focus on the containers team running on a VM in a public cloud.

In using a public cloud to get more resources for a virtual laboratory, we propose only the architecture (section 4.2) which is part of crowd-resourcing virtual laboratory (CRVL) where the resources could be taken from a private cloud, a public cloud and the crowd. In section 4.1 (related work), we describe some vir-

tual laboratories running on a public cloud. In section 4.3, we also describe some Application Program Interfaces (APIs) that could be used to communicate with a public cloud such as Amazon EC2 and Google Cloud.

4.1 Related Work

Cybersecurity or IT security training platform could be built on physical hardware such as the cybersecurity platform used by collegiate cyber defense competition or CCDC [120]. It is expensive and low scalability. Virtualization technologies offer a lower cost and higher scalability for the cybersecurity training environment. Virtual machine (VM) based cybersecurity training environments become popular such as Tele-Lab [70], SEED Labs [28], V-NetLab [109], Platoon [60], Peng Li et al. [58], John Hill et al. [38] and ISERink [48]. These systems have accelerated the spread of cybersecurity training usage, because of their significantly reduced costs for setting up the environment and creating scenarios. However, they could not serve a large number of users, because the training environment deployment is limited to the number of available resources while a VM needs a fix allocation of resources.

The rapid development of cloud computing brings cybersecurity trainings into clouds, because the cloud offers unlimited resources, pay per use and elasticity. Some cloud-based cybersecurity training platforms and systems have been developed such as Virtual Cyber Security Lab on Clouds [117], Cloudwhip [8], V-Lab [129], and EZSetup [59]. These systems are able to serve a large number of users as long as the resources could be provided by the public cloud provider.

Virtual Cyber Security Lab on Clouds [117] works closely with Amazon UK and takes advantage of Amazon's world-leading cloud infrastructure, Amazon Web Services, and security solutions. They design a Cloud-based Cyber Security lab built on Amazon Clouds to customize, configure, monitor and manage virtual resources in a simulated Cyber Security lab environment. Another training platform running on Amazon Web Services is Cloudwhip [8]. Cloudwhip puts the virtual laboratory in public cloud (Amazon) to make the virtual laboratory more scalable and to remove downtime during scaling hardware resources. It is developed to be accessible to even those people new to IaaS. It uses Virtual

Private Cloud (VPC) [5] to isolate the training environments. They claim that they have successfully implemented various network security training in the cloud and the results suggest that cybersecurity training in cloud computing does not only saves costs, but also relieves the educational institutions of the burden of handling and maintaining complex IT Infrastructure.

V-Lab [129] is a cloud-based virtual laboratory education platform that provides a training environment for hands-on experiments using virtualization technologies (such as Xen or KVM Cloud Platform) and OpenFlow switches. The students can access and remotely control the virtual machines (VMs) through OpenVPN connection, to perform the experimental tasks. The V-Lab platform uses an interactive Web GUI for resource management. V-Lab provides a progressive learning path with a series of experiments for network security education.

A Web application called EZSetup [59] is able to create a variety of custom cybersecurity practices (e.g., labs and competition scenarios) in one or more computing clouds (e.g., OpenStack and Amazon AWS). It does not depend on a specific type of cloud platform or technology. It can interact with many cloud platforms and create many virtual environments for security practices simultaneously. At the backend side, EZSetup uses a cloud API to instantiate the training scenario in one particular cloud. It has several APIs to interact with several cloud providers. At the frontend side, EZSetup provides a Web user interface that separate training designer from training participants. Using the EZSetup frontend, namely admin panel, the training designers can visually create a training scenario to hide the complexity in creating training environments.

Some Cybersecurity training platforms are not transparent whether they are using public cloud services on their backend such as the training system at the Center for Systems Security and Information Assurance (CSSIA) [22], NICE Challenge Project [75], Cyber Security Training Using NETLAB+ [76], Cybrary Virtual Security Labs [25], Virtual Hacking Labs [122], Hera Lab [29].

4.2 Architecture

In a Virtual Laboratory system, the amount of needed resources can be different from time to time. The additional resources must be provided in a short time so

that the users do not need to wait for a long time. Public cloud offers the flexibility to add or reduce resources rapidly based on the user's needs. Based on Tele-Lab architecture, we propose a Virtual Laboratory Hybrid Cloud architecture that is running on a private cloud and public cloud. A public cloud is used to provide more resources to run training environments for users. A private cloud is used for the main application and also to provide training environments when the resources are still available. We prefer to put the main application in a private cloud to reduce the cost while we already have a private cloud system. As described in [70], Tele-Lab uses OpenNebula for the private cloud platform.

Public cloud has a host controller to manage, start and stop VMs in the cloud. It also has a datastore to store VM images. As we can see in figure 4.1, a public cloud is used as a training environment in the Hybrid cloud virtual laboratory. A training environment has a pool of VMs or containers that can be used to create Teams. In a public cloud, a Team that consists of several VMs can be isolated using a Virtual Private Cloud (VPC) [5], and Firewall. For a team consists of containers in one VM, the Team can be isolated using VLAN that can be configured on a virtual switch. A Firewall and network address translation (NAT) are used to isolate the Team from the Internet but still accessible by the training participants.

A public cloud provider has a rule or policy regarding the usage of their resources. Cyber Security training activities such as sniffing using Wireshark or port scanning using NMAP could be considered as malicious activities. The cloud provider must be informed about the cybersecurity training activities, and they need to approve it. Another policy that must be handled is sharing the root access to the training participants. Some cloud providers do not let the customer share their root credentials to other people, while some training exercises need root access on the VM.

At the private cloud site, the Tele-Lab Backend must be able to communicate with the host controller in the Public Cloud to create the Team as needed by the training scenario. Most public clouds provide an API or a CLI for a client application to communicate with a Host Controller. To be able to use the API or the CLI, the Tele-Lab Backend must have a middleware as the client application to manage, start or stop VM instances in a public cloud. Every training scenario

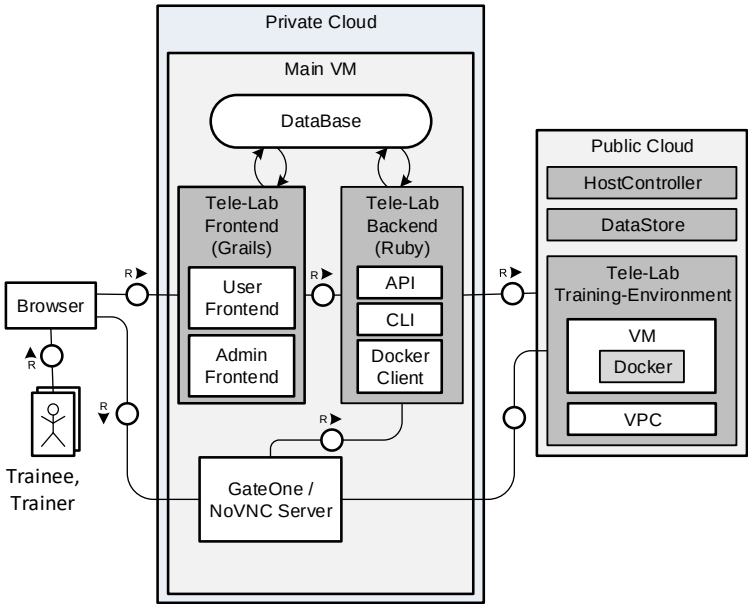

Figure 4.1: Virtual Laboratory Architecture in Public Cloud

has a script in Tele-Lab Backend for creating a Team. To create a Team that consists of containers, the Tele-Lab Backend must be able to communicate with a Docker Daemond inside a VM. The Tele-Lab Backend can use docker API or docker CLI client in the middleware to manage container instances in the VM to create a Team. The Tele-Lab backend can also use SSH connection and run a shell script to create a Team from containers in a VM. It also can use a messaging protocol such as XML-RPC, but the XML-RPC server must be installed on the VM.

After the Team is ready, the user (trainee) accesses the Team using SSH via GateOne or remote desktop using NoVNC server. To increase the performance, GateOne or NoVNC server can be installed on the public cloud site, so that the user can directly access the Team in the public cloud. To get updates about the situation in the public cloud, the Tele-Lab system in the private cloud must execute functions periodically and also trigger by an expired time of a Team.

All information about the usage of resources and the capacity of resources are stored in a DataBase in the private cloud. Another part of the Tele-Lab at the private cloud remains the same as in the existing Tele-Lab in the private cloud. On the trainee site, the browser is also the same that it has the HTML5

capabilities.

4.3 Middleware

The main part that enables an automatic interaction between virtual laboratory application and a cloud provider is cloud API. Each cloud provider has its specific API that could be used by the customer/user to manage their resources or services in the cloud. An application such as Tele-Lab needs a middleware developed using Cloud API to start and stop VMs in the cloud provider. The middleware needs to be able to classify which API to be used when interacting with a cloud provider. In this section, we describe some cloud APIs and library that could be used to manage resources in cloud providers.

The middleware mainly manages VM Instances, network and firewall configuration to make the virtual training environment isolated but still accessible by the training participants. It has three modules, i.e. a command parser, configuration setup, and executor. The command parser module adjusts the commands to use the specific Cloud API of the selected cloud provider. The configuration setup module configures VM instances using configuration management tools such as Saltstack[1] and Ansible[2]. It also configures the networking, network isolation, network address translation (NAT) and firewall. The executor module executes the commands and the configurations on the public cloud provider. This module communicates intensively with the cloud provider to build the virtual training environment and to get feedback from the cloud provider. It is also responsible for shutting down a virtual training environment.

The cloud APIs are mostly REST-based APIs, and they are specific to each cloud provider. The cloud API needs an access key to execute commands on the public cloud provider. This key could be created on the public cloud provider web interface. Amazon elastic compute cloud (EC2) has an API for almost each programming language such as Java, C++, and Python. Amazon bundles the API in a software development kits (SDK) for each programming language [6]. Other public cloud providers such as Google and Azure also have a similar SDK which

[1] https://www.saltstack.com/
[2] https://www.ansible.com/

includes libraries to provide basic functions such as signing requests and retrying requests, to make it easier to get started.

There are some cross-platform cloud APIs. Petcu et al. [83] propose a cross-platform cloud API that was part of mOSAIC project[1]. Petcu et al. also published a paper called "Experiences in building a mOSAIC of clouds" which presents an integrated overview of the mOSAIC approach and the development of applications using services from multi-cloud providers [84]. Some libraries are available that could be used for multi-cloud providers such as Libcloud[2] for python and Jcloud[3] for java. These libraries provides cloud API for AWS[4], Google Cloud[5], Azure[6], Openstakc[7], etc. The middleware could be built using these libraries.

4.4 Chapter Summary

In this chapter, we describe the use of a public cloud to provide more resources for Cybersecurity virtual laboratory. We propose an architecture that similar to the virtual laboratory on private cloud [70] where the public cloud is only used to run virtual training environments. The public cloud provides VMs and networking to build a virtual training environment. A middleware is built using cloud APIs to have automatic interactions between the virtual laboratory application and the public cloud providers. The challenges in using public cloud service to run a cybersecurity training are the isolation of the virtual training environment, port forwarding to let the user access the VM, policy of giving root access to training participants, and the permission to run hacking tools in the public cloud. These challenges could be handled using containers to represent a node in a training scenario. The containers are running on a VM.

[1] http://www.mosaic-cloud.eu/
[2] https://libcloud.apache.org/
[3] https://jclouds.apache.org/
[4] https://aws.amazon.com/
[5] https://cloud.google.com/
[6] https://azure.microsoft.com/
[7] https://www.openstack.org/

Chapter 5

Crowd-Resourcing Virtual Laboratory

Crowdsourcing method is used to provide a large number of resources by collecting resources from the crowd and integrate the resources into the Virtual Laboratory system. This method is a crowdsourcing of simple tasks or an integrative form of Crowdsourcing. Contributions are voluntary, and incentives may include self-benefit from the system by having priority to use the resources or satisfaction of contributing to a public good[94].

Similar to the peer-to-peer (P2P) cloud [108], to increase scalability, Tele-Lab could gather more resources from the people or the crowd without additional cost (budget). The crowd as contributors share their resources in the form of a virtual machine (VM) that can be used to run virtual laboratory activities. In this chapter, we present a Crowd-Resourcing Virtual Laboratory (CRVL) system that could automatically integrate the VM that was shared by a contributor to be used for Virtual Laboratory exercises. Section 5.1 describes the CRVL architecture, which is designed based on Tele-Lab architecture. Section 5.2 introduces a Team Placement Algorithm to select the best VM to run a Team for a particular user. Section 5.3 elaborates the processes of VM integration into the CRVL. Section 5.4 describes the VM fault recovery to recover the Teams on other VMs and divert the VM users to other VMs. Section 5.5 summarizes this chapter.

A crowd-resourcing virtual laboratory obtains some of the resources from the crowd. The virtual laboratory is for IT Security e-Learning, where a trainee needs

an isolated laboratory environment to do the practical exercises. The isolated laboratory environment (called as a Team) consists of virtual machines (VMs) or containers and virtual network devices. The crowd contributes their resources such as virtual machines or physical machines, to the virtual laboratory. The virtual laboratory automatically occupies the contributed resources and uses them to create a Team. The team that consists of containers is running in a VM. Since there could be a lot of VMs available, the system needs to select the best VM to run a Team.

5.1 Crowd Contribution

A person or a company can contribute resources in the form of a VM, a bare metal system, an account in a public cloud, a private cloud and an isolated VM Team. An account in the public cloud represents a resource in a public cloud. Using the account, the virtual laboratory (Tele-Lab) system can use and manage the resource, based on the training scenarios, similar to the Tele-Lab public cloud. In this thesis, we focus on a VM as a crowd-resource.

The Contributor should prepare the resources by installing all the software needed to become one of the Tele-Lab training environments in the Tele-Lab system. The Contributor must download an operating system (OS) image provided by the Tele-Lab system, to be used to run a VM or a physical machine. The OS image has been prepared with the needed software applications as well as an application (agent) to get integrated with the Tele-Lab system. This agent is used to verify the resource and to monitor user activities on the resources. The Contributor also needs to define the resource specification such as the amount of memory, the number of CPUs, the hard disk capacities, and bandwidth. Later, we need to define the minimum specification of the hardware resources that can be integrated into the Tele-Lab system.

If the resource is an account in a public cloud or a private cloud, Tele-Lab system executes a verification function using API or CLI to verify the resource whether the resource is qualified to be a training environment. The verification result will be sent to the Tele-Lab primary system so that it can do some actions

accordingly. In case of an isolated VMs Team, the isolation is the responsibility of the contributor.

The contributor resources can be shared based on a time range. Outside of this time range, the resource is released from the Tele-Lab system. This way, a person or a company that has an idle resource at the specified time, can share their resources and contribute to the Tele-Lab system.

The contributor shares a root account credential of the resources by sending it to the Tele-Lab system. Using the root account, the Tele-Lab system has full control of the resource and able to create and manage a training environment in the resource. To be able to control the system without intervention, the root credential is changed by the system while the resource is being occupied. The contributor gets the new credential when the system releases the resource. The system should map the contributor to its resources. This way, the contributor can have a priority to use their resources.

In creating a Team, combining several VMs from different location of a contributor cannot be done, because Routers and firewall devices do not forward broadcast. It is also hard to isolate the training environment from the Internet because the interconnection between VMs is through the Internet. The Contribution of one VM can be used only with containers and Linux based. For Contribution with Windows-based OS needs more than one VM at the same isolated Network. VPN can interconnect VMs from a different location, but it is hard to configure, and it consumes a lot of bandwidth.

5.2 CRVL Architecture

The crowd as contributors share their resources in the form of a VM. The VM is used to run a virtual training environment where containers are used to represent nodes in virtual laboratory scenarios. We enhanced the container based virtual laboratory architecture (Figure 3.1) by adding crowd-resources as another platform to get more VMs. As the container-based virtual laboratory is based on Tele-Lab, CRVL uses private cloud platform to run the main applications such as frontend and backend on a host or a VM. CRVL could also use VMs running on

a public cloud platform to run a virtual training environment. Figure 5.1 shows the general architecture of Tele-Lab.

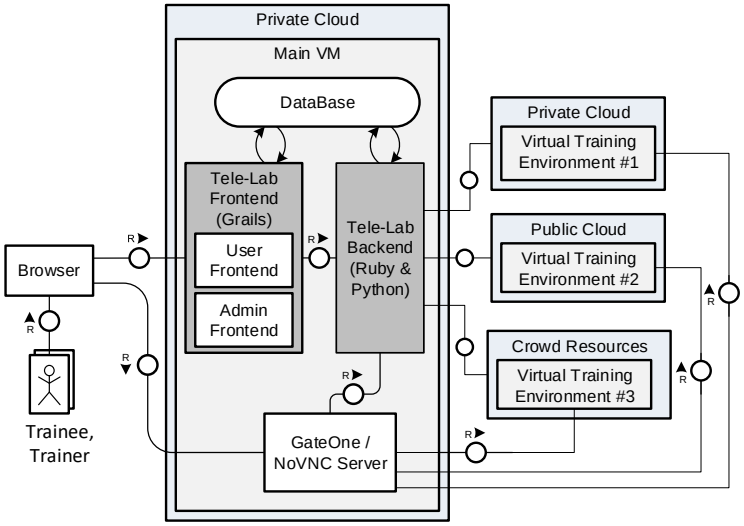

Figure 5.1: CRVL General Architecture

The CRVL Architecture is based on the Tele-Lab architecture on a private cloud platform. We add some functions in Tele-Lab middleware to communicate and occupy the public cloud and the crowd resources. The middleware is in the Tele-Lab Backend. It is built using an Application Programming Interface (API) or a Command Line Interface (CLI) that was provided by the public cloud or the resources platform. The frontend application is the same as the one in the container-based virtual laboratory. In the CRVL architecture, there are three types of virtual training environment.

Virtual Training Environment (VTE)#1 is in the private cloud where the main application is running. It could be used to create a Team consists of VMs and containers. This private cloud belongs to the owner of the virtual laboratory system. VTE#2 is located in a public cloud. It could also be used to run a Team based on VMs, but we need to have permission from the cloud provider to run attacking tools on the VM. The VMs need to be isolated from other tenant's VMs by using a virtual private cloud (VPC) on the public cloud. The owner of the system has an account in the public cloud and uses the public cloud resources to run the training environment. VTE#3 is located in Crowd-Resources. It

could only be used to run Teams based on containers. The Crowd resources are resources that are owned by the people (crowd). These resources are shared to be used by the Virtual Laboratory system for a specified time frame.

All types of contributed resources (Crowd-Resources) must be able to be integrated into the main system of the Virtual Laboratory. The Crowd-resources is used only as a training environment. In this architecture, the Tele-Lab Backend must be able to communicate with all the supported types of resources. Besides the ability to create a Team in the training environment of the private and the public cloud, the new Virtual laboratory system must also be able to create a Team in a VM and a bare Metal system. The new system must also be able to use a VM Team that was created by a contributor.

As mention above, the resources from a contributor could be in the form of a physical machine, an account in a public cloud, a private cloud, and a VM. The architecture of each resource form could be different. Figure 5.2 shows the common architecture of crowd-resources. It has a platform controller running on the private cloud next to the Tele-Lab Backend to communicate with the platform running a VM. Mainly the platform controller is a collection of APIs of public clouds, private clouds or Container platform. The crowd-resources have data storage to store information about the resources such as the location, memory, CPU. In a public cloud, the training environment could have a virtual private cloud (VPC) to isolate a group of VMs when the VTE uses VMs as nodes in the training scenario.

When the containers were used as a node in the training scenario, the containers are running inside a VM. The VM could have Docker Daemon, Dockerfile, Docker images if Docker is used to run containers. The VM could also have an agent for the software configuration or VM monitoring. The VM could have GateOne when the VM is running on a different location than the main virtual laboratory application, such as on contributor's network or a public cloud. The GateOne is installed on the VM to have a direct connection between trainee and a container in the VM as shown in figure 3.5.

In this chapter, we focus on a Team consists of containers and virtual network switches run on a VM. When someone wants to contribute in the form of a VM, he has to download a certain VM image from the virtual laboratory repository

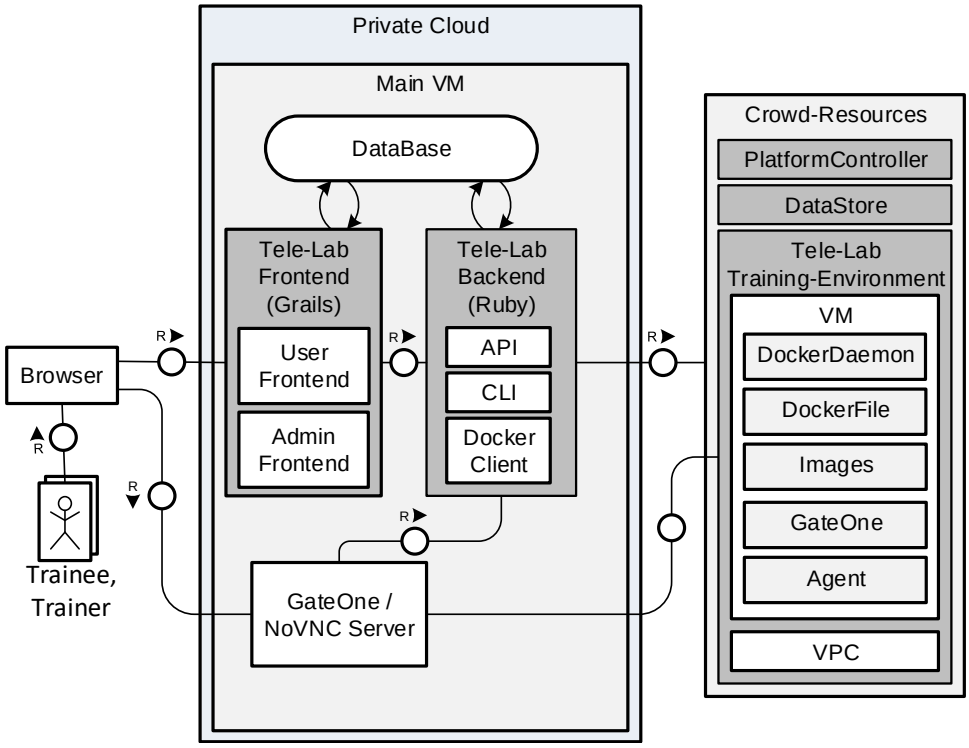

Figure 5.2: Crowd-Resources Architecture

and run a virtual machine based on that image on his host system. He also needs to submit a root privileged user account and the VM IP address to the virtual laboratory system. Using this root privilege user account, the virtual laboratory system occupies the VM. The virtual laboratory system creates a Team in the VM when a trainee needs to do the practical exercises. The trainee gets console access to one of the containers in a Team to do his tasks. The console access is given using Gateone interface, an HTML5-powered terminal emulator and SSH client. Gateone was installed on each VM.

5.3 Team Placement

In this section, we focus on a Team which consists of containers running on a VM. Since there are more than one VM available on the CRVL, we need to provide an algorithm to select the best VM based on a requesting user location and a load of VMs. We present CTPlace, an algorithm for Team Placement in a crowd-

resourcing virtual laboratory. CTPlace groups the VMs into hierarchical tree clusters based on the Geo-location of the VMs. CTPlace has two steps in Team placement. First, it selects the nearest cluster to the trainee location to get the highest throughput. Second, it selects a VM inside the selected cluster. To select a VM inside a public cloud cluster, it uses Most-Full-First (MFF) algorithm to reduce service cost by reducing the number of running VMs. To select a VM inside a private cloud or within contributed resources, it uses Least-Full-first (LFF) and Tag-Pack to balance the load and try to place the same type of Teams on the same VM. We compare the CTPlace with three other placement algorithms in a simulated environment, to evaluate the performance of the CTPlace.

5.3.1 Motivation

A Team can consist of containers and virtual switches. Although a container uses much fewer resources than a VM, it still consumes some of the resources. The number of available resources limits the number of containers that could be running on a virtual laboratory. When the limit is achieved, we need to find a way to increase the number of resources. Crowd-resourcing is one way to increase the number of resources where a person or a company can contribute by integrating their un-used resources to the Virtual Laboratory system[103].

In the crowd-resourcing virtual laboratory (CRVL), a Team that consists of containers is running on a VM. In case there are a lot of VMs available, the system needs to select the best VM to place the Team. The Research problem of this section is on how to place a Team in a CRVL in order to keep the cost as low as possible and to maintain the performance. A placement function is needed to decide which VM to run a Team automatically.

We introduce CTPlace, an approach for team placement in a crowd-resourcing virtual laboratory. CTPlace groups the VMs into hierarchical tree clusters, based on the Geo-location of the VMs. Geo-location of a VM is obtained from the information of the VM IP address. From the IP address, we can get the location of the VM such as the city, province, country, region, ASN (autonomous system number), the latitude and longitude. Using this information, CTPlace can group the VMs into the same city, or country. In the Team placement process, CTPlace selects the nearest cluster to the trainee by checking the Geo-location of the

trainee's IP address and find the nearest cluster to the trainee. After selecting the best cluster, CTPlace selects the best VM inside the selected cluster. The best VM criteria are different between the platform where the VM is running. CTPlace uses Most-Full-First (MFF) algorithm in a Public Cloud, and it uses Least-Full-First (LFF) and Tag-Pack algorithm in a private cloud and contributed resources. We compare the CTPlace with three other placement algorithms in a simulated environment.

5.3.2 Related Work

Placement algorithm has been used intensively in a cloud computing environment, to optimize the usage of physical resources in order to continue balancing between performance and operational cost. There is much work on VM placement algorithms in cloud data centres to reduce power consumption, maximize resource utilization and avoid traffic congestion.

Usmani et al. have done a survey of Virtual Machines placement Techniques in a cloud computing environment which focus on improving energy efficiency. They concluded that every placement technique has some specific target, migration technique, prominent resources and important parameters. They also suggest that there should be an approach to minimize the trade-off between energy consumption and good performance [119]. Z.A. Mann has also done a survey in Virtual Machines placement in cloud data centres, which focus on problem formulation and optimization algorithm. Similar to Usmani, Z.A. Mann also concluded that there are significant differences among virtual placement approaches, especially in the problem formulations [64]. Challita et.al. [16] studied virtual machine placement optimization in data centers. They classify the dynamic VM placement optimization solutions into four main approaches: Constraint Programming, Bin Packing, Stochastic Integer Programming, and Genetic Algorithm. They claimed that their study provided a better comprehension of the existing VM placement algorithms that deal with power cost and handle traffic in data centres.

Mills et al. [68] compare several bin-packing-style heuristics to find the best node to run a VM. They group the nodes into clusters. To place a VM in a cloud data centre, the algorithm needs to find the best cluster and the best node within the selected cluster. They concluded that the selection of the criterion for

choosing a cluster could lead to a huge difference in provider revenue. Tordsson et al. [112] propose a cloud brokering mechanism for optimized placement of VM across multi-providers. The cloud broker places a VM based on the user requirement. They concluded that multi-cloud deployment provides better performance and lower cost compared to a single cloud only.

Container placement is similar to VM placement. Container Placement in docker swarm uses three approaches, i.e. spread for load balancing, bin packing to reduce costs, and random. Peinl et al. [80] reports a survey result and propose their solution regarding Docker cluster management in the cloud where one of the functions is container placement within multiple hosts.

Beside VM and container placement, there are also service placement and data placement with different objectives. Eyad et al. [91] propose a secure placement algorithm to place a tenant (data) in a SaaS environment. Selimi et al. [97] propose a service placement approach that aware of bandwidth in community network clouds. They group the nodes based on their Geo-location using the k-means algorithm and select the head of the cluster based on the bandwidth to each node. The service is placed at the head of the cluster. Unuvar et al. [118] propose a hybrid cloud placement algorithm to automatically decide whether an application should be deployed on-premise, in a public cloud, or across private and public clouds.

Google has kubernetes[1] to manage containers inside their cloud system. Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. Kubernetes container placement based on current load of the host, collocation constraints, and availability constraints. It balances the load by spreading the pods across nodes. A pod is a group of one or more containers. Besides balancing the load, Kubernetes can also be configured to place a specific type of pod on a specific type of node. Docker Swarm[2] has clustering and placement tools for Docker containers. It uses spread strategies to determine which nodes, each container should run. Spread is used to distribute tasks evenly, over the values of the data centre node label. It balances containers across the nodes in a cluster, based on the nodes' available CPU and RAM.

[1] https://kubernetes.io/
[2] https://docs.docker.com

The placement algorithm might also be needed in a social cloud such as Subutai[1] and Mastodon[2]. In Mastodon, a user selects a server based on his interest. Subutai is a P2P cloud, as far as we know, there is no publication regarding the placement in Subutai.

Those placement algorithms could not be applied on CRVL, because CRVL has a unique architecture consist of private cloud, public cloud and crowd-resourcing platforms. Especially in the crowd-resourcing platform, the VMs and the users could be anywhere around the world. Geo-location must be used to pair a user (participant) to a VM. Our proposed algorithm combines several placement algorithms to tailor with the platforms and to get the best VM for a participant based on Geo-location and the load of the available VMs.

5.3.3 Team Placement Algorithm

Trainees and the contributors could come from all over the world. To cover the user and contributor from all over the world, we divide the world into hierarchical tree zones. Starting from the world as the root (zero levels), goes to the region as the first level. The region is divided into sub-regions such as Southeast Asia. The sub-region is divided into countries, and the country is divided into provinces, a province is divided into cities. Each region should have its Virtual laboratory server. Figure 5.3 shows the hierarchical tree zones.

In the virtual laboratory architecture, the primary virtual laboratory server is placed on a private cloud. Every region has a mirror of the virtual laboratory server to serve users in the same region. These mirror servers are located on public clouds. These servers are called region servers or secondary servers. A Team can be run on the VMs inside a private cloud, a public cloud or inside a contributor virtualization system. To let the users have access to the Team, Gateone is used as an SSH Gateway to one of the containers in the Team. This gateway will be installed in every contributed VM to shorten the route. This way, the user's browser has a direct connection to the Gateone server via a javascript function. User activity on the gateway server must be logged and sent to the Tele-lab server for monitoring and controlling functions.

[1] https://subut.ai
[2] https://mastodon.social

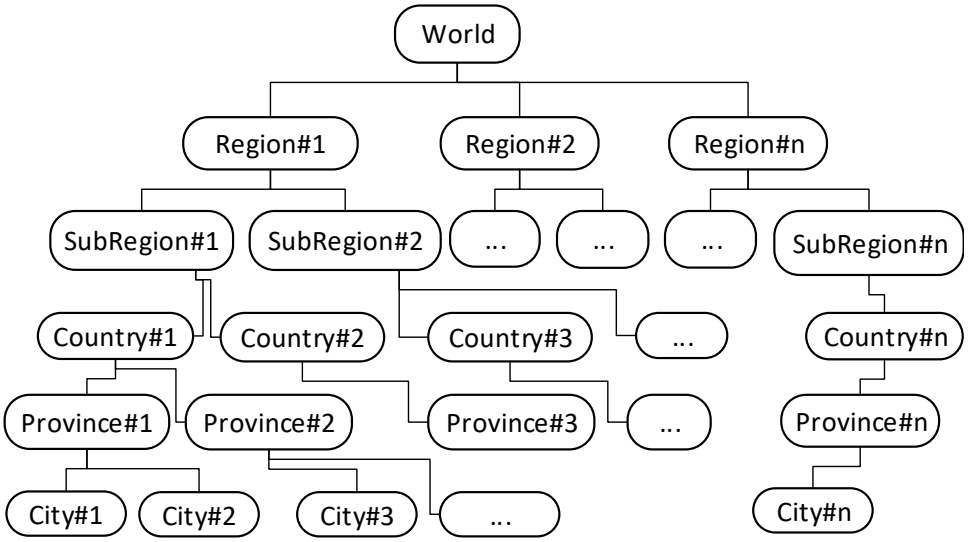

Figure 5.3: Hierarchical Zones

Team placement is a process of selecting the best VM to place a Team. The placement will be done online within short timescales. An online algorithm processes its input piece by piece in serial fashion, in order that the input is fed to the algorithm without having all the entire input available from the start. There will be no live migration because the lifetime of a Team is only around one to two hours or even shorter. The objectives of team placement are cost efficiency and performance. The load and throughput define the performance. To get higher throughput, a user should be connected to the nearest VM. To manage the load within an acceptable performance, we limit the load of each VM to 80%. A Team could have a different type of load, and it could be a high CPU load, high RAM load or high traffic load.

While the crowd-resourcing virtual laboratory consists of three platforms, i.e. private cloud, public cloud and crowd-resources, the objectives of team placement could be different on each platform. The cost efficiency will be more considered when selecting the best VM inside a public cloud because running more VMs means more payment. Public cloud is a pay per use service where a user needs to pay the cost of renting cloud resources and the cost of communication to the outside of a public cloud provider. In a private cloud, the cost of resource usage

is already paid up front. In a crowd resource platform, the cost is paid by the contributor. In these both platforms, the concern is more to the performance.

Contributor resources could be anywhere around the world. To narrow the search of the best VM, we cluster the VMs based on their geographical location. From the IP address, we can obtain the Geo-location such as the country and the city of a VM. This Geo-location information is used to create the same hierarchical zones as figure 5.3. We assume that the throughput and delay of the communication inside the same geographical location are better than between different geographical location. For example, the throughput within a country should be higher compared to the throughput between countries. We are proposing a Team Placement approach that aware of the Geo-location clusters to be able to get a VM as close as possible to the user (trainee) so that the throughput is higher and the delay is lower. This team placement is called as CTPlace.

CTPlace consists of two steps. First is to search for the smallest and the nearest cluster to the user. Second is to search for the best VM in the selected cluster. CTPlace starts by getting information on a user IP address, to get the Geo-location and the ASN (Autonomous System Number) of the user. In the Internet routing system, an autonomous system (AS) is a collection of the connected IP networks under the control of an administrative entity. This administrative entity defines and manages routing between networks in the autonomous system, which is identified by a number called ASN. The ASN is used in exterior routing protocols such as BGP (Border Gateway Protocol) or EGP (Exterior Gateway Protocol). The throughput between computer inside an autonomous system is usually higher than the throughput to the outside of an autonomous system because the connection inside an autonomous system is an Intranet or a Local Area Network connection.

If a user was in the same ASN as some VMs, CTPlace allocates one of these VMs to place the Team for the user. When there was more than one VM within the same AS, CTPlace searches the VM that has the nearest (if not the same) geo-location to the user, starting from the country to the city. If there was no VM in the same country as the user, CTPlace uses Tag-Pack and Least-full-first to select the best VM inside the AS.).

Algorithm 1 Team Placement algorithm

```
1: Input: UserIPAddress, TeamType, Infrastructure.
2: Output: selected_VM
3: Initialization: isPlaced ← False
4: userLocation ← ipInfo(UserIPAddress)
5: for each region in regionList do
6:     #Search for the same country with the user
7:     for each country in region.countryList do
8:         if country.id = UserIPAddress.country then
9:             if country.checkLoad(Team) is True then
10:                #Search the smallest cluster until the city
11:                selectCluster(country)
12:                #Select the best VM in the selected Cluster
13:                select_Tag_LFF_VM(selectedCluster)
14:                isPlaced ← True
15:            end if
16:        end if
17:    end for
18: end for
19: if isPlaced = False then
20:    #search for a VM in neighboring countries
21:    selectCluster(user region)
22:    vm = selectVM(selected cluster)
23:    if vm is not None then
24:        isPlaced ← True
25:    else
26:        selectCluster(other regions)
27:        vm = selectVM(selected cluster)
28:        if vm is not None then
29:            isPlaced ← True
30:        end if
31:    end if
32: end if
33: if isPlaced = False then
34:    rejectUser()
35: end if
```

If there was no VM in the same AS as the user, the search for a VM is started from the country because it is the highest cluster level that can be taken from an IP Address information. If there were some VMs in the same country as the user, then CTPlace searches for the same province. If there was more than one VMs in a province, CTPlace searches for the same city. Algorithm 1 shows the simplified algorithm of CTPlace where no VM in the same ASN as the user.

If there were no VMs in the same country, CTPlace searches for a VM in the nearest neighbours. If there was more than one neighbour has available VMs, CTPlace selects the neighbour that has a VM with the highest throughput. Throughput is measured using Iperf between a VM and a region server. If there was no VM available on the nearest neighbours, CTPlace continues searching for an available VM in any country in the sub-region. If there was no VM available in all countries in the region, CTPlace searches for a VM in a private cloud. If there was no private cloud in the region, CTPlace searches for a VM in a public cloud. Every region should have a public cloud. If there was no VM available in the public cloud, it continues searching until all the regions are searched. If there was no VM available, the request for a Team is rejected.

After CTPlace selected the best cluster, it selects the best VM inside the cluster. CTPlace uses the tag-pack and least-full-first algorithm to select a VM within a contributed resources cluster and within a private cloud cluster. The Tag-Pack is used to place the same Team type on a VM. When all the VMs have already been marked with a Team type, CTPlace uses a least-full-first algorithm to select a VM and place a new type of Team in the VM. In our approach, a VM could be marked with more than one type. We use least-full-first algorithm to balance the load to all of VMs in a private cloud and crowd-resources cluster. When all the VMs in the selected cluster is empty, CTPlace will select the one with the highest throughput to the region server.

CTPlace uses Most-Full-First(MFF) algorithm to select the best VM inside a public cloud. Using this algorithm, we can reduce the number of running VMs in the public cloud to save cost. MFF selects the best VM by searching for a VM with the highest load but still has the capacity to handle the Team Load.

Placing the same learning unit Teams in a VM has some benefits, such as the same VM configuration for the same learning unit, no need to download the same

files to a different VM, and isolate security vulnerability in a VM. If a user was able to use the vulnerability and get access to the VM, he could only get access to the same VM not on the other VMs. However, in a situation where the resources are limited, we do not have other choices except to put some different learning unit Teams on the same VM. To put the same learning unit teams on the same VM, we use Tag-Pack algorithm.

5.3.4 Evaluation

In order to evaluate our approach, we conducted several experiments with different conditions and compared the results with other placement approaches, i.e. Most-Full-First(MFF), Least-Full-First(LFF), and Random. We implement these four algorithms in Python language. We simulate the virtual laboratory infrastructure and the user requests as closely as possible to the real environment.

In our simulation, we create a virtual laboratory infrastructure that consists of 5 regions, ten countries per region where only five countries have contributed VMs, one public cloud per region, and one private cloud. Each country has 4 VMs, each public cloud has 5 VMs, and a private cloud has 10 VMs. To simplify the experiment and the comparison, we exclude sub-region, province and city from the virtual laboratory infrastructure simulation. These exclusions will not affect the result of experimentation and comparison.

The capacity of each VM is randomized between 40 to 60 loads. We assume that one VM might have a memory between 1.5 GB and 3 GB. The maximum capacity of a VM is defined by the hardware resources available on the VM such as RAM, CPU, and Hard disk. The higher the resources, the higher the maximum capacity. Based on our experience, 40 Teams could be run on a VM with 1.5 GB memory and 1 CPU. The team was for a MITM attack learning unit where Ettercap was installed and running on one of the containers. When all Teams were running, the CPU load was 100%, and the thread was around 300. Even though the load was so high, the user can be still able to run the exercise on one of the containers. The performance started to drop significantly when the VM started to use the disk cache (swap).

The type of user request is randomized according to the team type. There are three types of teams with the load between 1 to 3. The country of a user is

also generated randomly. In our experiments, we tested the placement approach against different number of users (different number of loads), to check the placement result in the light, medium and heavy conditions. Light condition is where the load is much below the capacity of the whole infrastructure. Medium condition is where the load is closed to the capacity, and the heavy condition is where the load is much above the capacity. In this experiment the light condition is 3000 users, the medium condition is 3500 users, and the heavy condition is 4000 users.

Our approach uses Geo-location to select a VM that geographically the nearest to the user, while other approaches do not use Geo-location. MFF selects the fullest and the smallest cluster available and then selects the most full VM from the selected cluster. LFF selects the least full and the smallest cluster available, and then select the least full VM from the selected cluster. Random selects the cluster and the VM randomly.

An experiment was executed by generating a virtual laboratory infrastructure and a number of user requests. After that, all the placement approaches were used to place the same user requests (Teams) on the same virtual laboratory infrastructure. We executed three experiments for each number of user requests and calculated the average of the result. We collect some data from the result of each placement approach, such as the total number of VMs that has been used, the total number of VMs running on public clouds, the number of team types on a VM, the number of teams that being placed on another country or region, and the number of users that has been rejected. Figure 5.4 to 5.9 show the results.

From the data that has been collected, we compare the cost and the performance of the placement approaches. To compare the cost efficiency, we use the data of the total number of VMs that has been used and the number of VMs on the public clouds. More VMs means more cost. This cost could be the cost of using services or the cost of electricity and bandwidth usage.

Figure 5.4 shows the total number of VMs that has been used by each placement algorithm to place such number of Teams. MFF uses the lowest number of VMs because MFF places a Team into a new empty VM, only when all the VMs in use have already been full. In our virtual laboratory architecture, the private cloud has already been paid up front, and the contributor pays for the crowd

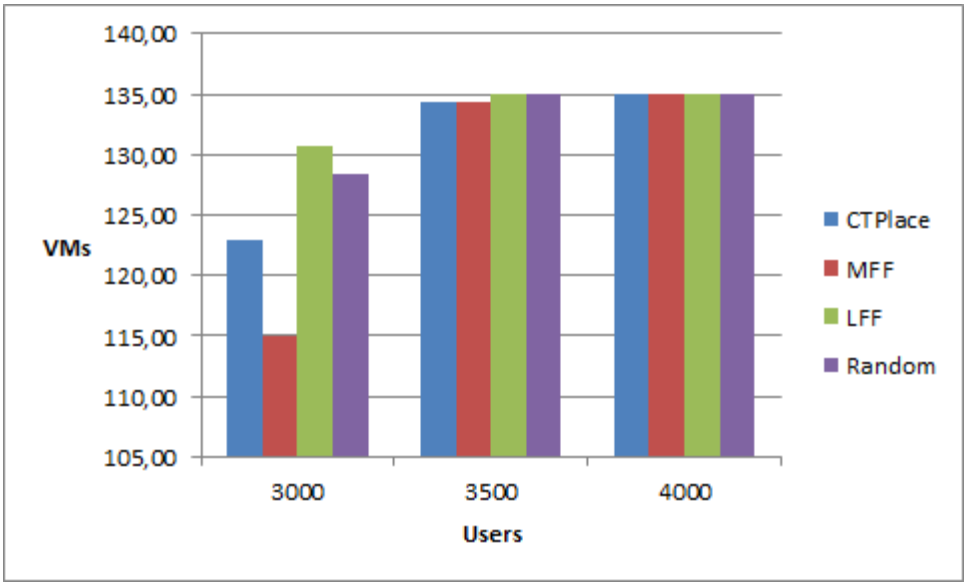

Figure 5.4: VM Number

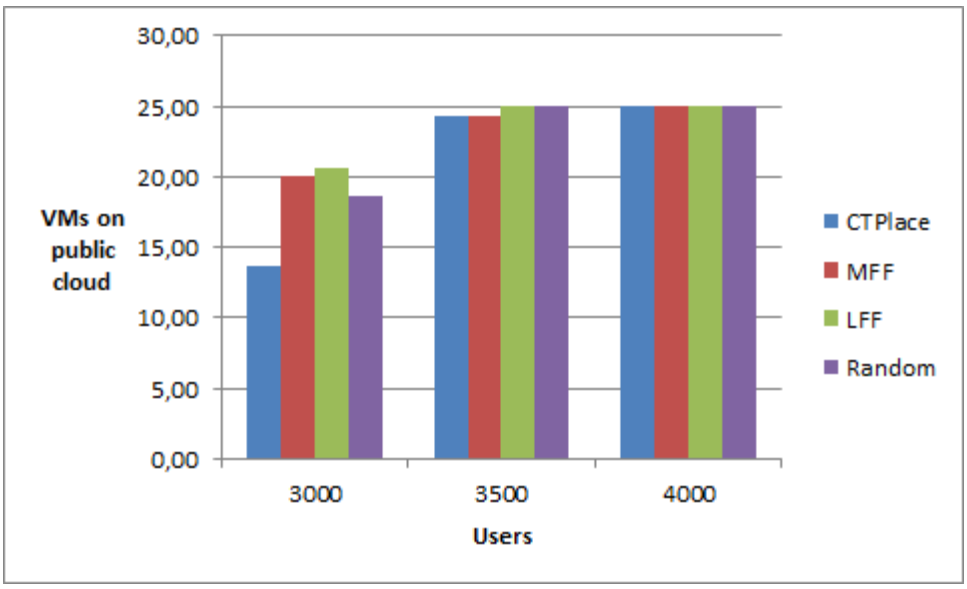

Figure 5.5: VM in Public Cloud

resources platform. We can focus on the public cloud when we want to calculate the cost. Figure 5.5 shows the total number of VMs in the public cloud. CTPlace uses the lowest number of VMs in the public cloud. In this context, CTPlace can give a better cost-efficiency (lower cost) compare to another placement approach.

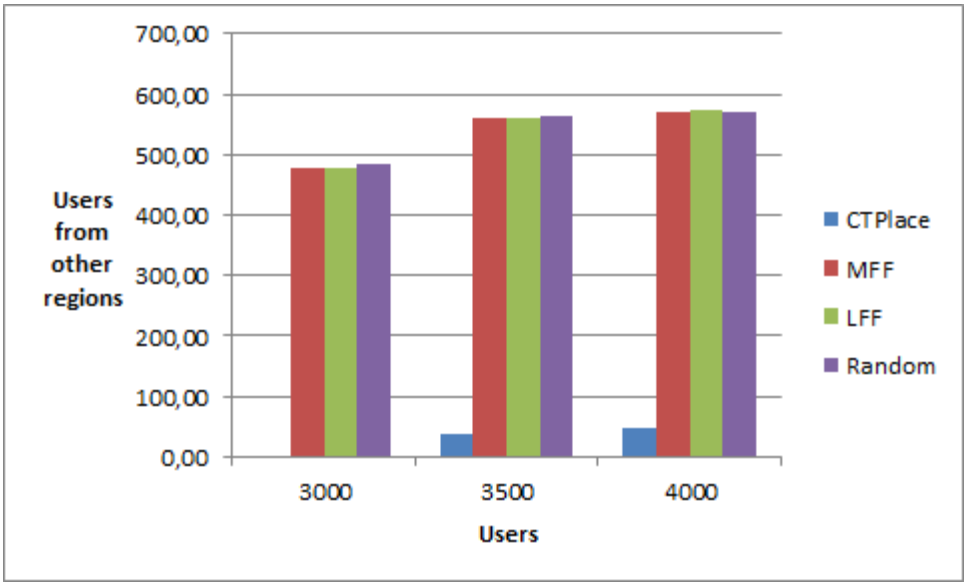

Figure 5.6: Teams in other Region

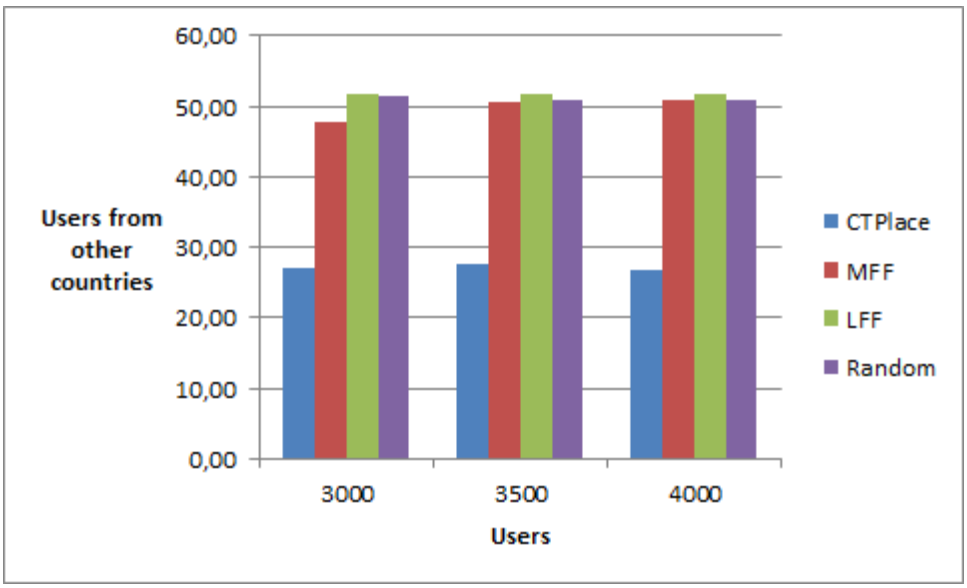

Figure 5.7: Teams in other Countries

To compare the performance, we need to look at the load and the throughput. In our scenario, the VM load should not be more than 80% of the maximum VM capacity. For example, a VM has a maximum capacity to serve 50 Teams, and the system will allocate only 40 Teams to this VM. So, no matter which

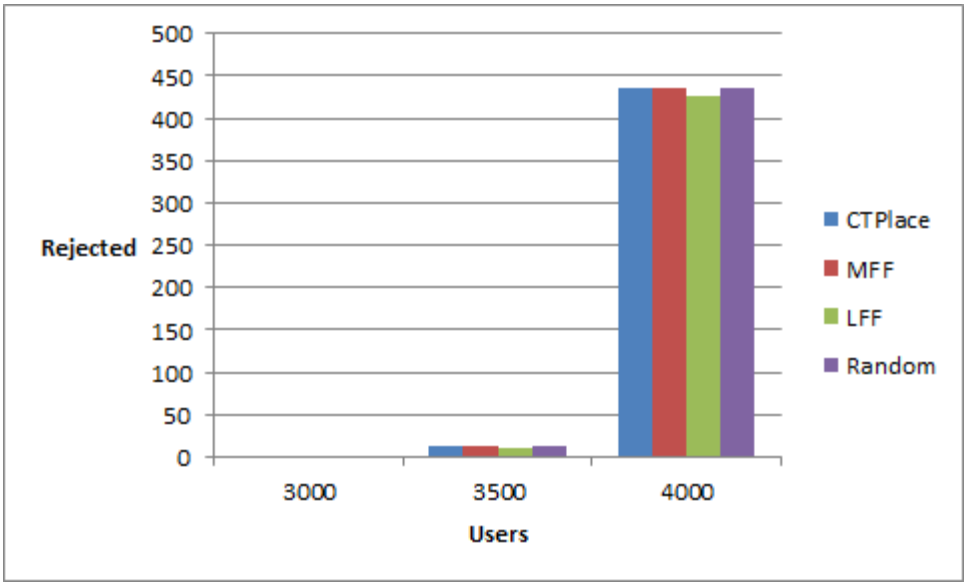

Figure 5.8: Rejected Users

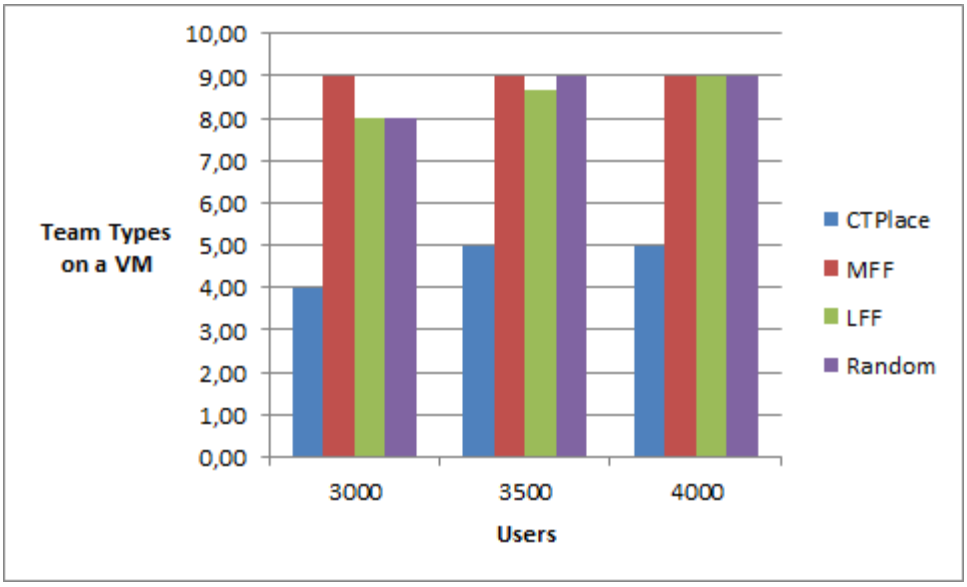

Figure 5.9: Tag Number per VM

placement algorithm was used, the load is still acceptable, because a VM will not be overloaded.

In our approach, we were trying to place a Team within the same Geo-location as the VM or as close as possible to the VM. From the experiment, we have

collected some data about the number of Teams that had been placed on VMs in different Geo-location than the team location. Figure 5.6 shows the number of Teams that had been placed in a different region. Figure 5.7 shows the number of Teams that had been placed in different countries. From these two figures, we can see that CTPlace can give the lowest number of Teams that had been placed in a different Geo-location. That means CTPlace has given the highest throughput compared to the other three placement algorithms.

The number of rejected users can also be used to measure the placement algorithm's performance, because the system needs to be able to serve user as much as possible. In figure 5.8, we can see that there are no rejected users when the load is much below the capacity. When the load is much higher than the capacity, all the placement algorithms have rejected almost the same number of users. The LFF has rejected a smaller number of users compared to the others because the LFF can place the Teams in a more balancing way so that the teams can be placed in the better combination.

As explained in section IV, there are some benefits in placing the same team types on the same VM. The variety of the team inside a VM can also be used to show the performance of the placement algorithm. The lowest the variety the better the performance. From figure 5.9, we can see that CTPlace has given the lowest number of team types in a VM because CTPlace implements TagPack in the placement process.

Overall, CTPlace can give a better cost efficiency and a better performance, because it uses a specific placement approach on a specific platform, based on the objective of the placement.

5.4 Virtual Machine Integration in CRVL

In the CRVL system, the crowd could contribute by sharing their resources in the form of a Virtual Machine (VM), a physical machine, an account in a public cloud, and a private cloud. In this section, we focus on the automatic integration of a VM into the CRVL system. Integrated means the VM is occupied and monitored to be used as a computing machine to run a virtual laboratory environment for some users. Some integration processes need to be done to make

sure the integration is running smoothly, and the VM could be securely used to run the virtual laboratory. We propose a mechanism of Integrating a VM into a crowd-resourcing virtual laboratory. It starts with verifying the VM integrity, followed by configuring the VM until the VM is ready to be used to run a Virtual Laboratory exercise. We use WebSocket for communication between VM and the Virtual Laboratory Server, to be able to monitor the VM Integrity. To evaluate the integration mechanism, we evaluate the performance by measuring the time needed to integrate a VM.

5.4.1 VM Integration Mechanism

The integration process consists of three phases, i.e. registration, verification, and configuration. In the registration phase, the contributor must manually fill a form in the CRVL registration web page to register the VM. The contributor needs to enter the hardware specification, the host OS, the public IP and Port address of the VM. This way the VM could be accessed by CRVL participants from the public network (Internet) to be able to do hands-on exercises on the VM. The contributor needs to make sure that the firewall and the host of the VM are configured to allow access and forward traffic from the Internet to the VM via specific ports.

After the registration phase is finished, the contributor could download the VM image, configured the host machine and starts a VM using the downloaded VM image. When the VM is up and running, the VMIV Hub initiate the verification and the configuration processes. The VMIV Hub is a service or agent within the VM that is responsible for all the Integration and the fault recovery processes. After the verification process was finished, a VM is connected to the CRVL system via a WebSocket connection. The VMIV Hub sends a report about the services in the VM to the CRVL controller. If it is needed to do more configuration or installation, the CRVL controller could give commands to the VMIV Hub via the open WebSocket. The CRVL controller tests the participant's connection to the VM. If the connection was failed, the CRVL controller informs the VM contributor to fix it.

The verification and configuration phases are executed automatically by the CRVL controller and the VMIV Hub. Before a VM is integrated into the CRVL

system, the integrity of the VM must be verified remotely by the CRVL Controller. Figure 5.10 shows the VM integration scheme, where a VM in a contributor network is trying to be integrated into the CRVL system. We proposed a VM integrity verification mechanism without using any specific hardware such as the Trusted Platform Module (TPM) in section 6.1. The integrity verification mechanism is a big part of the VM integration mechanism. This VM integrity verification uses kexec[1] to reboot the VM to a well-verified kernel. In the reboot process, before the init process of the userspace, the kernel loads a module that has a function to get the hash checksum of all the system files. The integrity verification sequential diagram is shown in figure 5.11. After the verification process, the VM is connected to the CRVL controller via WebSocket. The opened WebSocket connection is used to monitor the VM. If the WebSocket was disconnected and the session ID was changed, the disconnected VM must be re-verified.

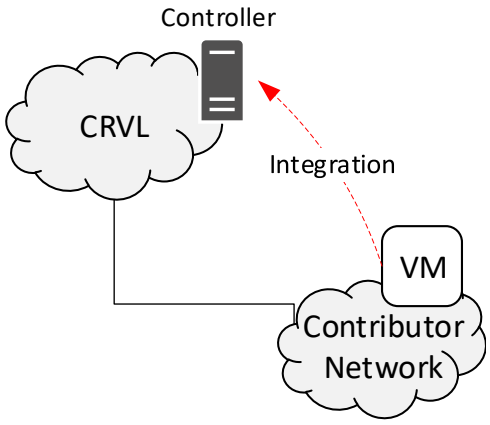

Figure 5.10: Integration Scheme

As shown in figure 5.11, the VMIV Hub initiates the integrity verification process. After the verification process, the VMIV Hub and the controller can communicate via the already opened WebSocket connection. The VM is configured as needed to be ready to run virtual laboratory instances. The VM should have docker, GateOne[2], Open vSwitch[3] and other supporting applications, installed and running. Any additional software installation and configuration could

[1] http://man7.org/Linux/man-pages/man8/kexec.8.html
[2] http://liftoff.github.io/GateOne/
[3] https://www.openvswitch.org/

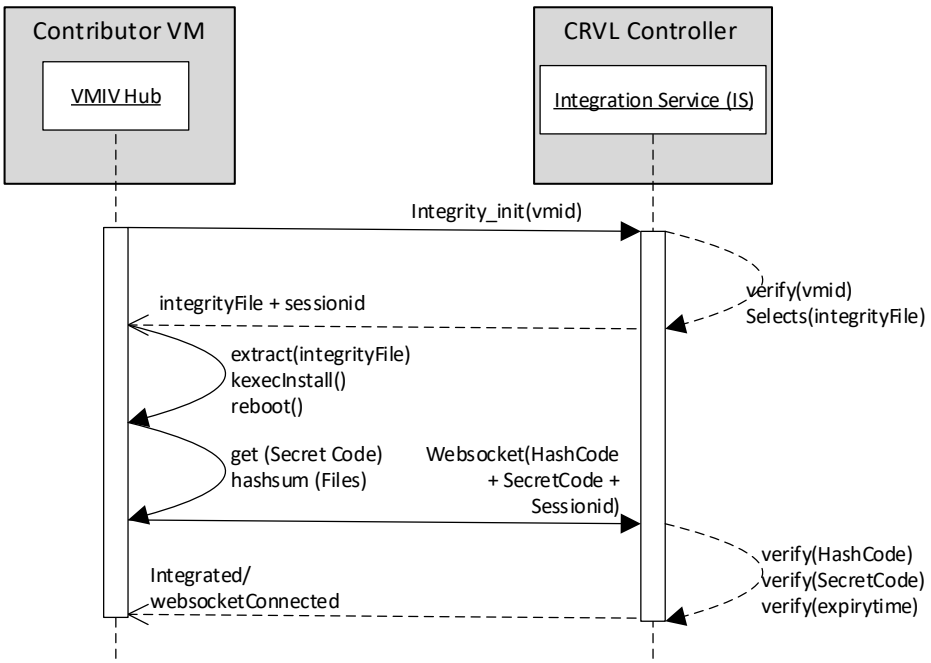

Figure 5.11: Integrity Verification Sequential Diagram.

be done via the opened WebSocket. Using the WebSocket connection, the CRVL Controller could send commands to the VM without any additional open port on the VM. The VM could be placed behind a firewall and only has a limited number of open ports.

5.4.2 Evaluation

The biggest part of the integration process is the verification of the VM Integrity which we described in section 6.1. The Integrated state is the state where the VM is connected to the CRVL controller using WebSocket. The experiment of the Integration approach is shown in sub-section 6.1.5, where the integration time was measured starting from the VMIV hub request for the kernel image from the CRVL controller until the WebSocket connection is built. The experiment was delivered in a Local Area Network (LAN) where the CRVL controller is a VM on OpenNebula and the contributor VM is running on VirtualBox. The Integration time was 1 minute 34 seconds. The time needed for the VM Integration could be

different on other places refer to speed or bandwidth different at each VM side and computation capability of the host machine.

5.5 Live Migration & Fault Recovery in CRVL

A VM shared by a contributor could be disintegrated from the CRVL system normally per schedule or suddenly without notice. When the VM is disintegrated, the CRVL system should be able to immediately provide other VMs to run Teams (containers) for the users to continue working on the virtual laboratory exercises. For the normally disintegrated VM, the CRVL system has time to prepare for the dis-integration to make the downtime as small as possible for the users of the dis-integrating VM. We propose a live migration mechanism to handle the normally dis-integrated VM. For the suddenly dis-integrated VM, we propose a fault recovery mechanism to divert users to other VMs.

These two mechanisms are quite similar, where the VM is recovered by restoring the latest states of the running Teams on the disintegrating VM to other VMs. The participants should be able to continue working on the exercises without starting from the beginning. The big difference between the two mechanisms is that the fault recovery mechanism might not recover the team (container) to the latest state before the VM down. The other difference is that on the live migration mechanism, the users will be noticed when the VM is going to be down. Both of mechanisms use SSHFS[1] to mount a remote file system on a file server to store the data of a VM, to make the user data (user logs and modified configuration files) available in the network even when the VM is down. Our experiments showed that our approaches could work appropriately, and the time for Team Recovery is similar to the time for Team Creation.

5.5.1 Related Work

The goal of fault recovery is to minimize the downtime of the system during failures. Bala and Chana [11] listed some fault recovery or fault tolerance techniques for cloud computing. They divide the techniques into Reactive and Proactive techniques. Reactive techniques wait for the fault and try to recover as soon as

[1] https://help.ubuntu.com/community/SSHFS

possible. Proactive techniques try to detect and avoid fault before it happens. These techniques are used to recover from a crashed application, a failed node or host machine. Some techniques are relevant to our problems, such as Replication, Checkpointing and Restart, and Job Migration. These techniques are reactive techniques where they try to prepare for the fault, so when the fault is happening, the system could recover the fault using the provided techniques.

Replication means to create and run task replicas on a secondary node as a backup. When the primary node was crashed, the backup node takes over the critical functionality for the system[114]. This technique is implemented in HAProxy[1]. Checkpointing and Restart preserve the state of a task or an application by checkpointing the state periodically to be able to rollback to the current state when the task is failed[11]. Job Migration is a technique that migrates a task to another resource when the task is failing in the original resource[11]. This technique could be implemented using the HA proxy.

Jhawar et al. proposed a fault tolerance management in cloud computing where the application users and developers do not become aware of the implementation details of the fault tolerance technique [49]. The fault tolerance technique is implemented in a different layer than the application layer, as independent modules where each module can transparently working on users' applications. They design a framework that offers fault tolerance as a service from a third party on the existing cloud infrastructure.

Live migration is a method to migrate a VM or a Container to another host with near zero downtime to the live services running on the VM or Container. Live migration can be used to create a secondary VM or Container on another host while the primary VM or Container is still running normally. The primary and secondary instance could be running together to provide a load balancing system. The secondary instance could also be used as the backup of the primary VM or Container. In cloud computing, live migration of a VM or a Container could also be done. Live migration is also used to backed-up the computing node state, also helps with server maintenance scenario, server consolidation and high availability within hardware zones and data centres.

[1] http://www.haproxy.org/

There are two major approaches in doing VM live migration from one host to another host[52]. The first approach is Post-Copy[18][82], which suspends the migrating VM at the source, copies or transfers minimal processor state and essential kernel data structures to the target node, resumes the virtual machine, starts the VM on the target node and begins fetching memory pages on demand over the network from the source. This approach could reduce the downtime, but it produces a much longer total migration time.

The second approach is Pre-Copy [3][39], which copies all the memory pages from the source node to the target node, without ever stopping the VM being migrated. After copying process, some memory pages could be changed during the memory copy process, because the migrating VM is still running. These memory pages are called dirty pages, that need to be transferred to the target node until the dirtying rate is less than the transferring rate. The migrating VM is stopped, and the remaining dirty pages are sent to the target node, and the VM in the target node is resumed.

These approaches have weaknesses that need to be improved. The performance after migration of the Post-Copy approach could be very low and set of pages could be fault [18][82]. The Pre-Copy approach is not suitable for copying some set of memory pages that are updated very frequently [39][3]. The live migration approaches have several challenges, such as Low Bandwidth over WAN, Network Fault, Memory intensive applications, and Memory state between clusters in the cloud [107]. These approaches (techniques) need some access into the host machine, which could not be done in CRVL system because the host is managed by the contributor and it is not shared with the Virtual Laboratory system.

Live migration could be used for backing up the VM for fault recovery. Nadgowda et al. proposed a cloud disaster recovery based on Image-Instance mapping which deduplicates changes across VMs and needs to replicate only the unique changes [73]. For Containers, there are several techniques to do live migration. Mirkin et al. proposed a live migration technique for a Container using the checkpointing and restart features which are implemented in OpenVZ[1] [69]. These features are implemented as a loadable kernel module. The technique enables the

[1]https://openvz.org/

user to checkpoint the state of a running Container and restarts it somewhere else. Live migration is also useful for fault tolerance management [69].

Nadgowda et al. proposed voyager, just-in-time live Container migration service which was designed following the Open Container Initiative (OCI)[1] [74]. Voyager performs live Container migration with minimal downtime, by binding union mount file systems with CRIU-based memory migration. Using union mount file systems, Voyager creates data federation across the source node and the target node. CRIU[2](Checkpoint/Restore In Userspace) is a software tool that could freeze a running application and checkpoint it as a collection of files on disk. Voyager can resume Container instantly on the target node while performing disk state transfer via lazy replication. Docker is also using CRIU into their system to create docker checkpoint [20]. At the moment, Docker checkpoint command can only be used for experimental, by editing the daemon.json and set experimental to true [27].

Storing backup files on a network union file system makes the files available anywhere in the network. However, backup files of memory (RAM) content could be very big, which could consume the memory, the bandwidth, the hard drive capacity and the CPU while the backup process is running. In, this section, we try to backup Teams states by backing-up the user command history and the modified files, which could be used to generate the latest Team states on another VM.

5.5.2 Live Migration

In CRVL, a user or a participant has a remote connection to one of the stations in the virtual laboratory environment. This station could be a Container or a VM, and the remote connection could be using SSH via Gateone or using a remote desktop connection. For a Container station, the participants could do their laboratory exercises by entering shell commands to configure or execute an attacking scenario. These commands are saved in the shell history of Linux system, and the Gateone user commands log. The history and the log could be

[1] https://www.openContainers.org/
[2] https://criu.org/

saved periodically to the backup server as the checkpoint to be able to roll back to the latest state when recovering from a failure VM.

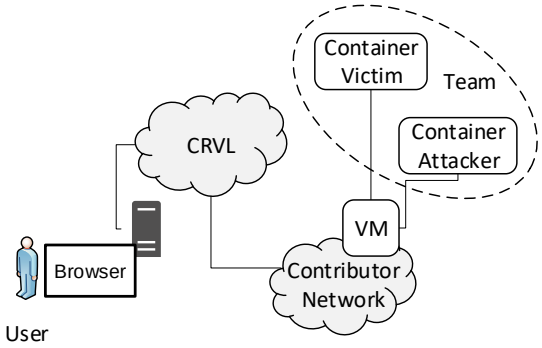

Figure 5.12: Team of Containers

When a contributor registers a VM to the CRVL system, the contributor needs to enter the access time of the VM on the registration page. This way when a VM is going down on schedule, the migration process could be executed to move the Teams to a new VM and redirect the participants to the new VM IP address. As shown in figure 5.12, a Team consists of Containers which are used to represent hosts (attacker or victim machine) in the exercise scenario. In the CRVL context, the live migration is executed for the participants (users) to continue working on the hands-on-exercise with nearly zero downtime. The participants do not need to start over the exercise from the beginning, but they can continue the exercise from the latest state of their activities at the old VM. In this case, the data that need to be moved from the migrating VM to the target VM are the modified files and the commands history of the Containers that being used by the users to do the exercises. Figure 5.13 shows the live migration scheme, where VM#1 is the migrating VM, VM#2 is the target VM.

VM live migration will be executed when the VM is shutting down normally on schedule. The Containers (Teams) being used by the users must be moved to other VMs before the shutting down schedule. The user should move to another VM before the original VM is shutting down. If the time needed to do one exercise is 45 minutes, the CRVL system stops creating a new Team in the VM starting at 45 minutes before the shutting down schedule. During these 45 minutes, the

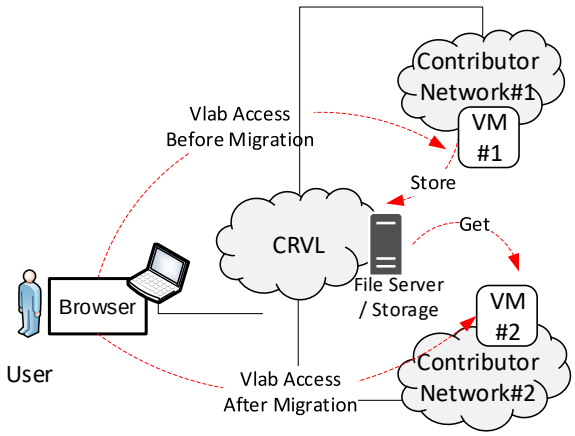

Figure 5.13: Live Migration Scheme

existing Team in the VM should be moved to another VM. In five minutes before the shutting down schedule, if some users are still accessing some teams running on the VM, the users will be warned that they need to go to another prepared VM to continue the exercise session. At the shutting down schedule, other VMs already have the latest state of the Containers from the shutting down VM and ready to serve the users.

Two approaches could be used to migrate the Containers to other VMs. The first approach is using "docker checkpoint and restore" to migrate the Team by copying the memory data, the disk data and CPU states to the target node. This approach could spend many resources especially the bandwidth usage because depending on the size of memory being used, each checkpoint could have a large file that needs to be transferred to the target node. For example, a MySQL Container checkpoint could create 117MB page map dump[74]. Docker checkpoint is saving the data from the memory and hard disk to a file. Docker checkpoint could not save only the different state of memory or hard disk. Docker checkpoint is using CRIU to freeze and dump a Container to a file, but CRIU could not freeze several applications running such as ettercap[1] and nmap[2].

The second approach is based on the participant activities in the Container. The participant activities are recorded and restore in the new Container. The

[1]https://www.ettercap-project.org/
[2]https://nmap.org/

participant shell commands history and modified files are needed to restore a Container on a new host. This way, we could save and transferred only the different state of the participant activities. By copying modified files and run the same shell commands on the new Container, we could restore the latest state of the user activities on it. This second approach could not recover the memory state of the Container, while the first approach could. We propose to use this second approach because, in the CRVL system, the participant does not need the same memory states to continue working on the hands-on exercises.

In the CRVL system, the Teams from a migrating VM could be migrated not only to one VM but to several VMs. The placement algorithm[102] is used to decide the migration destination of a Team based on the location of the participant that is using the Team. The Team consists of Containers and virtual network. The participant data that needs to be migrated is on the Container that was used by the participant to do the exercise. To get the data from Containers to the VM, docker Volumes could be used. Volumes create a directory in the VM that is shared to be used as a directory in the Containers. Containers could store persistent data to this shared directory to be able to use the same data when the Container is restarted.

To automatically transfer the Container's data to other VMs, we use a shared directory on a file server mounted as a network directory (remote file system) on the VM. The modified files and commands history could be stored in this network directory. When the Container is moved to another VM, the new Container in the new VM could mount the same network directory to get the latest state of the old Container. The network directory could be created using NFS[1] or SSHFS. Figure 5.14 shows the live migration sequential diagram, where the Directory in the CreateTeam function is the network directory.

5.5.3 Fault Recovery

VM Fault recovery is needed when a VM is suddenly shut down without notice and out of schedule. The CRVL system assumed that the VM is down when the WebSocket to the CRVL controller is disconnected. The CRVL system needs to

[1] https://help.ubuntu.com/lts/serverguide/network-file-system.html.en

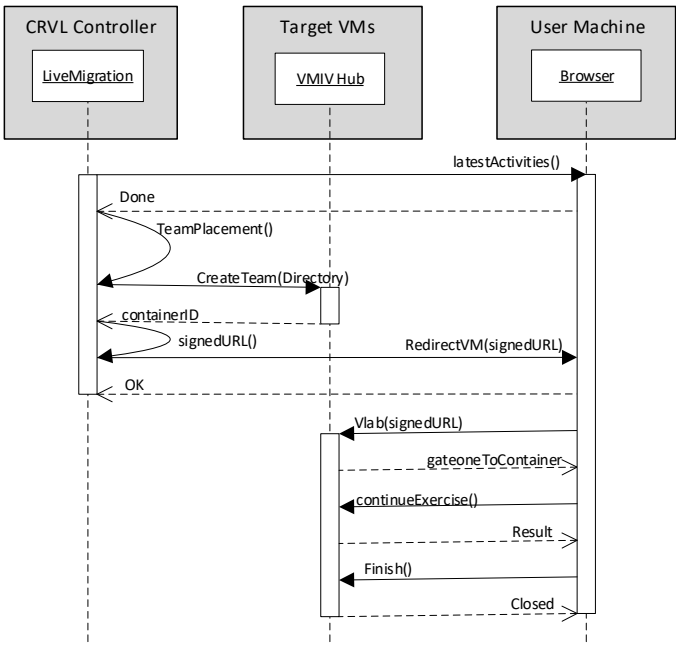

Figure 5.14: Live Migration Sequential Diagram.

do the Fault recovery process and inform the participants/users to get connected to another VM. In the CRVL system, the contributors able to disconnect the VM at any time. The VM could also be down because of technical problems such as host hardware problem or Internet connection problem. Similar to the live migration, in the VM Fault Recovery, we try to recover the Teams (containers) which was running on the fault VM, by re-creating the nearly the same team on another VM.

To be able to recover the Containers at the closest state to the latest state of the Containers, the Containers must be backed up periodically by the system. The data that needs to be backed up are modified files and shell command history. Similar to the live migration, VM fault recovery needs to create nearly the same Containers with the latest state of user activities on other VMs, and redirect users to these assigned VMs. At live migration, we could get the latest state of the Container just in time before the shutdown schedule, but at the VM fault recovery, the latest state is based on the latest backup before the downtime. The smaller the periodic backup time, the closer to the latest state. The smaller the

periodic time means more frequent, and this means more traffic to the backup server.

Fault recovery approach is similar to the live migration approach. The difference is that there will be no notification before a VM is down. We use figure 5.13 to show the scheme of the Fault Recovery approach, figure 5.15 to show the sequential diagram of the Fault Recovery approach. We create backup files by recording the user activities on the Container. We use the network directory to store the shell commands history and modified files to be able to get the latest up-to-date user activities.

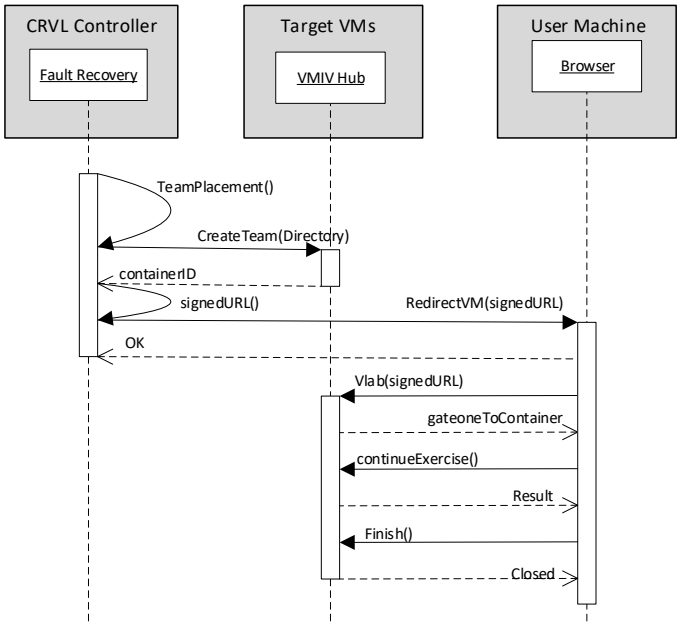

Figure 5.15: Fault Recovery Sequential Diagram.

When a VM is suddenly shut down, the CRVL controller will be notified because the WebSocket connection will also be down. CRVL controller executes the Team Placement algorithm for each user of the down VM to allocate them to other VMs. A user (participant) might get a different VM from another user. The CRVL controller sends a command to create a Team on the selected VM. The selected VM creates the Team and sends a Container ID to the CRVL controller. The CRVL controller creates a signedURL [101] and sends it to the user to redirect

the user to the selected VM. The user uses the signed URL to access the Container to continue the hands-on-exercise.

5.5.4 Evaluation

Every pre-built VM image should have all the needed application installed, such as an Integration Application (VMIV Hub), Gateone, Node.js, docker, Open vSwitch and iptables. The VMIV Hub needs to communicate with the CRVL controller for Live migration and Fault Recovery. We conducted several experiments to evaluate the functionality and performance of our approaches. Because Live Migration is similar to Fault Recovery, the experiment on Fault Recovery is also representing the Live Migration.

We implement the scheme in figure 5.13 to run a Fault Recovery experiment in a LAN where the File Server and the CRVL controller is a VM in OpenNebula private cloud. The fault (VM#1) and the target (VM#2) VMs are running on a VirtualBox in a workstation machine with Intel Core processor i5-4690 CPU @3.50 GHz and 8 GB of RAM. Each VM has 512 MB of RAM, one vCPU and 80 GB hard drive. The VM#1 was down when the user was in the middle of an exercise where some commands had already been entered and a file had been modified. In this experiment, there was only one target VM. We use SSHFS to mount a network directory on the file/storage server. We simulate 100 VMs on the crowd-resourcing platform. The maximum number of teams per VM is 40 Teams. There is a NAT to get into the VM which is provided using iptables.

We evaluate our approach by measuring the fault recovery time of a different number of users/teams e.i. 1, 5, 10, 20, 30 and 40. One user has one team. For each number of users, we measured the recovery time for ten times and calculated the average of the measurement results. The experiment results of average recovery time for a different number of teams is shown in figure 5.16.

The duration of fault recovery is measured starting from a disconnected VM until all the Teams for the users are created. The user needs to respond by clicking the new signed URL to get connected to the Team in the new VM. From the results in figure 5.16, we can see that the recovery time for one user is quite small compared to more than one user. The recovery time for more than one user is near to the multiplication of the number of users with the time for one user,

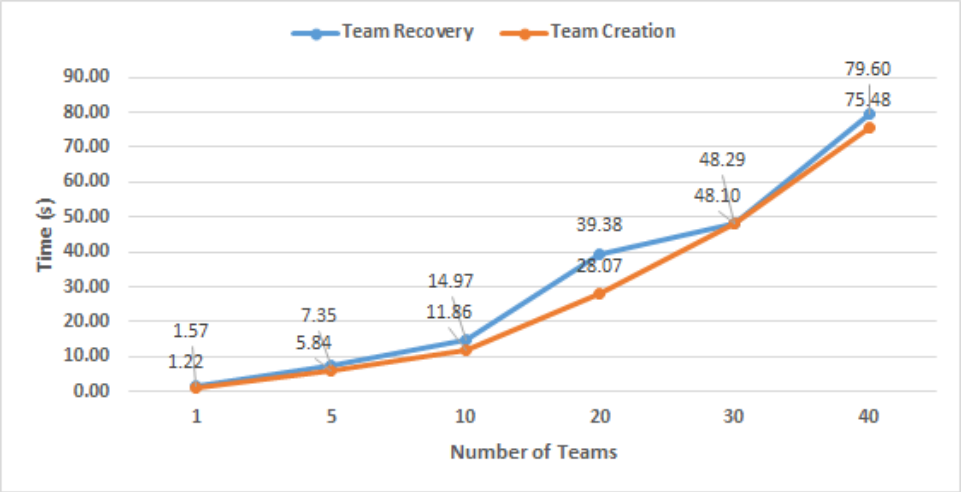

Figure 5.16: Fault Recovery & Team Creation Time.

because there was only one VM to recover the fault VM. All the recovered teams were created in one VM. The recovery time is representing the maximum time a user needs to wait before the Team is ready. If the Teams could be distributed to more than one VM, the recovery time for more than one user could be faster. For example, if there were 4 VMs as the target VMs to recover the fault VM, the recovery time for 40 teams is equal to the recovery time for ten teams, because each VM only needs to create ten teams at the same time. The number of VMs depends on the team placement algorithm [102] which is based on the geographical location and the VM Load.

The recovery time for 40 teams is around 79.80 seconds, which is still acceptable for a user to wait to get the latest state available, compare to starting over from the beginning. If we could distribute the teams to ten VMs, the recovery time would be around 7.98 seconds. As we can see from figure 5.15, there are several functions to recover the fault VM. Based on our experiment the function that took the longest time is CreateTeam() function that could take more than 1 second to create one team. This function starts containers, configure OpenvSwitch and iptables, and mount the network directory.

Besides the team recovery time, figure 5.16 shows the Team creation time which is the time needed to create a Team per user request. As we can see, they are not much different, because the difference is only on the shell commands that

must be executed during fault recovery. Shell commands history is representing the user activities on the shell. Shell commands execution time could be very small in milliseconds. In this experiment, the shell commands are only about configuration and modification of files. There is no installation, and the user had not executed the command to run the attacking exercise.

The overhead of our approach is mostly on the traffic from the VM to the file server, because of the network directory. Another overhead is that we need to provide a file server on the CRVL system, but it could be provided at the same machine as the CRVL controller.

5.5.5 Discussion

The widely used strategy to provide a fault recovery system is based on the notion of redundancy[49]. The redundancy system provides a back-up for the primary running system. The back-up system could be running together with the primary system as a load balancing system, and it could also be available only when the primary system is crashed. To provide redundancy, the primary system components are duplicated using additional software, hardware, and network resources.

Our approach is using network directory (remote file system) to store the modified files, and the shell commands history, which is used to generate a new Container (Team) on the target VM with the latest state of the old Container. This way, the duplicated components are only the modified files and the shell command history. Other fault recovery approaches create duplicate VM that must be done from the VM Host such as VM snapshot and images instance mapping [73]. Besides these approaches create large files, they are not applicable in the CRVL system, because the CRVL system does not have access to the VM host. Docker checkpoint and restore could be used to duplicate the Container, but it needs more resources because it duplicates the disk files, memory, and CPU.

It is easier to move or distribute the modified files among other VMs using the network directory. We could run Teams in different VMs to recover the failed Teams on the failed VM. The placement algorithm could be used to decide where to run each Team to be able to find the best place to run the Team. Network directory is depending on the network connection quality. When the network

connection is not reliable, the network directory could be lost, and the latest state of Containers could not be reached.

Our Live migration and fault recovery approach could also be implemented in other similar use cases such as microservices on Containers or Containers on P2P cloud when the memory states (data) do not need to be copied to the target VM. If the memory states need to be recovered, we need to do more research to find a way to copy the memory states to the network directory and to restore the states on another VM. It should be only the small part of memory data that need to be copied to the target VM, to be able to save disk space and bandwidth.

5.6 Chapter Summary

Scalability can be increased by providing more resources from the crowd. The crowd could contribute their spare resources in the form of VMs. In this chapter, we propose several approaches about the architecture, the Team placement, the VM Integration, the VM live migration, and the VM fault recovery. We propose an architecture based on the private cloud Tele-Lab architecture. In our architecture, a Team consists of containers and running on a VM.

This chapter describes our approaches to automatically integrate a VM and recover from a fault VM in CRVL system. The VM integrity verification is described in section 6.1 as a part of the Integration approach. The Fault Recovery approach is using the network directory to store the Container's shell commands history and modified files. When a VM is failed, the data on the network directory is still accessible via the network. This way the Container could be recovered by creating a new Container using the same network directory on another VM. The same shell commands are executed on the new Container to get the latest state of the user activities.

We do not need to back up the Container memory, because the Container is used for user's exercises, not to run an online service. For future work, the same fault recovery approach should be evaluated to back up the critical data stored in memory. The critical data should be enough to recover the latest state of the Container. The possibility of using services logs to recover the service should be investigated. This way, we could use our method on an online service.

The duration time of a VM fault recovery is depending on the number of target VMs that could be used to run the Teams that was running on the fault VM. When the throughput to each VM is the same, the more the number of target VMs, the smaller the time to recover.

In CRVL, everybody including a bad guy can contribute his spare resources into the system. The bad guys as a host admin have physical access to the resources. They could do some attacks on confidentiality such as tampering with the OS image and the data in the RAM. On the next chapter, we proposed three approaches to strengthen the VM from some attacks.

Chapter 6

Secure Virtual Machine on Untrusted Host Machine

Crowd-Resourcing Virtual Laboratory (CRVL) gather resources from the crowd mostly in the form of Virtual Machine (VM). Everyone that has spare resources could share their resources to contribute to the CRVL system. Everyone means including the bad guy. The host of the VM could be compromised, and the host admin could be malicious. The malicious admin could apply an insider attack to get the sensitive data from the VM where a team is running. This is the problem of privacy and confidentiality. We need to secure the VM to enhance the confidentiality of the sensitive data inside the VM. The sensitive data could be user credentials, Intellectual Property such as a virtual laboratory or learning materials.

In this chapter, we describe three methods to increase the data confidentiality in a VM running on Untrusted Host Machine. The first method is to remotely verify the integrity of the Virtual Machine without using any specific hardware such as the Trusted Platform Module (TPM) chip. The second method is moving sensitive data against live memory dumping, Spectre and Meltdown. The sensitive data is always moving from one memory location to another memory location. The third method is to monitor all the activities on the host of the VM using a monitoring agent on the host. The agent sends the monitoring result to an independent third party, which is trustworthy by the host owner and the CRVL system. These three methods could also be used in a public cloud. The

threat models are also described in this chapter.

6.1 Virtual Machine Integrity Verification

In cloud computing, users can use their operating system (OS) image to run a virtual machine (VM) on a remote host. The virtual machine OS is started by the user using some interfaces provided by a cloud provider in a public or private cloud. In peer to peer cloud, the VM is started by the host admin. After the VM is running, the user could get remote access to the VM to install, configure, and run services. For the security reasons, the user needs to verify the integrity of the running VM, because a malicious host admin could modify the image or even replace the image with a similar image, to be able to get sensitive data from the VM. We propose an approach to verify the integrity of a running VM on a remote host, without using any specific hardware such as Trusted Platform Module (TPM). Our approach is implemented on a Linux platform where the kernel files (vmlinuz and initrd) could be replaced with new files, while the VM is running. kexec is used to reboot the VM with the new kernel files. The new kernel has secret codes that will be used to verify whether the VM was started using the new kernel files. The new kernel is used to further measuring the integrity of the running VM.

6.1.1 Motivation

CRVL is used to increase the number of resources where a person or a company can contribute, by sharing their unused resources to be integrated into the Virtual Laboratory system. In CRVL system, a virtual machine (VM) could be running on an untrusted contributor's host machine. This scheme is similar to peer-to-peer (P2P) cloud or social cloud, where everyone with some spare resources could become a contributor. As the owner of the host machine, the contributor can do some malicious activities to get confidential data from a running VM. The malicious host admin could inject a malicious program into the OS image that will be executed whenever the OS is started. This problem is also applicable in the public cloud, where a VM is running on a provider host machine. A public cloud provider is bound by law and an agreement that they will keep the customers'

data private and secured. The cloud admin is not allowed to see or copy the customers' data. Some people still do not trust the provider that they do not want to put their confidential data in the cloud.

The OS image could be provided by a cloud provider in a public cloud or by a host admin in a P2P cloud. The OS image could also be provided by a user. The user could create an OS Image based on the user's needs and uploads the image to the remote host. In CRVL, the OS image is a pre-built (pre-installed) image. After being uploaded, the user could start a VM base on the OS Image, using an interface provided by the public cloud provider. In the P2P cloud, the VM is usually started by the host admin. The host admin in a public cloud or P2P cloud is able to modify the OS image that has been uploaded. When a user sends a request to start a VM based on the OS Image, the host admin could manage to start the VM from another image or a modified image.

The objective of this research is to find a solution in verifying the integrity of a running VM before it is integrated into the system of the CRVL or P2P cloud. The integrity of the files on the hard drive does not guarantee the integrity of running services, because the running services could be started from some hidden files or some files that were removed after the service is running. Some methods might be able to verify the integrity of a running VM, such as remote attestation, operating system fingerprinting, VM introspection, and memory based VM integrity verification. These existing methods are not suitable for mitigating the problem of verifying the integrity of a running VM in a Crowd-Resourcing Virtual Laboratory, because the VM might not have particular hardware or the user does not have access to the host machine.

We propose a new method that uses kexec[1] to reboot the VM from known good kernel files (vmlinuz and initrd). These files are sent to the running VM as part of the integration process in the CRVL system, where the integrity of the running VM needs to be verified. These files have secret codes that will be extracted into the kernel when they are used to boot the kernel. The secret codes have an expiry time. kexec is installed just before it is needed to reboot the VM. We assumed that the kexec has not been compromised. After the VM was rebooted, the kernel is considered as trustable to verify the integrity of the system files in the VM.

[1] http://man7.org/Linux/man-pages/man8/kexec.8.html

In the boot process, before the init process of the userspace, the kernel loads a module that has a function to get the hash checksum of all the system files. The secret codes and the hash checksum are sent to the VLab (CRVL) server. The VLabServer verifies the values and the expiry time of the secret codes and the hash checksum. If they are not valid, the server will reject the VM from being integrated into the CRVL system.

Our proposed integrity verifying mechanism was working well in our experiment. We run four experiment scenarios to verify the functionality of our approach. It has prevented the kernel from loading the malicious program and rejected a modified VM. We measured the performance by measuring the time as the overhead, and the result is still acceptable as our approach is a part of the VM provisioning time. Our experiment used Linux as the operating system of the VM because the Crowd-Resourcing Virtual Laboratory is using Linux VM to run containers. Our approach is designed based on Linux systems.

6.1.2 Related Work

6.1.2.1 OS Finger Printing

Operating system (OS) fingerprinting is the process of identifying which operating system is running on a particular device. The device could be a remote machine or a local virtual machine in a cloud computing infrastructure. For the remote machine, the identification process is done by analyzing some parameters such as flags in the packet header and data in the packets, and a device sends to the network. There are two types of remote machine OS fingerprinting, i.e. passive and active fingerprinting. Passive fingerprinting identifies the OS by sniffing and analyze the packet sends to the network, and active fingerprinting actively send some requests to the remote device and analyze the responses. Nmap[1] and Xprobe2[2] can be used to do active OS fingerprinting. This kind of tools suffers some problems, where they could be failing because of the remote machine disable network services by closing all the TCP/UDP ports and drops all ICMP packets.

[1] https://nmap.org/
[2] https://Linux.die.net/man/1/xprobe2

In a cloud computing infrastructure, the cloud admin needs to find out the kernel version of each running VM automatically, to secure the cloud infrastructure. The cloud admin needs to know the exact kernel version and updates to be able to configure the security parameters. Beside for security purpose, precise fingerprinting of an operating system is also needed for VM management applications in the cloud, such as VM introspection, kernel update, kernel dump analysis, and memory forensics. Some works have already been done to identify a kernel version trough memory dump, CPU Registers, and kernel images.

Lin et al.[62] propose SigGraph, which relies on identifying kernel data structures in memory to identify the kernel version, since the data structure varies across operating systems. SigGraph is a framework that systematically generates signatures for data structures in an OS kernel. SigGraph-based signatures can be used for brute force scanning to find out whether an instance of the corresponding data structure exists in the memory. Gu et al.[34] propose OS-Sommelier, which analyses physical memory dump file for kernel version identification. It searches the entire memory snapshot to identify the cluster, which contains core-kernel code. It generates the signatures, which are the cryptographic hash of the kernel pages. The signatures will be compared with the known signatures to identify the kernel version. Roussev et al.[86] propose a kernel fingerprinting, which uses the content of the kernel images on disk to build the signatures. They claim that their approach can distinguish among incremental kernel version updates. Kernel identification is delivered by looking for the presence of known kernel content in a RAM snapshot. They use a matching tool called sdhash, to extract kernel fingerprints.

This OS fingerprinting method could get the OS version of a running VM, but it could not solve the problem of integrity since it does not verify the integrity of the system files.

6.1.2.2 Virtual Machine Introspection

Virtual Machine (VM) Introspection is a technique for monitoring and analyzing the running state of a VM, from the hypervisor perspective. Garfinkel and Rosenblum[31] introduced this term in 2003, by proposing an intruder detection system (IDS) run on a host but pulled the IDS outside of the host. They used

Virtual Machine Monitoring to isolate the IDS from the monitored host. The VM Introspection has been used for various purposes such as digital forensics, anti-Malware, VM sizing, and migration. Suneja et al.[110] explored the existing VM Introspection techniques and created a taxonomy based on the VM Introspection operational principles. Using the taxonomy, they explored the trade-offs of the VM Introspection techniques qualitatively and quantitatively.

VM Introspection cannot be used to solve the problem of integrity in the CRVL system because it has to be done from the host while the remote VM user does not have access to the host.

6.1.2.3 Virtual Machine Integrity

VM integrity (VMI) is monitored and analyzed remotely at the verifier side. The VMI verification is conducted by doing remote attestation or Remote Integrity verification. Virtual machine integrity has become a popular topic in cloud trust. Some works have already been done in verifying the integrity of a VM. Yu et al.[131] implemented TCG (Trusted Computing Group) remote attestation under the assumption that the TPM (Trusted Platform Module) and hypervisor are secure and the privileged domain0 may be malicious. Perez et al.[81] proposed virtualized TPM that enables trusted computing for an unlimited number of virtual machines on a single hardware platform. This virtualized TPM needs a TPM chip on the hardware platform.

Intel developed Software Guard Extensions that create hardware-assisted trusted execution environment. It creates CPU-hardened "enclaves" or protected areas of execution in memory for the application that run sensitive code and data with the enclave data is written to disk encrypted and checked for integrity [45]. Using this Intel technology, software vendors are facing a problem of generating or obtaining the secrets in the first place. Malware could be listening to the I/O of the client machine. To mitigate this problem, Intel SGX has an advanced Remote Attestation feature, where the client must confirm its identity, verifies its integrity, and proves that it is using Intel SGX [46][51]. Intel seems to be able to solve the problem of VM integrity, but Swami [111] claimed that the Intel SGX Remote Attestation is not sufficient. He found some problems in the protocol

design that sub-computation of the protocol can be simulated outside of the enclave. Schwarz et al.[95] performed a cache side-channel attack on a collocated SGX enclave and can extract the RSA private key from other enclaves.

The existing VM integrity verification approaches using specific hardware such as TPM and SGX, are not suitable to verify the VM integrity in the Crowd-Resourcing Virtual laboratory, because the contributor might not have the needed hardware. We are proposing a new approach that does not use any specific hardware. Our approach does not verify the integrity of the BIOS, and we use kexec as the root of trust to reboot the VM using the fresh downloaded Linux kernel files such as initrd and vmlinuz.

6.1.3 Threat Model

In Crowd-Resourcing Virtual Laboratory (CRVL), a VM is started with a pre-built image that was ready to be used for a particular scenario. The VM could be running on an untrusted host machine, where the host admin could be malicious. A host admin has full access and control of all the resources in the host. Once a guest VM is already running, the host admin could not get into the VM without a proper user account. The malicious host admin could access the VM's files that are stored on the hard drive and dump the VM memory to get sensitive data out of it. The malicious host admin could also modify the image of the VM to add users, create a backdoor, etc. When a VM is started using the modified image, the host admin could get access to the VM using the added user or the backdoor. In this paper, we focus on solving the problem of verifying the integrity of a running VM on a remote host in the CRVL, that could be run from a compromised pre-built image.

Running a VM with a particular pre-built image on an untrusted host machine could have some threats regarding the privacy and confidentiality of the data inside the VM. Confidential information such as a user's credentials, confidential documents, and intellectual property, must be kept safe from being exposed to unauthorized parties. The VM should be run from a pre-built image that has already been proven to be secure. This image could be created by a user and uploaded to the remote host machine. When a user wants to run a VM using the image, the host admin can interfere with running another similar image that

perhaps has vulnerabilities. The malicious host admin can add a line of code in a script to run another (unknown) image instead of the one that requested by the user. This unknown image could have the same users and services that was copied from the real image. This image could have some vulnerabilities or backdoors that could lead to some attacks.

Another threat is that the malicious host admin could modify the pre-built image[1] and add a rootkit or a backdoor to get confidential data when the VM is running based on the modified image. The malicious host admin could modify some services inside the running VM, by embedding a script inside a service starter in the image, which will be run at the startup or after the VM is running using crontab. After the modified services are running, the malicious host admin could remove all the modified files and restore the original files while the modified service is still running. This way, verifying the integrity of the files does not mean verifying the integrity of the running services. Tools such as AIDE[2] and Tripwire[3] could not be used to detect the malicious or modified services, because the modification is done offline, the modified services are started before the tools are started, and the modified files are replaced with the original files after the modified services are running. The user could try to verify the integrity of the running services by monitoring their behaviour, but the malicious host admin could design some services which are hidden from the user. The user could also try to memory dump the services (processes) one by one, and analyze the dump files to measure the integrity. However, this is not easy to be done as the dump file of a process is always changing.

To be able to modify the pre-built image undetectable, the malicious host admin should do the modification, execute the modification, and remove the traces before the integrity verification tools could be executed. This could be done during the bootup process before the init program is called. The malicious host admin will try to embed the malicious code as soon as possible after all the functions needed has been ready to be used. This means that the malicious code will be embedded in the kernel space. The kernel is loaded from a compressed

[1] https://docs.openstack.org/image-guide/modify-images.html
[2] http://aide.sourceforge.net/
[3] https://www.tripwire.com/

kernel image file (vmlinuz) and an initrd.img file. The malicious host admin could modify these files and save the modified version with new names, and use these new files to run the VM by modifying the grub.cfg to boot from the modified files. After the kernel is loaded, the grub.cfg will be restored to the original version, and the modified files will be removed.

6.1.4 Verification Method

An integrity measurement architecture (IMA) is using TPM as the root of trust to store artefacts of a platform authentication. Using the TPM as the root of trust, the system could verify the integrity of all the system files before execution during the boot processes. Our approach does not use any specific hardware such as TPM as the root of trust, because the contributor might not have it and might not allow it to be used by the user of the shared VM. We propose an approach that could verify the integrity of a running VM without TPM, by copying several new kernel files into the running VM and reboot the VM using the new kernel files. The new running kernel is used as the root of trust. The kernel files have secret codes that will be extracted and stored in memory or log messages when the new kernel is running. These secret codes are used to verify whether the VM was started using the new kernel files.

Our approach works on a Linux environment because the Linux kernel files (vmlinuz and initramfs) could be replaced by new kernel files, on a running Linux OS. The VM is started using a pre-built OS image[1][2][3], to be able to have the same files and configurations as the one in the VLab server. This way the hash checksum (hashsum) could be used to verify the integrity of the VM.

Figure 6.1 shows the general architecture of the VM integrity verification (VMIV). On the server side (Vlab Server), there is an Integration service to verify the running VM who wants to be integrated into the Crowd-Resourcing Virtual Laboratory. This integration service communicates with VMIV hub on the running VM. VMIV hub executes the sequence of the Integration process,

[1] https://docs.openstack.org/image-guide/create-images-manually.html

[2] http://www.theLinuxdaily.com/2010/02/how-to-setup-a-pre-built-Virtualbox-guest-image-tutorialguide/

[3] https://wiki.ubuntu.com/Kernel/BuildYourOwnKernel

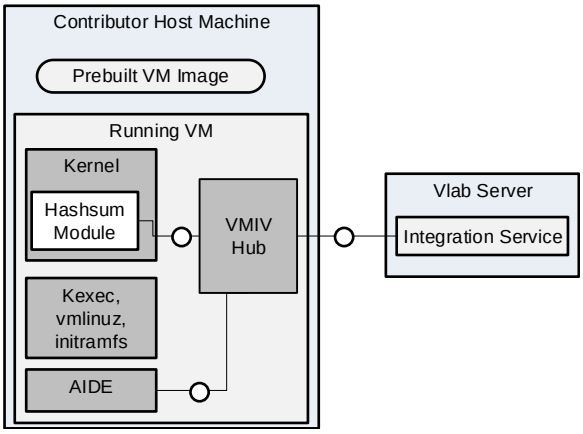

Figure 6.1: VMIV Architecture.

including the integrity verification process. VMIV hub uses kexec to reboot the VM and collects data from Integrity Verification (Hashsum) Module and AIDE as the monitoring tool. AIDE is pre-installed on the pre-built image. The pre-built image was downloaded from the Vlab Server.

Our model consists of two major components: enforcing and verifying. The enforcing is to create a clean base system that we use as a root of trust. The verifying is to verify whether the running VM has been tampered and whether some modifications had been made without known by the user of the running VM. As a part of enforcing, the kernel must be configured to only load modules that have valid signatures, to prevent an unauthorized module from being inserted into the kernel.

To create a clean kernel environment, some integrity files (vmlinuz, initramfs, hashsum module and other files) are copied to the running VM from the virtual laboratory (Vlab) server. These files have an expiry time (120 seconds) that within the expiry time, the files must be used to run a VM and the secret codes of the files must be sent to the Vlab server. The expiry time is used to limit the amount of time that an attacker has, in modifying the integrity files. vmlinuz, initramfs and hashsum module have secret codes that are different between each other. A secret code is inserted into vmlinuz via one of the Linux kernel source code files, e.g. main.c. A secret code in initramfs could be generated and inserted on the fly. A secret code in hashsum module is inserted via the source code file.

Some number of vmlinuz, initramfs, hashsum modules are provided and ready to be used on the Vlab server. The combination of vmlinuz, initramfs, and hashsum module are selected randomly to prevent replay attacks where a rogue contributor sends the already used secret codes to the Vlab Server. The secret codes of vmlinuz and initramfs will be extracted and stored in the memory or a log message (dmesg) if the kernel files are used to run a VM. The secret code of hashsum module will be used as a nonce in the hashsum function, to prevent the use of a fake (modified) hashsum module, because a fake hashsum module has a different nonce.

The running VM is rebooted using kexec that could reboot the VM into another kernel from the currently running kernel without performing hardware initialization that should be done by the BIOS (Basic Input/Output System). kexec is installed just before the VM is rebooted. We assumed that the kexec is not compromised. kexec uses kernel as a boot loader, and it does not use a conventional boot loader such as Grub or LILO. Figure 6.2 shows the Linux boot process and rebooting process using kexec. After the VM was rebooted, the secret codes of the vmlinuz and initramfs are stored in the memory. The secret codes will be sent to the Vlab server together with the hashsum code.

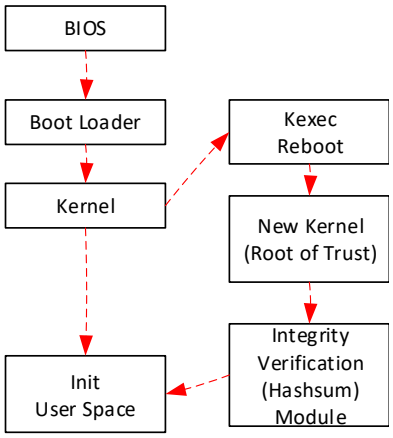

Figure 6.2: Linux Boot Process.

After the new kernel has been loaded, the integrity verification (Hash checksum) module is loaded. This verification function summarizes the hash values of all the system files using a hashsum function such as md5sum or sha1sum. The

hashsum module should be loaded right after the root file system is mounted and before the init process of userspace, to be able to verify the integrity of all system files before being executed.

The hashsum module is built using static libraries to stop the module from using a compromised library, and it is downloaded at the same time as the new kernel files. The hashsum result (HashCode) together with the secret codes are sent to the Vlab Integration service to be verified by comparing the HashCode and the secret codes with the ones stored in the server. If the HashCodes and the secret codes were the same, and the kernel files were not expired, the VM is allowed to be integrated as one of the resources of the virtual laboratory system. After the VM is integrated into the CRVL, AIDE is used to monitor all the files in the VM. The AIDE check is executed periodically, and the result is sent to the Vlab server.

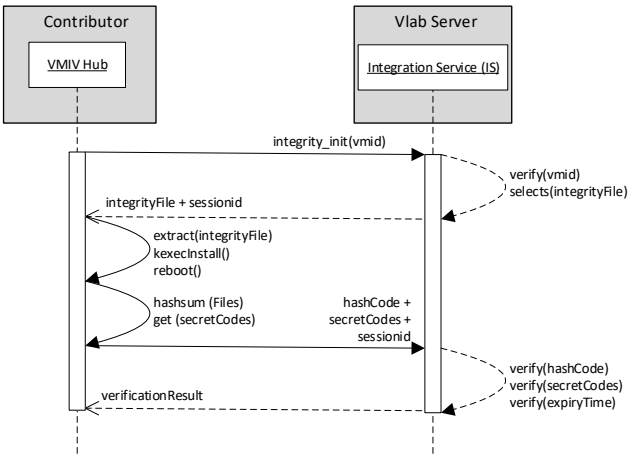

Figure 6.3: Integrity Verification Sequence.

Figure 6.3 shows the sequence of enforcing and verifying the integrity of a running VM. The running VM sends a message to initiate the integrity verification process, which consists of the VM Identifier (vmid). The Integration Service (IS) on the Vlab server verifies the vmid. If the vmid was valid, the IS randomly selects an Integrity file (integrityFile) that has never been sent to the contributor running VM and sends it to the contributor (VMIV Hub) together with a sessionid. The integrityFile is a compressed file consists of vmlinuz, initrd, hashsum module,

and some files needed to load the hashsum module during boot up. A sessionid is used to maintain the verification process after reboot. The VMIV hub receives and extracts the integrityFile, installs kexec, and uses kexec to reboot the VM using the new kernel files.

During boot time, a hash code (hashCode) is generated by the hashsum function. The hashsum function is executed by loading a module into the kernel. The hashCode together with the secret codes (secretCodes) from the kernel are sent to the IS, which verifies the secretCodes, the hashCode and the expiration status (expiryTime) of the kernel files. The IS calculates the time between the sending of the kernel files (integrityFile) to the contributor and the receiving of the secretCodes, to verify whether the kernel is already expired. The hashCode and the secretCodes are compared with the one stored in the database. If the hashCode and the secretCodes are the same and the kernel is not expired, the VM integrity is valid and allowed to be used. The communication between a Contributor (VMIV Hub) and the Vlab server (IS) is encrypted, but to make it simple, we do not show it in the Integrity verification sequence. We use WebSocket Secure (WSS) for the communication between the VM and the Vlab server. If the WebSocket was disconnected, the disconnected VM has to start the verification process from the beginning.

6.1.5 Evaluation

We conducted several experiments to evaluate the functionality and performance of our approach. Figure 6.4 shows the architecture of the experiment. VM#1 is the verifier which will verify the integrity of the shared VM on the workstation. VM#1 is running on an OpenNebula in the Tele-Lab server, and the shared VM is running on a VirtualBox in a workstation machine with Intel Core processor i5-4690 CPU @3.50 GHz and 8 GB of RAM. The shared VM has 2 GB of RAM, one vCPU and 80 GB hard drive. The shared VM is started using a modified pre-installed OS image.

The pre-installed OS image is created using Linux Ubuntu 18.04 Bionic Beaver as the base image. The HashCode of the files inside the image is stored in a VM#1. Hash sum module (hashsum.ko), "/etc/modules", "/lib/modules/4.15.18-custom/modules.dep" and "/lib/modules/4.15.18-custom/modules.dep.bin" and

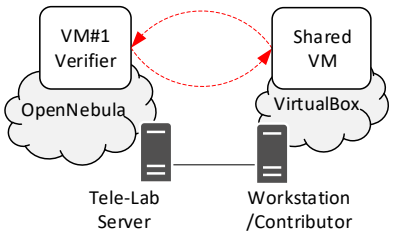

Figure 6.4: Experiment Architecture.

all the files in "/boot" folder, are compressed into one file and uploaded to the VM#1. Hashsum.ko is the module that will be loaded during boot time to hash sum the system files in the VM. The system files are the files that stored in these folders: "/bin", "/boot", "/etc", "/lib", "/sbin", "/usr/bin", "/usr/sbin", and "/usr/lib". The pre-installed image is copied to the workstation to run a VM using VirtualBox. The pre-installed image is modified by running a VM using the pre-installed image, modify several files inside the running VM, and create a modified pre-installed image from the already modified Virtual Machine. The modified pre-installed image is used to run a VM which is going to be verified by the VM#1. After the shared VM is running, we implement our approach to verify the Integrity of the running VM.

To evaluate the functionality of our approach, we modified or added some files on the image, run a VM using the image and verify the integrity of the running VM using our approach. We conducted four experiment scenarios, where the modified files are different. In the first experiment, we add a rootkit module called nurupo[1] into initramfs, by modifying "/etc/initramfs-tools/modules" file and store the rootkit.ko file into "/lib/modules/4.15.18-custom/kernel/drivers" folder. Linux command "update-initramfs -u" is executed to activate the new configuration, and the VM needs to be restarted. The VM is restarted, but the rootkit module could not be loaded because it did not pass the signature verification.

The second experiment is conducted by manually adding a nurupo rootkit module into "/etc/modules" and add the rootkit.ko file to the "/lib/modules/4.15.18-custom/kernel/drivers" folder. Linux command "depmod" needs to be executed

[1]https://github.com/nurupo/rootkit

to find all the dependencies of the module. The VM is restarted, but the rootkit module could not be loaded because it did not pass the signature verification. Loading the rootkit module into the kernel after boot process was also failed because of the unverified module.

In the third experiment, we add a simple backdoor[1] into the image that will be executed at boot time. The backdoor could be running, and the shell is shown in the attacker machine. We implement our approach to the running VM, and the hashsum of the running VM was not the same with the one stored in the server, which means the Integrity Verification of the running VM is failed. The fourth experiment is conducted by manually modifying a library of the image. After the VM is run using the modified image, our approach could find out that the Integrity of the running VM was false and the integrity verification was failed.

To evaluate the performance of our approach, we measure the overheads needed to implement our approach. We measure the needed time and memory space in implementing our approach on the VM side only. The time overhead can be obtained by measuring the download time of kernel files (t_d), the time of rebooting and hashing the system files(t_rh), the communication time between the VM and the server(t_c) including the verification time on the Vlab server. We could not separate the measuring time of reboot and hashing, because the module of hashing is loaded during reboot. Using kexec, the average of rebooting and hashing time is 1 minute 28 seconds. Without the hash module, the rebooting time is 29 seconds. The download time of kernel files is 2 seconds, where the size of the compressed file is 113 MB. The communication and verification time is 4 seconds. Total additional time in implementing our approach is 1 minute 34 seconds. This additional time is not significant and can be accepted for VM provisioning in the CRVL or P2P cloud.

The memory (RAM) space overhead is measured by calculating the amount of memory is being used by additional process or module that resident in memory. The Hashsum module and VMIV hub are resident in the memory. In this experiment, the hashsum module called a shell script to get the HashCode of the system files. The memory space used by the hashsum module is 16.3 KB. The VMIV hub is a part of VM Integration service in the CRVL which was built using

[1]https://github.com/buckyroberts/Turtle

python. The size of the VMIV hub is 19.1 KB. This memory space overhead is quite small that it would not affect the performance.

The root of trust in our approach is in the fresh installation of kexec, which is used to reboot the VM using the known good kernel files downloaded from the server. Our approach does not verify the integrity of bootloader and BIOS. The BIOS and the bootloader could be infected by a malicious program such as keylogger and rootkit. Our approach could not detect these malicious programs while they do not change any system files in the hard drive. Brossard[14] created a hardware backdoor (Rakshasa) proof-of-concept that replaces the BIOS and compromises the OS at boot time by patching the kernel on the fly without any traces on the hard drive. By rebooting the OS using kexec and new kernel files, the compromised OS could be replaced with a clean OS, because kexec does not use the BIOS and the boot loader to reboot the OS. We assumed that a malicious program does not compromise the fresh kexec, but in reality, there is still a possibility that the kexec is compromised right after the installation. To enhance our approach, an anti-virus or anti-rootkit such as chkrootkit could be used to detect any malicious program from memory (RAM) before kexec is installed.

There are several Master Boot Record (MBR) or boot loader rootkits (bootkits) such as stoned[54] for windows and kitgen[1] for Linux. The stoned bootkit needs to install a driver or a malware on the OS that will change the hashsum of the system files in the hard drive. The kitgen is designed to infect the initrd file, which will also change the hashsum of the system files and using our approach, the infected initrd file will be overwritten by a clean initrd from the Vlab server. Our approach could detect these bootkits because they modify the system files. We are not aware of another type of bootkit, but as long as the bootkit modifies something on the system files, we are sure that our approach is able to verify VM integrity and will reject the VM from being used in the virtual laboratory. To enhance our approach against bootkit, we could also verify the integrity of the MBR by doing hashsum to the MBR. The Linux command to hashsum the MBR is shown in Listing 6.1.

[1] https://github.com/chesteroni/kitgen/

```
1 dd if=/dev/sda bs=1024 count=1 | hexdump -C | sha1sum
```

Listing 6.1: Hashsum MBR Command

The secret code in the kernel files and the expiry time of the kernel files could guarantee that the running kernel is started using the good known kernel files. With enough time, the attacker could extract the secret code from the kernel files, but it will much longer than the expiry time. The attacker could use this secret code for the next verification. The secret codes of the next kernel files should be different every time the verification is being processed. There must be many kernel files with different secret codes stored in the server. The verification application must randomly select which one to be sent to a VM. The secret codes should not be the same as the previous ten secret codes that were sent to the same VM. The secret codes could be obfuscated to make it harder to be extracted from the kernel files.

A module must pass the signature verification to be able to be loaded into the OS. The public key to verify the signature is stored in a keyring in a file. The attacker could try to add a public key to the keyring to be able to load a new module, but the additional public key must be signed by the key that already in the keyring. To make it more secure, the OS must be compiled with a configuration that does not allow to add an extra public key.

A critical point in our approach is that the hashsum module must be loaded right after the root file system is mounted and before the init process of userspace. This is to make sure that all the processes or services are started from files that are already calculated in the hashsum function. An attacker could modify the OS image, to load the hashsum module after running the malicious program on the userspace and restore all the files to the original version. This way, the malicious program could not be detected. However, to do this, the attacker needs Linux commands, e.g. insmod and modprobe. These commands need privilege access using sudo. We could prevent the user account from using these commands by editing the sudoers file. If the attacker modifies the sudoers file to be able to load the hashsum module, the hashsum result will not be matched with the one in the Vlab server, because the sudoers file has been changed.

Encryption could also be used to verify the integrity of a running VM by preventing it from being modified by an attacker. To run the VM, the image needs to be decrypted, and the user needs to enter the secret key when the VM is booting up. For a VM on a remote host machine, the encryption key must be entered by the host admin, because the Vlab admin does not have access to the VM console while booting. In another case where the Vlab admin can have a console during boot, the malicious host admin can still use the Evil Maid Attack [89] to put a keylogger on MBR of an encrypted hard disk to get the password or encryption key. He can modify the VM image and hook up an evil maid program into the encryption software bootloader in the MBR.

6.2 Moving Sensitive Data

The emergence of cloud computing allows users to easily host their Virtual Machines with no up-front investment and the guarantee of always available anytime anywhere. However, with the Virtual Machine (VM) is hosted outside of the user's premise, the user loses the physical control of the VM as it could be running on untrusted host machines in the cloud. Malicious host administrator could launch live memory dumping, Spectre, or Meltdown attacks in order to extract sensitive information from the VM's memory, e.g. passwords or cryptographic keys of applications running in the VM. In this paper, inspired by the moving target defense (MTD) scheme, we propose a novel approach to increase the security of application's sensitive data in the VM by continuously moving the sensitive data among several memory allocations (blocks) in Random Access Memory (RAM). A movement function is added into the application source code in order for the function to be running concurrently with the application's main function. Our approach could reduce the possibility of VM's sensitive data in the memory to be leaked into a memory dump file by 25% and secure the sensitive data from Spectre and Meltdown attacks. Our approach's overhead depends on the number and the size of the sensitive data.

6.2.1 Motivation

Cloud computing offers Infrastructure-as-a-Service (IaaS) service model where users could host their Virtual Machines (VMs) with a pay-as-you-go pricing model, a rapid resources allocation and de-allocation on demand, and the ubiquitous availability[132]. However, VM hosting in the cloud could have several security and privacy challenges that potentially threaten confidential information stored in the VM. Multi-tenancy in the cloud could poses side channel attack as multiple VMs are sharing physical resources [26]. Guest VM could also be running on an untrusted host machine where a malicious cloud administrator could extract confidential information from the VM. Since a guest VM could not be accessed directly by the malicious administrator, e.g. SSH login, they could execute unauthorized memory dumping attack to generate a memory dump file from the volatile memory (RAM) of a VM [85]. Sensitive data, e.g. password, cryptographic keys, or personal information, currently processed in the VM could be extracted from the VM's memory dump file. The VM could also be susceptible to Spectre [56] and Meltdown [63] attacks that could leak sensitive data from guest VMs.

We propose a novel solution that reduces the possibility of VM's sensitive data to be captured in a memory dump file and increases the security of sensitive data in the VM's memory from Spectre and Meltdown attacks. Following moving target defense (MTD) concept [96], our approach increases complexity and cost of attack for attackers trying to access VM's sensitive data. It moves the sensitive data of application running in the VM across the application's memory allocations (blocks) in RAM. Our approach works on the application level where a function needs to be added into the application source code to move the sensitive data. The movement function runs concurrently with the application's main function.

We implement our approach into encryption-decryption proof-of-concept application as we evaluate it against live memory dumping attack where it could reduce the probability of the keys to be captured in the memory dump file by 25%. We also evaluate our approach against Spectre and Meltdown attacks by extending Spectre's and Meltdown's proof of concept (PoC) application with the result shows that our approach could reduce the sensitive data leakage from memory.

6.2.2 Related Work

6.2.2.1 Memory Dumping

Memory dumping process, also known as memory acquisition, is one of the techniques used in digital forensics that helps to show the latest state of the data, application, and system from the machine's memory [4]. It copies the content of volatile physical memory (RAM) from the beginning of memory address, which is the lowest address, until the end or particular part of the memory at certain memory state. After it finishes copying the content of RAM, it writes the content into a hard disk as a memory dump file. It is also possible to get VM's memory from a virtualization system or hypervisor. With the hypervisor is responsible for allocating memory to various VMs, it is possible to acquire VM's memory without any modification to the software or virtual hardware of the VM itself [37].

By gaining memory dump file from VM, a malicious user could extract sensitive information contained in it, such as cryptographic keys, passwords, and the user's sensitive data [85][4]. Several tools or commands can be used to help extract information from memory dump file, such as rsakeyfind[1] to find RSA key and strings command to find cleartext password.

There are memory dumping countermeasures designed with different purposes such as to prevent memory acquisition from happening, to mitigate reverse engineering, or to prevent sensitive data from being analyzed based on memory dump file. Milković [67] developed a memory anti-forensic toolkit called Dementia that is able to hide information inside the memory dump file during memory acquisition on Windows operating system (OS). Guard pages [115] is one of the methods to avoid unencrypted data to be acquired from memory in Windows OS. It offers on-demand decryption/decompression system for packer and protector by marking all memory pages not immediately needed during loading executable in the memory's run-time as guard pages. This kind of methods could not protect the sensitive data if memory dumping is executed from outside of the VM.

Other methods stores important information outside of RAM to avoid a cold-boot attack. Simmons [105] proposed disk encryption software called Loop-

[1] http://manpages.ubuntu.com/manpages/xenial/man1/rsakeyfind.1.html

Amnesia. It permanently stores randomly generated encryption key inside CPU's model-specific register that ensures no key is ever leaked to RAM. Therefore memory dump file does not contain the key. Mueller et al.[72] proposed TRESOR, a Linux kernel patch for x86 architecture that implements a method, that stores key in CPU's debug registers and runs the AES algorithm on the microprocessor to avoid RAM usage for its encryption and decryption. Mueller et al.[71] integrated AES with Linux kernel to run entirely on microprocessor by using Streaming SIMD Extensions (SSE), which is available in a modern processor, to ensure no information about the key is leaked to RAM. These methods are a hardware-based hiding technique using cache or register to avoid storing the data in the RAM. These techniques are not practical in real-life especially in the cloud computing environment, since the registers' size are quite small to store all the sensitive data of the guest VM, and they require some changes within OS and the virtualization software.

Other countermeasures propose the solution through computer's hardware. AMD developed memory encryption techniques called Secure Memory Encryption (SME) and Secure Encrypted Virtualization (SEV) [53]. SME uses high-performance AES engine for each controller that encrypts data when it is written to DRAM and decrypts it when it is read from DRAM. SEV enforces code execution to run at different isolated levels with each has no access to the resources of the other through cryptographic isolation. Intel developed Software Guard Extensions (SGX) that create a hardware-assisted trusted execution environment. It creates CPU-hardened "enclaves" or protected areas of execution in memory for the application that run sensitive code and data with the enclave data is written to disk encrypted and checked for integrity [45].

Our approach is different from those existing countermeasures since our approach does not need any additional specific hardware. It is working on an application level and able to work even if memory dumping was executed from outside of the VM.

6.2.2.2 Spectre and Meltdown

Spectre [56] and Meltdown [63] extract sensitive data from a restricted area of the volatile memory (RAM) that belong to other running programs. They ex-

ploit critical vulnerabilities in modern processors which have several optimization features, i.e. branch prediction, speculative and out-of-order execution. By manipulating these optimization features, Spectre and Meltdown can extract data from restricted memory area to the cache. The data in the cache could be read using a microarchitectural covert channel (e.g., Flush+Reload[130]) to the outside world. Spectre exploits branch prediction and speculative execution, while Meltdown exploits out-of-order execution.

Some works create patches on the kernel level to stop the vulnerability from being utilized by the attacker. Meltdown patch called KAISER[1] isolates the kernel addresses from the user space process by adding a "shadow" page table contains a copy of all user-space mappings without kernel-space addresses. Some kernel-space address can still be read from user-space and leaves a residual attack surface for Meltdown. Google developed Retpoline [116] to mitigate Spectre in exploiting indirect branch prediction to execute "Branch Target injection". Retpoline is a software construct made of return operations that allows indirect branches to be isolated from speculative execution and ensures that any speculative execution will bounce endlessly. Intel developed IBRS[2] (Indirect Branch Restricted Speculation) to mitigate the branch prediction by restricting the speculation of indirect branches. It stops near returns and indirect jumps/calls from allowing their predicted target address to be controlled by code that is executed in a less privileged prediction mode. These patches are working on the kernel level, while our approach is working on the application level. These patches are not practical against a malicious host admin since these patches need to be implemented on the host machine. The malicious host admin will not implement the patches.

The way Spectre and Meltdown copy the sensitive data from the memory to the outside world is different from live memory dumping. Spectre and Meltdown copy the sensitive data only a small amount of data at one time, e.g. character by character. Live memory dumping can copy data from the memory to hard drive, page by page (4kB). This means live memory dumping can copy one whole sensitive data to the hard drive at once, while the Spectre and Meltdown have to copy

[1] https://lwn.net/Articles/738975/
[2] https://lwn.net/Articles/743019/

the sensitive data character by character. This different way of copying sensitive data from memory to the outside world affects the result of implementing our approach against these attacks. The difference could be seen in the experiment section.

6.2.2.3 Moving Target Defense

According to [96], MTD is the concept of controlling change across multiple system dimensions in order to increase uncertainty and apparent complexity for attackers, reduce their window of opportunity and increase the costs of their probing and attack efforts. There are a lot of MTD techniques have been developed to make the system more defensible against various attacks. Cheng Lei et al.[57] have done a survey on MTD techniques. They analyzed the MTD concept and explain the MTD design principles and system architecture. Jun Xu et al.[128] compares different MTD techniques. They propose a three-layer model to evaluate and compare the effectiveness of different MTD techniques. One of MTD techniques that has been well known and widely deployed is Address Space Layout Randomization (ALSR). It has been used in the operating system to randomize the location of system executable in the memory in order to tackle the buffer-overflow attacks.

Our approach is a novel technique in the MTD field since it is different from the existing MTD techniques. Our approach reduces the attacker's window of opportunity and increases attack efforts by moving the sensitive data across memory blocks in RAM.

6.2.3 Threat Model

Our main objective is to secure the sensitive data in the volatile memory of a remote VM, which is running on a shared host machine in cloud computing, such as public cloud, peer to peer (P2P) cloud, or social cloud. Our approach focuses on mitigating these three threat models that could be committed to extracting the sensitive data from the VM's memory:

1. **Live memory dumping**: Using a live memory dumping tool, the malicious host admin is able to copy the contents of the VM memory to a file on a

hard drive, **without freezing or pausing the VM**. After the memory is dumped to a file, the malicious host admin then could search for sensitive data, such as a secret key or a password. Using the password or the secret key, the malicious host admin could get into the VM to access the data available in the VM.

2. **Spectre**: Using Spectre variant two vulnerability, a user from a guest VM could extract sensitive data out of another guest VM memory content on the same host machine[1]. A user creates an application that exploits the Spectre vulnerability and runs the application on a guest VM. The application can influence the victim process to execute a gadget in the victim address space by training the Branch Target Buffer (BTB) to mispredict a branch from an indirect branch instruction to the address of the gadget. This gadget then could put the sensitive data in the cache [56]. After the sensitive data is stored in the cache, the application could get the sensitive data out using the covert channel method.

3. **Meltdown**: This attack could be delivered by the host admin and the container users. As a container user, the attacker could create an application that exploits Meltdown vulnerability on the container to get data out of the kernel memory and other containers in the same kernel [63]. Meltdown could also be used by the malicious host admin to get the data from guest VMs.

In this thesis, we are using the scenario where the sensitive data used by the application in the VM is stored encrypted in hard drive. We assume that attacker can steal the sensitive data only from memory using these three attacks mentioned above since the sensitive data is stored unencrypted in the memory (RAM).

6.2.4 Moving Sensitive Data in RAM

The sensitive data in the memory is prone to live memory dumping, Spectre and Meltdown attacks. We propose a method to mitigate these attacks by continuously moving the sensitive data among memory blocks of a program in Random

[1] https://security.googleblog.com/2018/01/todays-cpu-vulnerability-what-you-need.html

Access Memory (RAM), as the memory is read from lower memory address to higher memory address during these attacks. A memory block is a group of one or more contiguous chars (bytes) of a memory. In the C programming language, the memory block is created using the *malloc* function.

Moving sensitive data among memory blocks means copying the sensitive data from one memory block to another memory block while removing it from the previous memory block. Removing sensitive data does not mean deallocating its memory allocation, but filling the memory block with random synthetic data, to ensure no sensitive data leaves the trace in memory during the process and to create a fake sensitive data. The memory blocks are created only once and will never be deallocated until the sensitive data is removed from the memory. A pointer is created to always point out to the current block of the sensitive data. This way the main application function can access the sensitive data via the pointer.

The sensitive data are moving in a loop across memory blocks as can be seen in Figure 6.5, where there are five memory blocks (A, B, C, D, E) for the movement of the sensitive data. From the application perspective, the memory blocks are consecutive and contiguous, because the application sees the virtual memory. The virtual memory is mapped to the physical memory by the operating system (OS). In the physical memory, the memory blocks are allocated randomly that their location might not be consecutive and contiguous.

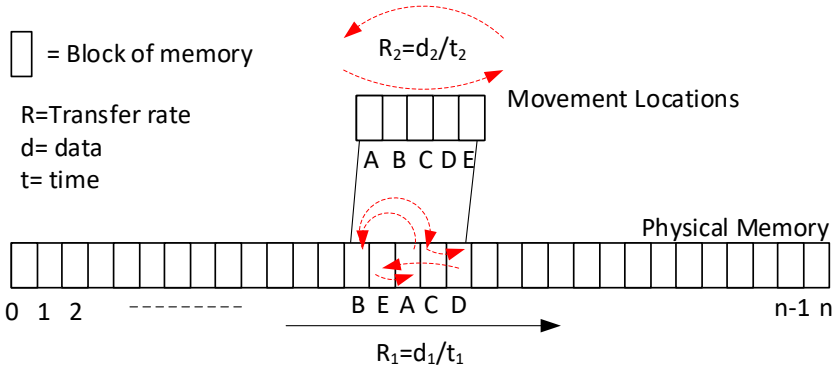

Figure 6.5: Movement in Physical Memory.

In Figure 6.5, the movement locations are from the virtual memory perspective, where the sensitive data is moved forward from one memory block to another memory block sequentially from lowest to highest address (A→B→C→D→E) to finish one loop of movement, and back to the lowest address to start another loop. In the physical memory, the memory blocks are not located sequentially, thus to finish one loop of the sensitive data movement, the sensitive data move forwards and backwards randomly according to the location of the next memory block. Live memory dumping, Spectre and Meltdown are moving forward from low address to high address in physical memory (RAM).

The sensitive data is moved by a function called movement function. This movement function is added into an application source code to be able to move the application's sensitive data. The movement function is called using the thread by the main function that is using the sensitive data to run both of the functions concurrently. The application that is using the sensitive data uses a pointer to a current variable that stores the sensitive data.

The algorithm of the application that moves the sensitive data in the memory can be seen in Algorithm 2. In this algorithm, we use a decryption process as an example of the main function (procedure). Inputs of the algorithm are SecretKey (sensitive data) and EncryptedText. The output is cleartext, and the number of memory blocks (arrayKey) for movement is 3. A volatile global variable is used as a pointer to the secret key as the sensitive data. A movement function runs concurrently with the main function using thread. A thread lock (mutex) is added to synchronize the access to the SecretKey. The movement function continually moves the secret key among memory blocks while still keeping the global variable pointing to the current block of the secret key.

Algorithm 2 Movement Algorithm

```
1: Input: SecretKey, EncryptedText.
2: Output: ClearText
3: Initialization: globalKeyPointer ← NULL, x ← False
4: procedure MOVEMENT
5:     #Allocate several memory allocations (array)
6:     arrayKey[3] ← malloc()
7:     i ← 0
8:     while x is False do
9:         #random key (fake) to override secret key in memory allocations
10:        randomKey ← generateRandomKey()
11:        #moving SecretKey within memory allocations (array)
12:        copy (arrayKey[i], globalKeyPointer)
13:        oldGlobalKey ← globalKeyPointer
14:        globalKeyPointer ← arrayKey[i]
15:        copy (oldGlobalKey, randomKey)
16:        i++
17:        if (i == arrayKey.length) then i ← 0
18:        nanosleep()
19:    end while
20: end procedure
21: procedure MAIN
22:    #SecretKey, a pointer to the memory location of a secret Key
23:    SecretKey ← getSecretKey()
24:    globalKeyPointer ← SecretKey
25:    EncryptedText ← getEncText()
26:    createThreadMutex(MOVEMENT)
27:    for as long as it is needed do
28:        #Decrypt the EncryptedText
29:        decrypt(EncryptedText, globalKeyPointer)
30:        nanosleep()
31:    end for
32:    #stop movement function
33:    x ← True
34:    exitThread()
35: end procedure
```

Our approach could be implemented in any application where the sensitive data could be moved across memory allocations. For example, the ssl_module (mod_ssl.c) of Apache web server could be modified by adding a private key moving function to it. In the client site application, the browser could be modified to implement our approach to secure user credentials. There is a trade-off of our proposal as it needs more memory and more CPU works for the movement function. The time complexity of the movement algorithm is *O(n)* where n is the number of movement function executions which depends on the thread scheduler and the time of the sensitive data is being stored in the memory. The space complexity of the movement algorithm is *O(1)* because the number of memory allocations and the size of the sensitive data (secret key) are fixed during the execution of the movement function. When the size of sensitive data is not fixed, the space complexity is O(n), where n is the size of sensitive data. The overheads are measured in experiments.

6.2.4.1 Moving Sensitive Data Against Live Memory Dumping

When a memory dumping tool copies the data from main memory to a hard drive, it will go sequentially page by page from the first memory location until the last memory location. Usually, the amount of a memory page is 4 kB. The transfer speed R_1 from the main memory to hard drive is based on the memory speed, front side bus (FSB) speed, the hard drive buffer, etc. While the memory dumping tool copies data from memory to a hard drive, the sensitive data is also moving among certain memory blocks with the speed R_2 based on the RAM's frequency specification. The transfer speed is in MB/s. R_1 is much smaller than R_2. As shown in Figure 6.5, the sensitive data are moving in a loop from memory block A to block E then back to block A and so on, until it is removed from the memory.

Virtual memory is created by an OS to expand the main memory capacity by combining the physical memory (RAM) and secondary storage (hard drive). The memory blocks for the sensitive data in the virtual memory are consecutive. In the physical memory, those memory blocks are not consecutive, as they are randomly distributed among the available memory for the application (memory heap). Figure 6.5 shows the flow of moving sensitive data in physical memory. In

this figure, the memory blocks A to E for the sensitive data are not consecutive. Although sensitive data are moving sequentially from memory block A to E in the virtual memory, the memory block of A to E are random in the physical memory; therefore the movement is not consecutive. The memory dumping tool captures the sensitive data if the sensitive data exists in the memory block being dumped to the hard drive. Because the memory blocks are allocated randomly and not consecutive in the physical RAM, the probability of the sensitive data is in the same memory block as the memory dumping tool is 50%.

The success of memory dumping process is measured by capturing sensitive data no matter where its location is or how many times it is captured in multiple locations (blocks) as it is only counted as one time. Captured means the sensitive data is contained in the memory dump file. Because captured sensitive data is counted only once, from the result point of view, there are only two memory allocations available: the location of captured sensitive data and non-captured memory location. The probability of the sensitive data being caught by the memory dumping tool is 75%, which is the same probability of getting at least one head when tossing two coins at the same time.

6.2.4.2 Moving Sensitive Data Against Spectre and Meltdown

Because Spectre and Meltdown extract the sensitive data out of the memory character by character, they could be mitigated by moving the sensitive data within memory blocks. Suppose they could get one character at the first address when they try to get data from the second address, they will get a random character as the content of the memory has been changed with the fake data and the sensitive data has already been moved to another memory allocation. Figure 6.6 shows the example of moving sensitive data against Spectre. In this example, we assume that the speed of moving sensitive data is the same as the speed of the Spectre getting the data from memory and the sensitive data in the restricted memory area is "CroviLab". The moving process and the Spectre process are executed alternately by the process scheduler. Because the sensitive data is moved within 3 memory locations, the data extracted by Spectre is "Cgdvewah" therefore the adversary is getting a wrong result.

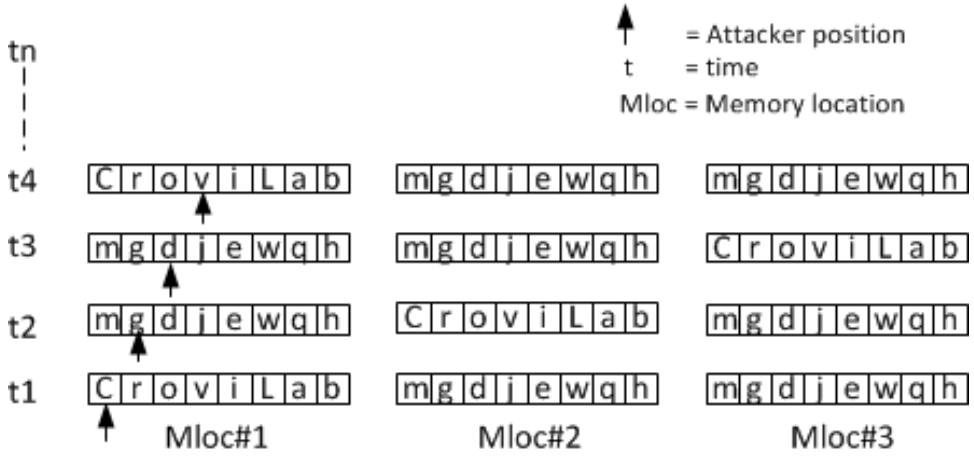

Figure 6.6: Movement Against Spectre.

Meltdown can dump kernel and physical memory with up to 503 kB/s[63]. This speed is much smaller than the speed of copying data within a different location in memory, which is up to 3.2 GB/s for DDR2 with 400 MHz FSB. Our proposal allows sensitive data to not stay at one memory location while the attacking process is extracting data from restricted memory. Another factor that influences the moving of sensitive data against the attacker process is Process Scheduler. In a multi-tasking environment, such as Linux, the process scheduler manages all processes and fairly determines which process is running. The process with sensitive data movement function and the attacker process are running together concurrently. This way we can be sure that the sensitive data always move to another memory location after the attacker process extracted one character from the restricted memory area.

6.2.5 Evaluation

In this Section, we evaluate our approach against live memory dumping, Spectre, and Meltdown attacks by developing proof-of-concept suitable to mitigate each attack. Our approach is working only on a programming language that supports mutable variable where the content of the variable memory location could be replaced with a new value, e.g. C, Ruby, and PHP.

6.2.5.1 Live Memory Dumping

In the live memory dumping experiment, we implemented our proposal into proof-of-concept application for encrypting and decrypting a message[1] that is developed in C language and running in the VM. It includes the secret key moving function and thread creation to make a moving function to be parallel with the encryption/decryption function. The goal of this experiment is to capture secret key used for encryption and decryption from a memory dump file of running VM.

LiMe memory dumping tool[2] is installed and running on the host machine while our application is running on a VM. The host machine is Linux Kubuntu 17.10 32 Bit with Intel Pentium CPU 2.27 GHz and 3 GB RAM. VirtualBox hypervisor is used to run Linux Ubuntu 14.04 VM with one vCPU and 512 MB base memory without swap space. The Linux Kubuntu host machine and the Linux Ubuntu VM are using the default configuration. We created two bash scripts to run the experiment automatically: one script in the VM starts the application after startup and another script on the host machine starts the VM, runs memory dumping process, and reboots the host machine since the host machine needs to be restarted to clear the memory from old data after each trial. It takes about 15 minutes to run a trial in a real live environment.

When the application is running, the encryption and decryption functions are looped long enough until the memory dumping tool has finished dumping all the content of memory (RAM) to a hard drive. This creates the condition of the secret key to be used and available in the memory while the dumping tool is running. Inside the application, we added a function that moves the secret key within three memory allocations. The size of memory allocation is 4096 bytes, which is the size of a memory page because the memory dumping tool copies the data to hard drive page by page. The secret key is in the string format to make it easy to be extracted from the memory dump file using `strings` command. In reality, cryptographic key finder, such as aeskeyfind, could be used to search for a cryptographic key in a memory dump file.

In this experiment, we used three memory allocations (array) to store the secret key. The experiment did not record the memory location of the captured as it

[1] https://gist.github.com/int64Ago/bc816bd950b179e04955
[2] https://github.com/504ensicsLabs/LiME

only records whether one secret key is captured. When the memory dumping tool captured the secret key, it is counted as one captured, no matter how many times the secret key has been captured during one period/trial of an experiment. If the secret key is captured in a memory location, the other locations are considered to be not captured and could be counted as another location thus making it becomes two possible locations of the secret key. Therefore there are 4 possible results of each trial: [0,0], [0,1], [1,0], and [1,1], where captured secret key is represented by 1 and not captured secret key is represented by 0. The experiment runs for 56 trials with the result 41 times the secret key was captured and 15 times was not captured. The success rate of the captured secret key is 73.2%. Increasing the number of memory allocations does not affect the result of our experiment.

We also measured the performances of our approach within different loads of memory by the running time of the application and the movement count of the sensitive data. In this experiment, the sensitive data is a 32 bytes AES secret key and the size of the text to be encrypted is 1024 bytes. The encryption and decryption functions are looped 10000 times. We use *stress-ng*[1] test tool to generate different memory loads from 30% until 90%. Figure 6.7 and 6.8 show the result of the running time and movement count measurement of different memory loads. The running time and the movement count are increased following memory load increase because when the stress tool is executed, the CPU needs to serve more processes simultaneously; therefore it takes more time to execute a function. Because moving function is lighter than the encryption/decryption function, the moving function is executed more often. The running time difference of 30% and 90% memory load is 21 milliseconds, and the movement count difference is 7. These values are really small, so we could assume that the performance of our approach is stable as long as there are still some memory spaces available used for moving the sensitive data.

Moving function has created overhead of the memory and the CPU usage. The overhead can be measured by calculating memory space and time parameters. Memory space overhead is measured by calculating the size difference of additional memory because of the moving function. Time overhead is measured

[1]https://www.cyberciti.biz/faq/stress-test-linux-unix-server-with-stress-ng/

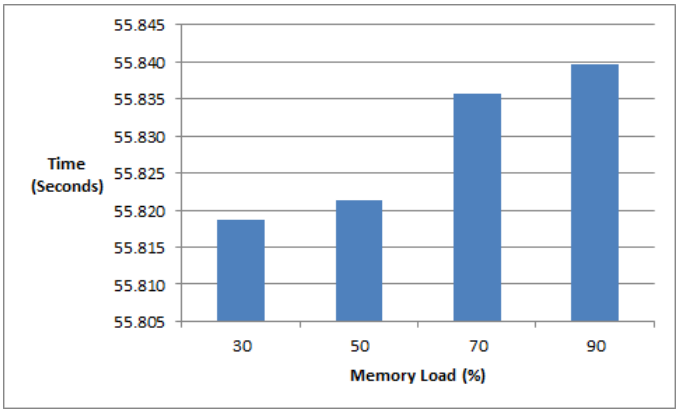

Figure 6.7: 10000 Loops Execution Time.

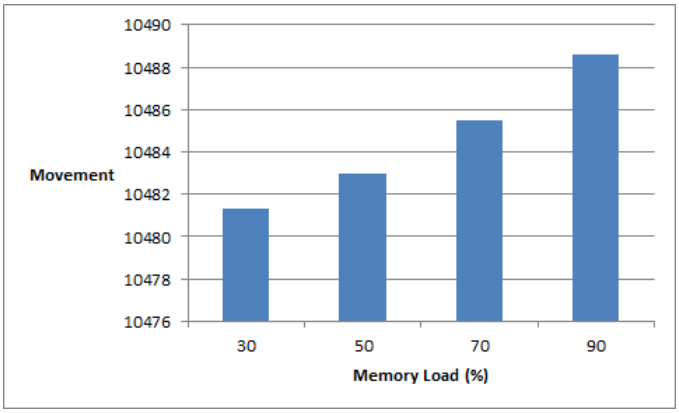

Figure 6.8: 10000 Loops Movements Number.

by calculating the additional times needed because of the additional moving function. Based on the previous experiment, the overhead memory space is 6.68% (954 bytes) since the file size without moving function is 13330 bytes and the file size with the moving function is 14284 bytes. The overhead time is 0.02% (13 milliseconds) where the running time without moving function is 54.538 seconds and the running time with moving function is 54.668 seconds.

From the experiment results, we can conclude that our approach works well in a high load memory (90%) and high load CPU (100%) where there are no significant differences on the execution (running) time and the number of movements. The overheads of our approach are really small that would not affect the performance. Adding memory allocations would not affect the result. In this experiment the size of sensitive data is small.

6.2.5.2 Spectre & Meltdown

In this experiment, we modified PoC source code of Spectre[1] and Meltdown[2] by adding a function that moves the secret value continually among three memory allocations following Algorithm 2. The modified versions could be downloaded from our repository[3]. The function is then running concurrently with the Spectre or Meltdown function in main function thread. We stored the secret value in an array and created a pointer to the array. In the moving function, we created a list of arrays to store the secret value. When the secret value is moved to another array, the pointer is changed to point the new place of the secret value. The old array should be filled with some random characters representing fake sensitive data, but in this experiment, we used the character "b" to see the difference easily. In this experiment, the number of memory allocations for movement is 3.

We run the experiment several times with the result is shown in Table 6.1. The first row is the result of running Spectre/Meltdown PoC without modification, the second row is the result of running a modified version of Spectre PoC which has a movement function inside, and the third row is the result of a modified Meltdown PoC. Spectre PoC generates a "?" when it cannot resolve which character is in a memory location, while Meltdown PoC does not generate anything when it cannot resolve the character of a memory location. That is why the result of Meltdown is shorter than the original data. Spectre could capture 8 characters correctly and Meltdown could capture 9 characters correctly. The captured characters will be meaningless, especially if "b" is replaced by a random character. We can reduce the number of correctly captured characters by increasing the memory allocations because more memory allocations mean more places to move, which means Spectre or Meltdown will capture more fake characters.

Because the behaviour of Spectre against moving sensitive data is similar to Meltdown, we only measure the performance of modified Spectre PoC. To measure the performance, we have tested our approach in four kinds of memory loads: 30%, 50%, 70% and 90%, where the CPU load was always 100%. We use *stress-ng* test tool to generate the load of memory. Performances are measured by

[1] https://github.com/crozone/SpectrePoC
[2] https://github.com/iaik/meltdown
[3] https://github.com/nfs2018/MemoryDumping

Table 6.1: Spectre & Meltdown Experiment Results

Type	Results
Original	The Magic Words are Squeamish Ossifrage.
Spectre	Tbbbbabibb????bb?rbbbqbbbbis?bbbb?[b?gbb
Meltdown	Tbbbabbbobbbab bbubbbbbsbibbbb.

the execution time of the application and the movements number of the sensitive data. We run the modified Spectre PoC for 10 times for each memory load to measure the times and the movements number and calculate the average result.

The results of the experiment are shown in Figure 6.9, 6.10 and 6.11. Figure 6.9 shows the running time of several different memory loads where the higher the memory load is, the higher the running time. When stress tool creates more memory loads, it also creates more CPU loads. Thus it takes more time to run the application, although the difference between memory load 50% and 90% is not significant. The running time difference between memory load 90% and 30% is 1.555 seconds, which is really small to be considered for affecting the performance.

Figure 6.10 shows the movement number of sensitive data within memory allocations. The higher the memory load is, the higher the movement number will be since it takes more time to execute the main function thread on a heavier load memory. The moving number function was executed more frequently because it is lighter than the Spectre function. For 40 characters of sensitive data, the movement number is between 316 and 409 times because the thread system is executing lighter function more frequently. Figure 6.11 shows that the higher the memory load is, the lesser the captured characters will be because the moving function was executed more frequently.

From the experiment results, we can conclude that our approach could protect sensitive data against Spectre and Meltdown attacks. Our approach also works very well in different kind of memory load because the memory load does not significantly affect the execution time and the number of the correct data being captured.

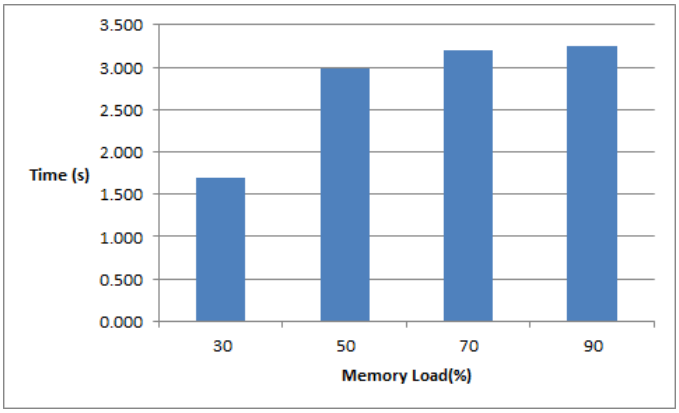

Figure 6.9: Spectre Execution Time.

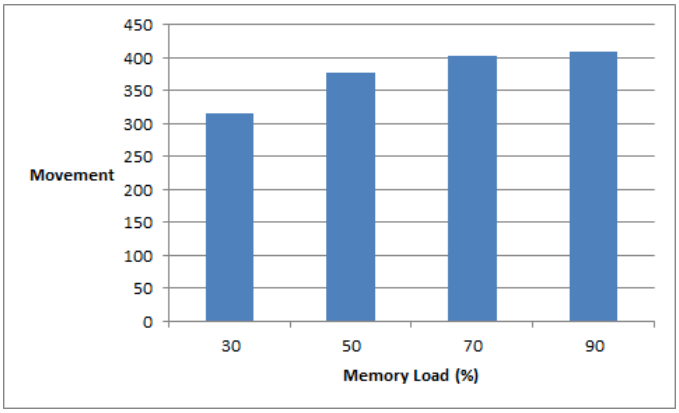

Figure 6.10: Spectre Movements Number.

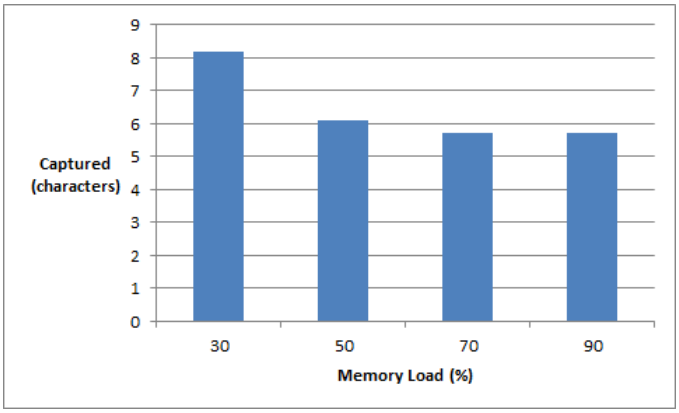

Figure 6.11: Spectre Correct Characters.

6.3 ABTiCI - Agent Based Trust in Cloud Infrastructure

ABTiCI was not specifically designed for CRVL, but it could also be used on CRVL. It was designed to increase the customer trust on the public cloud provider. In CRVL, a VM could be running on many platforms such as a private cloud, a public cloud, and a hypervisor of a contributor's computer machine. These platforms have a similar virtualization concept, where a VM is running on a host machine and the host machine is managed by a host admin. ABTiCI could be used to monitor these platforms remotely, but in this section, we use public cloud platform as a use-case. This approach could be used if the host machine has a TPM chip and the host admin gives permit to install software agent on the host machine.

By design, the cloud system does not allow a cloud administrator to access the customer data in a VM without customer's permission. However, a cloud administrator is able to modify the software/hardware configuration in a way that allows unauthorized access to customer data. This is because the cloud administrator has full control of the cloud infrastructure. He is a superuser in the cloud system and has physical access to the cloud infrastructure. ABTiCI system detects unauthorized access by verifying and monitoring the Integrity of cloud infrastructure security relevant parts. ABTiCI performs integrity verification at boot-time and run-time. ABTiCI uses trusted boot with TPM (Trusted Platform Module) to perform integrity verification at boot-time. ABTiCI also monitors access to security-relevant parts, such as hardware and software configuration, to be able to detect any changes at run-time. ABTiCI uses agents to do integrity verification and to communicate between entities in the cloud infrastructure.

6.3.1 Motivation

The pay-per-use payment model in cloud computing provides advantages from an economic and the scalability point of view. The customer does not need to invest in a high-cost infrastructure and only needs to pay for what has been used, and additional computing resources can be allocated on-demand. Two of the significant issues in slowing down the speed of cloud computing adoption are data confidentiality and integrity. These issues prevent some potential customers from

moving their services and data to the cloud. They need to keep the confidentiality of their data even from the cloud provider. They cannot trust the cloud provider without any evidence. They want to know whenever there is unauthorized access to their data by anyone including the cloud administrator.

Encryption is usually used to protect data and provide security guarantees. However, in cloud computing, encryption might not guarantee data protection from compromised or dishonest infrastructure providers. Such providers have full control of the cloud infrastructure. They can configure and modify the cloud infrastructure to allow unrestricted access to the data without being known by the customer. They can load a malicious hypervisor module, install malicious hardware, perform a side channel attack, modify system software to get access to the customer's data. Rocha and Correia show how a cloud administrator can obtain a private key using a memory snapshot [85].

To be able to access customer's data in the customer's VM, compromised or dishonest cloud infrastructure providers need to modify the hardware or software configuration. For example, to obtain a private key, a cloud administrator needs to enable memory dumping in the hypervisor. As the cloud infrastructure provider has full control of the hardware and software, it is not possible to prevent the cloud provider from modifying them. However, there is a possibility to detect whenever the hardware or software configuration has been changed. In this case, in order for the customer to detect unauthorized access, it is necessary to detect whenever the hardware or software configuration is being changed. This can be done by monitoring the relevant parts of the infrastructure provider and verifying their integrity. By letting the customer monitors part of the infrastructure provider, the cloud becomes more transparent to the customer.

The research problem of this section focuses on how to monitor the security relevant parts of the cloud infrastructure and how to verify their integrity in order to detect unauthorized access that could be gained due to a compromised system or a dishonest cloud infrastructure provider.

6.3.2 Related Work

Huang and Nicol[44] propose a trust mechanism based on evidence, attribute certification, and validation, and conclude by suggesting a framework for integrating

various trust mechanisms to reveal a chain of trust in the cloud. Bouchenak et al. [13] raise the problem of verifying cloud services and survey the existing work in this area. Furthermore, they identify gaps in existing technology in terms of the verification tools provided to the user. The authors discuss the challenges and direction to bridge these gaps. Butt et al. [15] propose a solution that modifies the hypervisor to allow an administration domain (Dom0) for every customer, to enable security and privacy in relation to provider administration domain. This way, it will be possible to have flexible control over clients' VMs. CSA (Cloud Security Alliance) [21] proposes CTP (Cloud Trust Protocol) and STAR (Security, Trust and Assurance Registry) to improve trust in the cloud and ICT market by offering transparency and assurance. These proposals show that the challenges to provide Trusted Cloud with evidence to verify, is still an ongoing and exciting research area.

Sim KM [104] introduces an agent-based paradigm for constructing software tools and testbeds for cloud resource management. He is concerned with the design and development of software agents for bolstering cloud service discovery, service negotiation, and service composition. However, he does not discuss the issue of privacy and security in relation to the cloud provider. Hada et al. [35] propose a trust model for cloud architecture which uses mobile agents as security agents. These exclusive mobile agents obtain useful information from the virtual machine which the user and service provider can utilize to keep track of privacy of their data and virtual machines. This approach uses mobile agents that can alert the cloud provider of the problem if the client installs a malicious code which results in the agent replicating itself in the cloud.

Santos et al. [93] propose a Trusted Cloud Computing Platform (TCCP) to provide a closed box execution environment for DomUs (guest VMs) in the cloud. It adds a new entity to certify all the nodes (physical hosts). This entity called TC (Trusted Coordinator). Every node must be certified by TC to get involved in the cloud. When a customer wants to start their VM in one node, the node must ask TC to decrypt the session key that was sent by the customer. The session key is encrypted using the TC public key. The node needs the session key to communicate with the customer.

[77][36][90] propose an approach to answer the problem stated in this paper using trusted computing technology with TPM (Trusted Platform Module). Here, the integrity of security relevant parts is verified at the boot-time of every physical machine in the cloud. [77] adds the function of integrity verification at run-time. These approaches use TPM to produce the key, hash and store the result. These approaches cannot verify the integrity of the security-relevant parts of a physical machine (Dom0) in an infrastructure provider when the verification software is not installed. A cloud infrastructure provider can have thousands of physical machines. The Certifier cannot verify whether in the Dom0 the verification software is already installed. If the verification software is not installed in the Dom0, the compromised or dishonest cloud provider can change the software/hardware configuration without the knowledge of the customer/certifier. This means that the cloud provider could place certain customer VMs in the Dom0 without verification software being used. This could be performed for the first time when the VM is started or later when migrating the VM to a new Dom0. Migrating could be done any time without being noticed by the customer/certifier, especially during load balancing or failover system and when restoring backup data.

6.3.3 Threat Model

Some potential customers with strict data protection policies do not want to let any unauthorized party (including a cloud provider) access their data. In the cloud system, customers have very limited access to the infrastructure provider. They do not know where their data is stored, and they do not know how the cloud infrastructure is managed. Even though the cloud infrastructure can claim confidentiality and integrity, customers need to be able to verify the claim of the cloud provider. Customers need to be sure that any unauthorized party does not have accessed to their data.

Cloud infrastructure providers have full access to every resource (memory, CPU, hard disk, etc.) in their cloud, but by design, they cannot access the VM directly without receiving authorization from the VM owner. While the VM cannot be accessed directly by the cloud infrastructure provider, it still has access to the resources being used by the VMs. This is because every VM needs resources that are provided by the cloud infrastructure provider. According to

[77], as the owner of the resources, a cloud infrastructure provider can carry out various attacks to access the customers' data. These attacks can (1) access data directly in the physical hosts, (2) exploit vulnerabilities in the cloud customers' VMs, (3) access data directly in the storage, (4) capture data using network sniffers, (5) use back up images to retrieve data. To protect the cloud customer from the attacks, encryption is usually used. However, even though encryption is used to secure data in the cloud, the compromised or dishonest cloud provider can still access the data by configuring or modifying cloud infrastructure without the customer's knowledge. Our research is focused on securing the customer data from an attack that involves accessing data directly in the physical host.

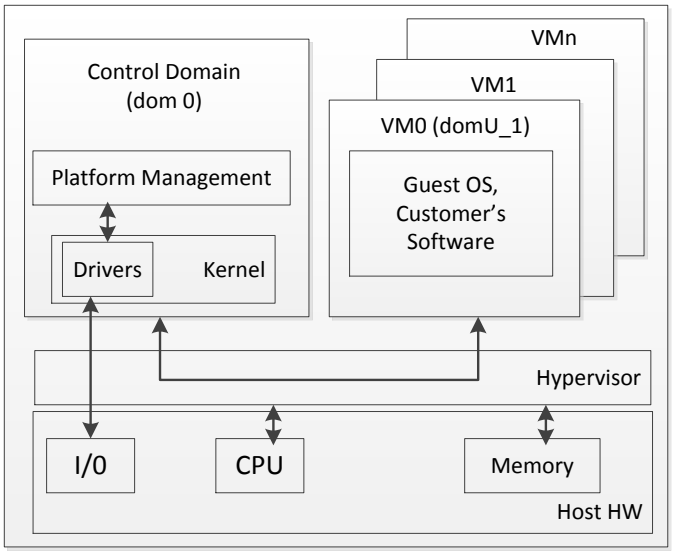

Figure 6.12: Xen Cloud Platform Host.

To describe the problem clearly, Xen Cloud Platform (XCP) is used as the basis. Figure 6.12 shows the architecture of XCP in a physical host and also shows the communication between Dom0 (Domain Zero), DomUs (Domain Unprivileged) and the Hypervisor in XCP. DomU (VMn) is an unprivileged domain (virtual environment) for the customer to be used as needed. DomUs can only access its own resources, like the CPU, and its memory area, and does not have direct access to other hardware devices. The Dom0 is a control domain with special privileges for hardware access. The Dom0 is used for the mapping and management of the Virtual Disk, network interfaces, and other device drivers

required for hardware access by DomUs. The Dom0 provides system files including the hypervisor, the kernel, the kernel module, the drivers (virtual disk and network) and the Xen management tools.

Xen Hypervisor is a computer software that creates and runs virtual machines. Xen Hypervisor manages access to the physical host's resources, Dom0, and DomUs. Xen Hypervisor doesn't allow a VM to access or modify data in the memory of other VM. It manages the communication between DomUs and the hardware devices, such as memory, CPU, and disk drives. The administrators of the infrastructure providers have full access to Dom0 and Hypervisor. From this architecture, we can see that administrators can configure and modify Dom0 and hypervisor to get access to the customer's data without the customer's knowledge.

The cloud administrator has physical access to the host. By having full access on the software and physical access to the host, a cloud administrator is able to install malicious hardware, perform a side channel attack, or modify the system software running on Dom0 or DomUs. He is able to reset the machine and boot with a modified hypervisor or operating system to get access to the customer' data [77]. Cloud administrator can get the customer's data and encryption keys from the memory by using memory snapshot [85]. Another way to have access to the memory is to use DMA (Direct Memory Access). DMA allows access or modifications to the memory without hypervisor control. Cloud administrator can create DMA by inserting a new hardware device or by misusing already available components [77]. It is also possible to create DMA by modifications of the respective drivers [77]. Beside through memory, cloud administrator can modify the disk utilities and configurations to create a duplicate of the data stored by the customer' VMs. He can also modify the network utilities or configurations to redirect all network traffic to a file.

To be able to execute these kind of attacks, most of the time, the cloud administrator needs to modify the hardware or software configuration. For example, to be able to do the memory snapshot, cloud administrator needs to modify the hypervisor to enable the memory dumping function. To be able to create DMA, cloud administrator needs to insert new hardware or modify the driver. In this paper, we focus on attacks targeting modifications of the hardware and software

that allow access or modification to customer's data. We also limit the problem in scope to unauthorized access to data in customer Virtual Machines.

We assume that the cloud system implements encryption to secure the customer's data. In this case, the cloud provider can access the data and encryption key from the memory. We can also assume that whenever the VM is off, the key is not stored in a clear text and a password is required to gain access to it. So, the possibility to gain access to the data and encryption key is via memory. This is because whenever the data is accessed or processed, it is simultaneously stored in the memory in clear text. Both the encryption key and the password are also stored as clear text in the memory.

Neisse et al. [77] propose a bonafides system to solve the problem by doing remote attestation of the integrity of the cloud's infrastructure's integrity. However, this system is not capable of finding out whether the bonafides system is already installed and running in the Dom0 of every DomUs. Therefore, the compromised or dishonest cloud infrastructure provider can put certain DomUs in the physical machines where the bonafides has not been installed. They can do this the first time the VM is created, or when doing recovery from back up, or at the migration to another physical machine.

6.3.4 Architecture and Implementation

In the spirit of providing more transparency and allowing the customer to verify the claim that has been stated by the cloud infrastructure provider, we propose the use of an agent as a mediator between customer and cloud provider. Both sides need to be able to trust this agent. The agent monitors the security-relevant parts of cloud infrastructure and sends reports to the base station. The cloud infrastructure provider knows the capability of the agent exactly. It also recognizes the source code of the agent program to verify whether the agent program is safe and secure for the cloud provider operation. While this agent system cannot stop unauthorized access, it can detect and send reports to the customer. To be able to detect correctly, the integrity of the agent must be maintained on time.

Our first goal is to verify the integrity of the hardware/software configuration remotely. Second goal is to find out whether the Dom0 of physical machines where the VMs are hosted, has an installed verification software. In this case, we

need to verify if there is an agent installed in Dom0. We use the agent to monitor and to verify during the run-time, and we implement a trusted boot using TPM to verify the integrity at the boot-time. We are proposing ABTiCI (Agent-Based Trust in Cloud Infrastructure) system as the answer to the problem.

ABTiCI system is designed to verify integrity at the boot-time and also at the run-time. Verifying the integrity at the boot-time was proposed by [77][36][90], where trusted boot is done using TPM. TPM stores the hash values of the hardware and software configurations in Platform Configuration Registers or PCRs. These values are also sent and stored in a database of a remote machine called Cloud Certifier. The system can verify the integrity of the host machine by comparing the hash result with the one stored in the database. Because the system configuration can still be changed after the boot-time, verifying the system integrity at run-time is still necessary. We focus on verifying the integrity at run-time. Figure 6.13 shows the architecture of ABTiCI.

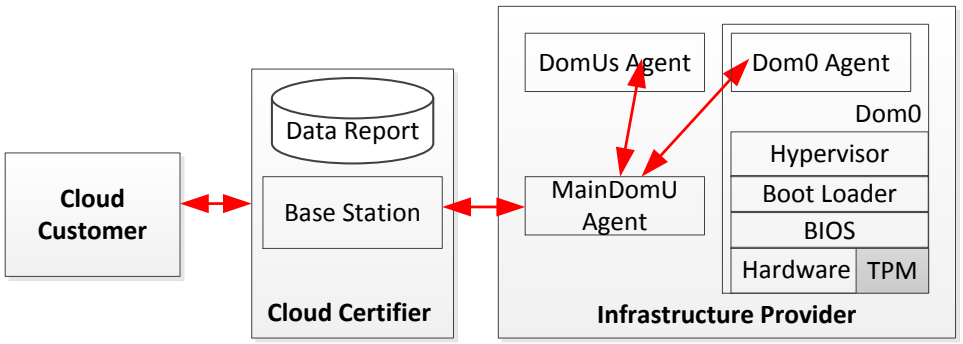

Figure 6.13: ABTiCI Architecture.

To verify the integrity at run-time, ABTiCI uses inotify[1] to monitor the relevant files and an agent to communicate with another agent. The monitoring location is in Dom0 where the compromised or dishonest cloud provider could change the hardware/software configuration to get access to the customer data. We focus on monitoring these security relevant files: hypervisor, kernel module, network driver, I/O driver, etc. Inotify is a Linux kernel subsystem used to monitor files. Whenever the file is changed, inotify informs the Dom0 Agent.

[1]http://man7.org/linux/man-pages/man7/inotify.7.html

The Dom0 Agent is started by MainDomU Agent by executing SSH Connection to the Dom0 OS, to verify the agent binary file and execute the agent program. The MainDomU Agent executes the "ps aux" command and send the output to MainDomU and store it in the database and also send the output to the base station. The MainDomU Agent and the base station know the PID (Process Identifier) and the time when the Dom0 agent program is started.

Via the same SSH connection, the MainDomU Agent executes the "xe vm-vif-list" command on the Dom0 to get the list of VM MAC addresses that was hosted in the same physical machine. The result is stored in the database and also be sent to the base station. The MainDomU Agent gets the MAC Address information from every DomUs and sends it to the base station. This way the MainDomU agent and the base station can construct a map of Dom0 and DomUs based on their MAC Address.

Randomly, the MainDomU Agent sets up the SSH connection and verify whether the running agent is being tampered with or not by checking the PID and the time when the agent was started. If verification failed, the cloud provider must give an explanation and proof. The failed event is recorded in the log for future analysis. Whenever the Dom0 Agent was executed, it takes the integrity information in TPM and sends it to the MainDomU Agent.

6.3.4.1 Roles

As shown in Figure 6.13, ABTiCI system has five different roles: Cloud Customer, Cloud Certifier, Dom0 Agent, MainDomU Agent, and DomU Agents. The Cloud Customer can be a cloud service provider or a service consumer. It can access the data report in the Cloud Certifier to see the status of it's VM. The report could be in a dashboard form to make it easy to understand. The reporting system can be arranged as needed.

Cloud Certifier. The Cloud Certifier has a role in inspecting the infrastructure provider. It determines whether the infrastructure provider is a trustworthy provider or not. As shown in figure 6.13, Cloud Certifier has a base station with the function of starting ABTiCI by sending a message to the MainDomU Agent to start the Dom0 Agent. A base station has an agent that communicates with the MainDomU Agent to send a command to start ABTiCI or to receive data

reports from the MainDomU Agent. In the base station, there is an application that analyzes the data reports and creates a dashboard view for every customer. The dashboard view, informs the customer when the security relevant parts of the cloud infrastructure provider change. Furthermore, it informs the customers about the location of their VMs in the cloud by mapping the DomUs to the Dom0 using MAC Addresses. There is only one Cloud Certifier in the ABTiCI architecture.

Dom0 Agent. Dom0 is an administrative domain where the cloud infrastructure provider manages the resources in a physical host that will be outsourced by the VMs. The Dom0 operating system is the first operating system installed in a physical host. Dom0 Agent is an agent in a Dom0 that monitors the relevant parts of a physical host in a cloud infrastructure provider and sends the report to the MainDomU Agent. The Dom0 Agent monitors the relevant parts of the physical host in order to detect unauthorized access by a compromised or dishonest cloud provider. The Dom0 Agent uses hash and digital signature in TPM to hash and sign a message that is sent to the MainDomU Agent. Dom0 Agents also send the list of MAC addresses of DomUs to the MainDomU Agent. Every physical host has one Dom0 Agent. The Dom0 Agent uses inotify to monitor several files and uses trusted computing technology with TPM to detect the integrity of the hypervisor, kernel and kernel module at boot-time.

MainDomU Agent. MainDomU Agent is an agent in one of the VMs of a physical host in the cloud infrastructure. This VM is controlled by the Cloud Certifier to monitor the cloud infrastructure provider. MainDomU Agent plays a central role in communicating with another agent in the cloud infrastructure to perform the verification and monitoring function. MainDomU Agent verifies the agent program in Dom0 and starts the Dom0 Agent through the SSH connection. MainDomU Agent is responsible for polling the MAC address of every DomUs. MainDomU Agent polls the MAC Addresses of the Dom0 and also receives data reports from Dom0 Agent and sends data reports to the base station. There is only one MainDomU Agent for the whole cloud infrastructure.

DomUs Agent. DomUs Agent is an agent in every DomU except the one in the MainDomU. DomUs Agent is responsible for sending its own MAC Address to the Main DomU Agent. There is one agent in every DomU.

6.3.4.2 Integrity Verification

The integrity verification is performed on security relevant parts and on the agent itself to detect the possibility of unauthorized access.

Agent Integrity Verification. The agent is verified before it is started and randomly at certain time intervals after the agent was started. Figure 6.14 shows the agent verification process. As shown in the figure, the MainDomU Agent uses SSH to connect to Dom0 and to run the verification commands to verify the agent. MainDomU Agent should log-in with enough access rights to do the verification process. To verify agent integrity, the Main DomU Agent hashes the agent binary and a nonce and compares the result with the result of the hash of the same file and nonce. The original binary agent is stored in the base station and the MainDomU Agent. If the result is the same, the agent binary file is authentic. MainDomU Agent executes the agent binary file and executes the ps command to get the PID, starting time, and file location. MainDomU Agent also executes the "xe vm-vif-list" command, to get the list of MAC address of all VMs in that physical host. MainDomU Agent will also get the current content of PCRs to verify the integrity of the security-relevant parts. In the end, the MainDomU Agent closes the SSH connection. The same verification will then be done randomly, to maintain the integrity of the agent.

Security-Relevant Parts Integrity. Verification of the security-relevant parts is delivered at the boot-time and the run-time. The security relevant parts are hypervisor, kernel, kernel module, system libraries, I/O driver, etc. Integrity Verification at the boot-time is done by using a TPM chip to deliver trusted boot. This is carried out by using the hash function in TPM to hash the relevant parts and put the hash result in the TPM (PCRs). After the Dom0 Agent was started, the agent gets the hash values of PCRs from the TPM and sends them to the MainDomU Agent. The MainDomU Agent forward it to the base station/certifier. The base station compares the hash values to the one stored in its database. If the result is the same, it is confirmed that the file has not been changed. If the file has been changed the new content is saved in the database, and the certifier asks the cloud provider an explanation of the reasons behind the changes.

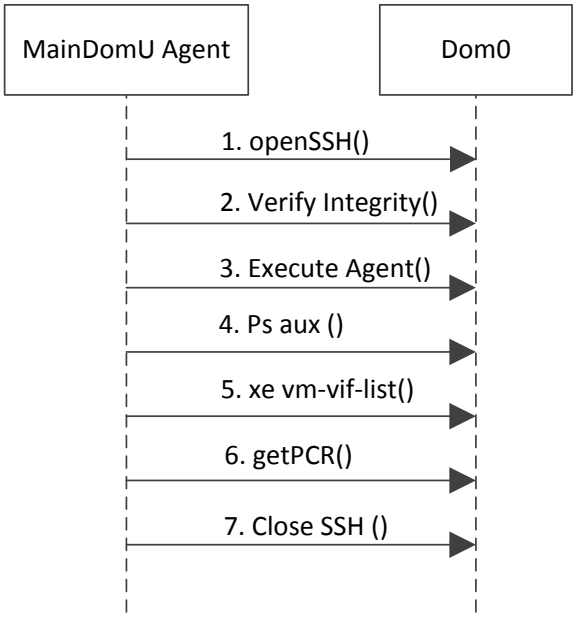

Figure 6.14: Agent Integrity Verification.

Verification at the run-time is delivered using an agent. Dom0 Agent uses inotify to monitor the security-relevant parts of cloud infrastructure. Whenever the files have been changed, inotify let the agent know. As shown in figure 6.15, after being informed by inotify about the change, the Dom0 Agent informs the MainDomU Agent that one of the files has been changed. The MainDomU Agent asks the Dom0 Agent to send the changed file name and the time of change (changelog). The MainDomU Agent sends a report to the base station, and again the certifier asks the cloud infrastructure provider to explain the reason for the change. The changelog will also be stored in the database.

Figure 6.15 shows the whole ABTiCI message flows. Whenever a physical host in the cloud infrastructure is turned ON and joins the cloud cluster, the MainDomU Agent informs the base station. To let the MainDomU Agent knows whenever a new physical host has joined the cloud pool, the Dom0 Agent tells the MainDomU Agent. The Dom0 Agent checks the list of hosts periodically. The time range of the periodical check will be decided later, but it depends on the possibility that the new host will be installed in the amount of time.

The message continues to flow when the MainDomU Agent starts the agent

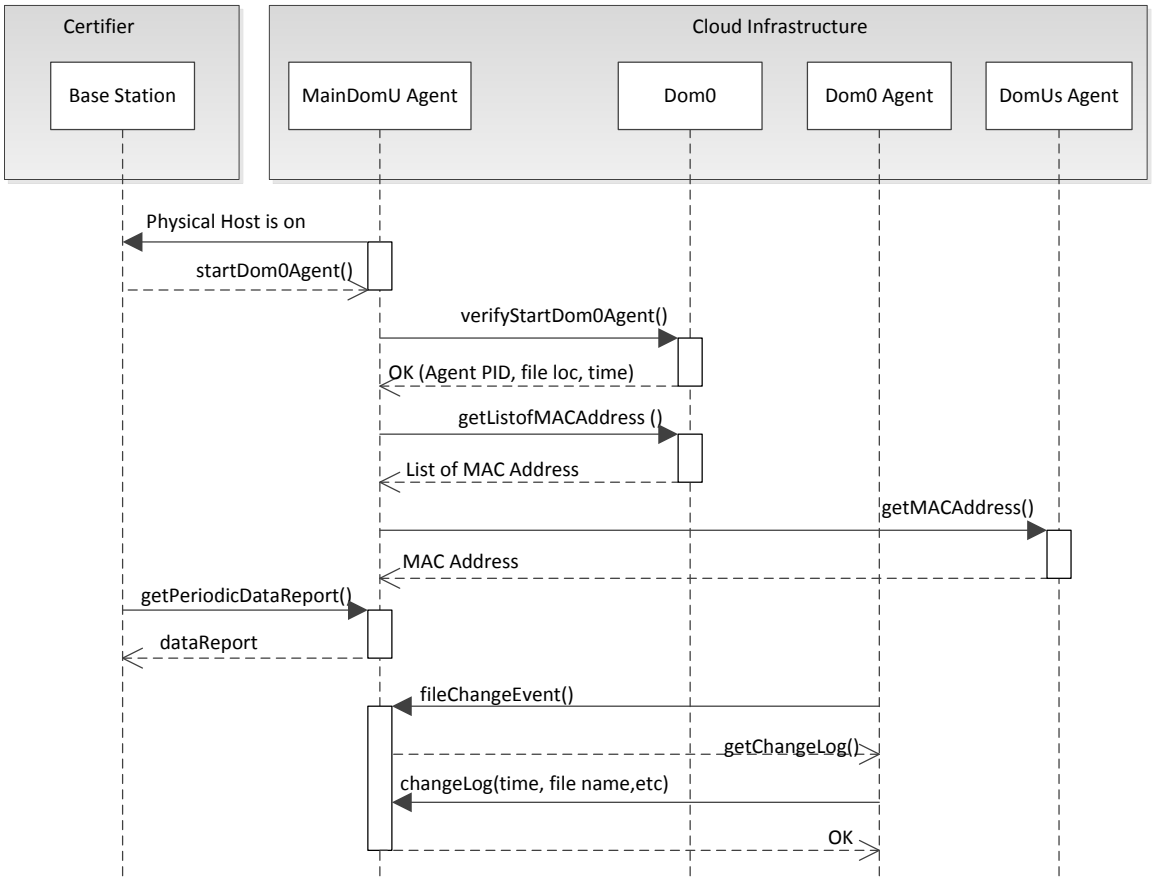

Figure 6.15: ABTiCI Message Flows.

in Dom0 at the new physical host by performing the integrity verification beforehand. After the Dom0 Agent was started, the MainDomU Agent stores the PID, file location and starting time in the database and also sends it to the base station. In order to map the DomUs to the Dom0, the MainDomU Agent gets the list of VM MAC addresses from Dom0 and receives the MAC address from DomUs.

6.3.4.3 Implementation

ABTiCI depends on other software or libraries that must be pre-installed before it can be started. An agent communication platform must be installed for all roles except the customers. In this research, we used JADE as the communication platform for the agents. JADE must be installed in every host where the agent

exists. Agent programs for Dom0, DomUs and MainDomU must also be installed before the ABTiCI system is started. All other software or applications like a hypervisor, SSH, and inotify must also be installed. Especially for the Dom0 Agent program, the cloud infrastructure provider should study the source code and verify whether the agent program is secured for its cloud system.

Most of the software or libraries that are used in this research are already available on the Internet. We need to create an agent program with specific functions as has previously been described above.

6.4 Chapter Summary

In this chapter, we describe three approaches to strengthen the VM from a malicious host admin attacking the data confidentiality. The first approach is to verify the Integrity of a VM that is running on an untrustworthy host machine before it is being integrated into the Crowd-Resourcing Virtual Laboratory (CRVL). This is to prevent a compromised VM from being integrated into the virtual laboratory. New kernel files are used to create an authentic kernel as a root of trust in measuring the integrity of the VM. kexec is used to reboot the VM into the new kernel. The new kernel is used as the root of trust in measuring the integrity of system files in Linux.

This approach is using module signature verification to prevent the malicious module from being injected to the kernel, and by using the hash checksum of system files, the approach could detect a compromised VM. We assumed that the kexec is not compromised, because it is installed just before it is executed. The existence of Secret Code in the kernel files, forces the usage of the authentic kernel files at reboot, to be able to be integrated into the virtual laboratory system. Even though it is difficult to do, the attacker might still be able to patch the authentic kernel on the fly, but more researches need to be done to support this statement. In the future, we need to find a way to make sure that a malicious program is not infecting the kexec.

The second approach is to secure sensitive data of a running application in a VM from live memory dumping, Spectre, and Meltdown attacks by moving the sensitive data within several memory locations. We prove that our approach could

reduce the probability of sensitive data leakage by 25% for live memory dumping attack and able to protect sensitive data from Spectre and Meltdown attacks. Our approach could also perform well in heavy memory load. For the future work, our approach could be combined with anti-memory dumping techniques or obfuscation techniques to better secure sensitive data from memory acquisition attack. An idea of splitting sensitive data into small pieces (blocks) and put them in the non-consecutive area should also be investigated to strengthen our approach.

The third approach is to verify and monitor the security-relevant parts of cloud infrastructure using software agents to detect unauthorized access by a malicious host admin of the cloud infrastructure providers. The agents are used as a mediator between cloud infrastructure provider and cloud certifier, to provide a more transparent cloud. This approach is called ABTiCI (Agent-Based Trust in Cloud Infrastructure). In the ABTiCI System, the agents get the data report from every Dom0 and DomUs and send the report to the base station. The base station calculates the data and verifies the Integrity of the security-relevant parts.

ABTiCI is using SSH connection between MainDomU and Dom0. This connection is established only when needed. ABTiCI could be improved by always connecting the MainDomU to Dom0 via SSH connection, to intensively monitor the security-relevant parts of Dom0. ABTiCI is not easy to implement because it needs access to the host machine (Dom0) to install and run the software agent. The host admin needs to give access to the ABTiCI system, but in most cases, the host admin is not willing to give access for a lot of reasons.

Chapter 7

Conclusion

IT security (Cyber Security) e-Learning platform needs to provide a virtual laboratory for hands-on exercises in attacking and defending the IT system. The attacker or victim nodes are represented by Virtual Machines (VMs) in an isolated environment. Since VM needs a large and fixed allocation of resources, scalability has become a problem where the number of users is limited by the available resources. The scalability could be increased by increasing efficiency and by providing more resources. In this thesis, several technical solutions are proposed to improve scalability. We also propose several approaches to strengthen the VM from malicious host admin, because one approach in providing more resources is by gathering resources from the crowd in the form of a VM.

One approach to increase the scalability is increasing the usage efficiency of the available resources by replacing Virtual Machines with Containers whenever it is compatible to the exercise scenario. The architecture is built based on Tele-Lab architecture by adding Containers platform (Docker) to the existing Tele-Lab architecture. Another approach in increasing the efficiency is by sharing the load with the user on-premise machine. For example, a VM that was used to run an Internet browser to attack a web application is replaced with the user on-premise machine.

An approach to providing more resources is using public cloud services, but it is limited to the available budget. The architecture of IT Security virtual laboratory on public cloud is similar to the Tele-Lab on Private Cloud. The difference is on the middleware because they use different API to start up or shut down

a VM. Another approach to providing more resources is by gathering resources from the crowd. The resources could be in the form of a VM, a bare metal system, an account in the public cloud, a private cloud, and an isolated group of VMs, but in this thesis, we focus on a VM. The crowd as contributors shared their VMs by giving the credential of the VM admins to the crowd-resourcing virtual laboratory (CRVL) system.

In this thesis, we propose a CRVL architecture to manage and monitor the VMs from the crowd. We also propose methods to Integrate VMs to CRVL system automatically. Live Migration and Fault Recovery are also proposed to dis-Integrate a VM and redirect users to other VMs automatically. Since VMs and Users could be spread around the world, we propose a Team Placement algorithm to select a VM to run a virtual laboratory for a particular user. The algorithm is based on Users and VMs geo-location and the VM loads. These approaches could function and perform well in our implementation.

In the CRVL system, everyone including a bad guy could contribute a VM into the CRVL system. As the host admin of a VM, the bad guy could get sensitive data out of the VM by using several attacks such as live memory dumping, Spectre, Meltdown, and by tampering with the OS Image. We propose an approach to strengthen the VM from the live memory dumping, Spectre and Meltdown attacks, by continually moving the sensitive data in RAM. We also propose an approach to verify the integrity of the VM to find out whether the OS Image and the running VM have already tampered or not. Using an agent to monitor the host machine of a running VM is also proposed in this thesis. We implement these approaches to evaluate their functions and performances.

In the future, it is necessary to further research on the CRVL system to improve the scalability and the security. To increase the efficiency, we need to investigate running a desktop on a container and share more resources on user on-premise machine. To enhance security, we need to improve our approach to be able to mitigate the live memory dumping, and we need to find an approach to mitigate static memory dumping.

The proposed approaches in this thesis are our contributions to answer the research questions regarding increasing the scalability of cybersecurity virtual laboratory. Our approaches are novel, unique and could be used to solve the

same problem on other platforms. Based on the experiment results, we can see that our solutions could answer the research questions.

References

[1] Docker overview. https://docs.docker.com/engine/docker-overview/, accessed on 2019-04-05. 18, 20, 21

[2] Dockerfile reference. https://docs.docker.com/engine/reference/builder/, accessed on 2019-04-05. 20

[3] Akoush, S., Sohan, R., Rice, A., Moore, A. W., and Hopper, A. (2010). Predicting the performance of virtual machine migration. In *18th IEEE/ACM International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS 2010)*, pages 37–46. IEEE. 70

[4] Amari, K. (2009). Techniques and tools for recovering and analyzing data from volatile memory. https://www.sans.org/reading-room/whitepapers/forensics/techniques-tools-recovering-analyzing-data-volatile-memory-33049. Accessed on 23 April 2019. 102

[5] Amazon (2019a). Amazon virtual private cloud. https://aws.amazon.com/vpc/. Accessed on 24 April 2019. 39, 40

[6] Amazon (2019b). Making api requests. https://docs.aws.amazon.com/AWSEC2/latest/ APIReference/making-api-requests.html. Accessed on 25 April 2019. 42

[7] Amazon (2019c). Share an object with others. https://docs.aws.amazon.com/AmazonS3/latest/dev/ ShareObjectPreSignedURL.html, Accessed on 2019-04-08. 28

[8] Amorin, K., Shekar, N. H., and AlAufi, L. (2014). Cloudwhip: A tool for provisioning cyber security labs in the amazon cloud. In *Proceedings of the International Conference on Security and Management (SAM)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer 38

[9] Apache (2019). Reverse proxy guide. https://httpd.apache.org/docs/2.4/howto/reverse_proxy.html, accessed on 23 April 2019. 29

[10] ATMC Social (2019). Cybersecurity: one million job openings and counting. https://www.latrobe.edu.au/nest/cybersecurity-one-million-job-openings-and-counting/. Accessed on 26 April 2019. 1

[11] Bala, A. and Chana, I. (2012). Fault tolerance-challenges, techniques and implementation in cloud computing. *International Journal of Computer Science Issues (IJCSI)*, 9(1):288. 68, 69

[12] Border, C. (2007). The development and deployment of a multi-user, remote access virtualization system for networking, security, and system administration classes. In *ACM SIGCSE Bulletin*, volume 39, pages 576–580. ACM. 11

[13] Bouchenak, S., Chockler, G., Chockler, H., Gheorghe, G., Santos, N., and Shraer, A. (2013). Verifying cloud services: present and future. *ACM SIGOPS operating systems review*, 47(2):6–19. 121

[14] Brossard, J. and Demetrescu, F. (2012). Hardware backdooring is practical. *BlackHat, Las Vegas, USA*. 98

[15] Butt, S., Lagar-Cavilla, H. A., Srivastava, A., and Ganapathy, V. (2012). Self-service cloud computing. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 253–264. ACM. 121

[16] Challita, S., Paraiso, F., and Merle, P. (2017). A study of virtual machine placement optimization in data centers. In *7th International Conference on Cloud Computing and Services Science (CLOSER)*, pages 343–350. 52

[17] Cisco (2019). Threats are rising. https://www.cisco.com/c/m/en_au/products/security/offers/cybersecurity-reports.html. Accessed on 26 April 2019. 1

[18] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. (2005). Live migration of virtual machines. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association. 70

[19] Cordel, D., Meinel, C., Repp, S., and Willems, C. (2008). Explorative learning of wireless network security with tele-lab it-security. In *International Conference on Hybrid Learning and Education*, pages 213–224. Springer. 12

[20] CRIU (2019). Docker. https://criu.org/Docker. Accessed on 24 April 2019. 71

[21] CSA (2019). Csa star (security , trust and assurance registry) program. https://cloudsecurityalliance.org/star/. Accessed on 15 April 2019. 121

[22] CSSIA (2019). National center for systems security and information assurance. http://www.cssia.org/. Accessed on 24 April 2019. 39

[23] Cybersecurity Ventures (2019a). 2019 cybersecurity almanac: 100 facts, figures, predictions and statistics. https://cybersecurityventures.com/cybersecurity-almanac-2019/. Accessed on 26 April 2019. 1

[24] Cybersecurity Ventures (2019b). Cybersecurity jobs report 2018-2021. https://cybersecurityventures.com/jobs/. Accessed on 26 April 2019. 1

[25] Cybrary (2019). Cybrary virtual security labs. https://www.cybrary.it/. Accessed on 24 April 2019. 39

[26] Dillon, T., Wu, C., and Chang, E. (2010). Cloud computing: issues and challenges. In *2010 24th IEEE international conference on advanced information networking and applications*, pages 27–33. Ieee. 101

[27] docker (2019). Docker checkpoint. https://docs.docker.com/engine/reference/commandline/checkpoint/. Accessed on 24 April 2019. 71

[28] Du, W. and Wang, R. (2008). Seed: A suite of instructional laboratories for computer security education. *Journal on Educational Resources in Computing (JERIC)*, 8(1):3. 38

[29] Elearnsecurity (2019). Hera lab. https://www.elearnsecurity.com/virtual-labs/hera/. Accessed on 24 April 2019. 39

[30] Felter, W., Ferreira, A., Rajamony, R., and Rubio, J. (2015). An updated performance comparison of virtual machines and linux containers. In *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pages 171–172. IEEE. 18

[31] Garfinkel, T., Rosenblum, M., et al. (2003). A virtual machine introspection based architecture for intrusion detection. In *Ndss*, volume 3, pages 191–206. 87

[32] Google (2019). V4 signing process with your own program. https://cloud.google.com/storage/docs/access-control/signing-urls-manually, Accessed on 2019-04-23. 28

[33] Graupner, H., Torkura, K., Berger, P., Meinel, C., and Schnjakin, M. (2015). Secure access control for multi-cloud resources. In *2015 IEEE 40th Local Computer Networks Conference Workshops (LCN Workshops)*, pages 722–729. IEEE. 29

[34] Gu, Y., Fu, Y., Prakash, A., Lin, Z., and Yin, H. (2012). Os-sommelier: memory-only operating system fingerprinting in the cloud. In *Proceedings of the Third ACM Symposium on Cloud Computing*. 87

[35] Hada, P. S., Singh, R., and Manmohan, M. (2011). Security agents: A mobile agent based trust model for cloud computing. *International Journal of Computer Applications*, 36(12):12–15. 121

[36] Haldar, V., Chandra, D., and Franz, M. (2004). Semantic remote attestation: a virtual machine directed approach to trusted computing. In *USENIX Virtual Machine Research and Technology Symposium*, volume 2004. 122, 126

[37] Hay, B., Bishop, M., and Nance, K. (2009). Live analysis: Progress and challenges. *IEEE Security & Privacy*, 7(2). 102

[38] Hill, J., Carver Jr, C. A., Humphries, J. W., and Pooch, U. W. (2001). Using an isolated network laboratory to teach advanced networks and security. In *ACM SIGCSE Bulletin*, volume 33, pages 36–40. ACM. 38

[39] Hu, B., Lei, Z., Lei, Y., Xu, D., and Li, J. (2011). A time-series based precopy approach for live migration of virtual machines. In *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, pages 947–952. IEEE. 70

[40] Hu, J., Cordel, D., and Meinel, C. (2005). Virtual machine management for tele-lab ït-securityŝerver. In *10th IEEE Symposium on Computers and Communications (ISCC05)*, pages 448–453. IEEE. 12

[41] Hu, J. and Meinel, C. (2004). Tele-lab "it-security" on cd: portable, reliable and safe it security training. *Computers & Security*, 23(4):282–289. 11

[42] Hu, J., Meinel, C., and Schmitt, M. (2004). Tele-lab it security: an architecture for interactive lessons for security education. In *ACM SIGCSE Bulletin*, volume 36, pages 412–416. ACM. 12

[43] Hu, J., Schmitt, M., Willems, C., and Meinel, C. (2003). A tutoring system for it security. In *Security education and critical infrastructures*, pages 51–60. Springer. 11

[44] Huang, J. and Nicol, D. M. (2013). Trust mechanisms for cloud computing. *Journal of Cloud Computing: Advances, Systems and Applications*, 2(1):9. 120

[45] Intel Corporation (2019a). Intel software guard extensions (intel sgx). https://software.intel.com/en-us/sgx/details. Accessed on 23 April 2019. 88, 103

[46] Intel Corporation (2019b). Intel® software guard extensions remote attestation end-to-end example. https://software.intel.com/en-us/articles/intel-software-guard-extensions-remote-attestation-end-to-end-example. Accessed on 23 April 2019. 88

[47] (ISC)2 (2019). (isc)2 report finds cybersecurity workforce gap has increased to more than 2.9 million globally. https://www.isc2.org/News-and-Events/Press-Room/Posts/2018/10/17/ISC2-Report-Finds-Cybersecurity-Workforce-Gap-Has-Increased-to-More-Than-2-9-Million-Globally. Accessed on 26 April 2019. 1

[48] ISU (2019). Iserink. http://www.iserink.org/. Accessed on 24 April 2019. 38

[49] Jhawar, R., Piuri, V., and Santambrogio, M. (2013). Fault tolerance management in cloud computing: A system-level perspective. *IEEE Systems Journal*, 7(2):288–297. 69, 79

[50] Jim Finkle, D. V. (2019). Database of 191 million u.s. voters exposed on internet: researcher. https://www.reuters.com/article/us-usa-voters-breach-idUSKBN0UB1E020151229. Accessed on 26 April 2019. 1

[51] Johnson, S., Scarlata, V., Rozas, C., Brickell, E., and Mckeen, F. (2016). Intel® software guard extensions: Epid provisioning and attestation services. *White Paper*, 1:1–10. 88

[52] Kapil, D., Pilli, E. S., and Joshi, R. C. (2013). Live virtual machine migration techniques: Survey and research challenges. In *2013 3rd IEEE International Advance Computing Conference (IACC)*, pages 963–969. IEEE. 70

[53] Kaplan, D., Powell, J., and Woller, T. (2019). Amd memory encryption. http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf. Accessed on 223 April 2019. 103

[54] Kleissner, P. (2009). Stoned bootkit. *Black Hat USA*, pages 5–7. 98

[55] Knopper, K. (2000). Building a self-contained autoconfiguring linux system on an iso9660 file system. In *Annual Linux Showcase & Conference*. 11

[56] Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., and Yarom, Y. (2018). Spectre attacks: Exploiting speculative execution. *ArXiv e-prints*. 101, 103, 106

[57] Lei, C., Zhang, H.-Q., Tan, J.-L., Zhang, Y.-C., and Liu, X.-H. (2018). Moving target defense techniques: A survey. *Security and Communication Networks*, 2018. 105

[58] Li, P. and Mohammed, T. (2008). Integration of virtualization technology into network security laboratory. In *2008 38th Annual Frontiers in Education Conference*, pages S2A–7. IEEE. 38

[59] Li, Y., Nguyen, D., and Xie, M. (2017). Ezsetup: A novel tool for cybersecurity practices utilizing cloud resources. In *Proceedings of the 18th Annual Conference on Information Technology Education*, pages 53–58. ACM. 38, 39

[60] Li, Y. and Xie, M. (2016). Platoon: A virtual platform for team-oriented cybersecurity training and exercises. In *Proceedings of the 17th Annual Conference on Information Technology Education*, pages 20–25. ACM. 38

[61] Lin, C.-H., Liu, J.-C., and Lien, C.-C. (2008). Detection method based on reverse proxy against web flooding attacks. In *2008 Eighth International Conference on Intelligent Systems Design and Applications*, volume 3, pages 281–284. IEEE. 29

[62] Lin, Z., Rhee, J., Zhang, X., Xu, D., and Jiang, X. (2011). Siggraph: Brute force scanning of kernel data structure instances using graph-based signatures. In *Ndss*. 87

[63] Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., and Hamburg, M. (2018). Meltdown. *ArXiv e-prints*. 101, 103, 106, 112

[64] Mann, Z. Á. (2015). Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms. *Acm Computing Surveys (CSUR)*, 48(1):11. 52

[65] McAuley, A., Stewart, B., Siemens, G., and Cormier, D. (2010). The mooc model for digital practice. http://www.academia.edu/download/43171388/MOOC_Final.pdf. Accessed on 23 April 2019. 2

[66] Microsoft (2019). Using shared access signatures (sas). https://docs.microsoft.com/en-us/azure/storage/common/storage-dotnet-shared-access-signature-part-1, Accessed on 2019-04-23. 28

[67] Milković, L. (2012). Defeating windows memory forensics. https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/dementia-forensics/Defeating%20Windows%20memory%20forensics.pdf. Accessed on 24 April 2019. 102

[68] Mills, K., Filliben, J., and Dabrowski, C. (2011). Comparing vm-placement algorithms for on-demand clouds. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 91–98. IEEE. 52

[69] Mirkin, A., Kuznetsov, A., and Kolyshkin, K. (2008). Containers checkpointing and live migration. In *Proceedings of the Linux Symposium*, volume 2, pages 85–90. 70, 71

[70] Moritz, D., Willems, C., Goderbauer, M., Moeller, P., and Meinel, C. (2013). Enhancing a virtual security lab with a private cloud framework. In *Proceedings of 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*, pages 314–320. IEEE. 3, 4, 10, 12, 13, 38, 40, 43

[71] Müller, T., Dewald, A., and Freiling, F. C. (2010). Aesse: a cold-boot resistant implementation of aes. In *Proceedings of the Third European Workshop on System Security*, pages 42–47. ACM. 103

[72] Müller, T., Freiling, F. C., and Dewald, A. (2011). Tresor runs encryption securely outside ram. In *USENIX Security Symposium*, volume 17. 103

[73] Nadgowda, S., Jayachandran, P., and Verma, A. (2013). 12map: Cloud disaster recovery based on image-instance mapping. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 204–225. Springer. 70, 79

[74] Nadgowda, S., Suneja, S., Bila, N., and Isci, C. (2017). Voyager: Complete container state migration. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2137–2142. IEEE. 71, 73

[75] National Initiative For Cybersecuriy Education (2019). Nice challenge project. https://nice-challenge.com/. Accessed on 24 April 2019. 39

[76] NDG (2019). Cyber security training using netlab+. https://www.netdevgroup.com/content/cybersecurity/. Accessed on 24 April 2019. 39

[77] Neisse, R., Holling, D., and Pretschner, A. (2011). Implementing trust in cloud infrastructures. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 524–533. IEEE Computer Society. 122, 123, 124, 125, 126

[78] Openstack (2019). Temporary url. https://docs.openstack.org/juno/config-reference/content/object-storage-tempurl.html, Accessed on 2019-04-08. 28

[79] Pappano, L. (2012). The year of the mooc. *The New York Times*, 2(12):2012. 2

[80] Peinl, R., Holzschuher, F., and Pfitzer, F. (2016). Docker cluster management for the cloud-survey results and own solution. *Journal of Grid Computing*, 14(2):265–282. 53

[81] Perez, R., Sailer, R., van Doorn, L., et al. (2006). vtpm: virtualizing the trusted platform module. In *Proc. 15th Conf. on USENIX Security Symposium*, pages 305–320. 88

[82] Perez-Botero, D. (2011). A brief tutorial on live virtual machine migration from a security perspective. *University of Princeton, USA*, page 8. 70

[83] Petcu, D., Craciun, C., and Rak, M. (2011). Towards a cross platform cloud api. In *1st International Conference on Cloud Computing and Services Science*, pages 166–169. 43

[84] Petcu, D., Di Martino, B., Venticinque, S., Rak, M., Máhr, T., Lopez, G. E., Brito, F., Cossu, R., Stopar, M., Šperka, S., et al. (2013). Experiences in building a mosaic of clouds. *Journal of Cloud Computing: Advances, Systems and Applications*, 2(1):12. 43

[85] Rocha, F. and Correia, M. (2011). Lucy in the sky without diamonds: Stealing confidential data in the cloud. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 129–134. IEEE. 101, 102, 120, 124

[86] Roussev, V., Ahmed, I., and Sires, T. (2014). Image-based kernel fingerprinting. *Digital Investigation*, 11:S13–S21. 87

[87] Roussey, B. (2019). The 8 most in-demand cybersecurity skills for 2019. http://techgenix.com/in-demand-cybersecurity-skills/. Accessed on 26 April 2019. 1

[88] Rubens, P. (2019). 2019 it security employment outlook: The hottest skills and markets. https://www.esecurityplanet.com/network-security/2019-it-security-employment-outlook.html. Accessed on 26 April 2019. 1

[89] Rutkowska, J. (2009). Evil maid goes after truecrypt! https://theinvisiblethings.blogspot.de/2009/10/evil-maid-goes-after-truecrypt.html. Accessed on 23 April 2019. 100

[90] Sadeghi, A.-R., Stüble, C., and Winandy, M. (2008). Property-based tpm virtualization. In *International Conference on Information Security*, pages 1–16. Springer. 122, 126

[91] Saleh, E., Sianipar, J., Takouna, I., and Meinel, C. (2014). Secplace: A security-aware placement model for multi-tenant saas environments. In *2014 IEEE 11th Intl Conf on Ubiquitous Intelligence and Computing and 2014 IEEE 11th Intl Conf on Autonomic and Trusted Computing and 2014 IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops*, pages 596–602. IEEE. 53

[92] Saleh, E., Takouna, I., and Meinel, C. (2013). Signedquery: Protecting users data in multi-tenant saas environments. In *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 213–218. IEEE. 29

[93] Santos, N., Gummadi, K. P., and Rodrigues, R. (2009). Towards trusted cloud computing. *HotCloud*, 9(9):3. 121

[94] Schenk, E. and Guittard, C. (2011). Towards a characterization of crowdsourcing practices. *Journal of Innovation Economics Management*, (1):93–107. 45

[95] Schwarz, M., Weiser, S., Gruss, D., Maurice, C., and Mangard, S. (2017). Malware guard extension: Using sgx to conceal cache attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 3–24. Springer. 89

[96] Security, H. (2019). Moving target defense. https://www.dhs.gov/science-and-technology/csd-mtd. Accessed on 23 April 2019. 101, 105

[97] Selimi, M., Cerdà-Alabern, L., Wang, L., Sathiaseelan, A., Veiga, L., and Freitag, F. (2016). Bandwidth-aware service placement in community network micro-clouds. In *2016 IEEE 41st Conference on Local Computer Networks (LCN)*, pages 220–223. IEEE. 53

[98] Seo, K.-T., Hwang, H.-S., Moon, I.-Y., Kwon, O.-Y., and Kim, B.-J. (2014). Performance comparison analysis of linux container and virtual machine for building cloud. *Advanced Science and Technology Letters*, 66(105-111):2. 18

[99] Sharma, P., Chaufournier, L., Shenoy, P., and Tay, Y. (2016). Containers and virtual machines at scale: A comparative study. In *Proceedings of the 17th International Middleware Conference*, page 1. ACM. 18

[100] Sherry, L. (1995). Issues in distance learning. *International journal of educational telecommunications*, 1(4):337–365. 1

[101] Sianipar, J., Willems, C., and Meinel, C. (2017a). Signed url for an isolated web server in a virtual laboratory. In *Proceedings of the 2017 9th International Conference on Education Technology and Computers*, pages 218–222. ACM. 76

[102] Sianipar, J., Willems, C., and Meinel, C. (2017b). Team placement in crowd-resourcing virtual laboratory for it security e-learning. In *Proceedings of*

the 2017 International Conference on Cloud and Big Data Computing, pages 60–66. ACM. 74, 78

[103] Sianipar, J. H., Willems, C., and Meinel, C. (2016). Crowdresourcing virtual laboratory architecture on hybrid cloud. In *INTED2016 Proceedings*, 10th International Technology, Education and Development Conference, pages 2940–2949. IATED. 51

[104] Sim, K. M. (2012). Agent-based cloud computing. *IEEE Transactions on services computing*, 5(4):564–577. 121

[105] Simmons, P. (2011). Security through amnesia: a software-based solution to the cold boot attack on disk encryption. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 73–82. ACM. 102

[106] Simpson, O. (2013). *Supporting students in online open and distance learning*. Routledge. 1

[107] Singh, G. and Gupta, P. (2016). A review on migration techniques and challenges in live virtual machine migration. In *2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*, pages 542–546. IEEE. 70

[108] Subutai (2019). Peer-to-peer (p2p) cloud computing. https://subutai.io/p2p-cloud.html. Accessed on 2019-04-23. 45

[109] Sun, W., Katta, V., Krishna, K., and Sekar, R. (2008). V-netlab: An approach for realizing logically isolated networks for security experiments. *CSET*, 8:1–6. 38

[110] Suneja, S., Isci, C., de Lara, E., and Bala, V. (2015). Exploring vm introspection: Techniques and trade-offs. In *Acm Sigplan Notices*, volume 50, pages 133–146. ACM. 88

[111] Swami, Y. (2017). Intel sgx remote attestation is not sufficient. *IACR*. 88

[112] Tordsson, J., Montero, R. S., Moreno-Vozmediano, R., and Llorente, I. M. (2012). Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future generation computer systems*, 28(2):358–367. 53

[113] Trautman, L. J. and Ormerod, P. C. (2016). Corporate directors' and officers' cybersecurity standard of care: The yahoo data breach. *Am. UL Rev.*, 66:1231. 1

[114] Treaster, M. (2005). A survey of fault-tolerance and fault-recovery techniques in parallel systems. *arXiv preprint cs/0501002*. 69

[115] Tully, J. (2008). An anti-reverse engineering guide. https://www.codeproject.com/Articles/30815/An-Anti-Reverse-Engineering-Guide. Accessed on 23 April 2019. 102

[116] Turner, P., Engineer, S. S., and Infrastructure, T. (2018). Retpoline: a software construct for preventing branch-target-injection. 104

[117] University of West London (2019). Virtual cyber security lab on clouds. https://www.uwl.ac.uk/academic-schools/computing/doctoral-research/virtual-cyber-security-lab-clouds. Accessed on 24 April 2019. 38

[118] Unuvar, M., Steinder, M., and Tantawi, A. N. (2014). Hybrid cloud placement algorithm. In *2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, pages 197–206. IEEE. 53

[119] Usmani, Z. and Singh, S. (2016). A survey of virtual machine placement techniques in a cloud data center. *Procedia Computer Science*, 78:491–498. 52

[120] UTSA (2019). http://www.nationalccdc.org/. Accessed on 24 April 2019. 38

[121] Valeur, F., Vigna, G., Kruegel, C., and Kirda, E. (2006). An anomaly-driven reverse proxy for web applications. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 361–368. ACM. 29

[122] Virtual Hacking Labs (2019). Virtual hacking labs. https://www.virtualhackinglabs.com/. Accessed on 24 April 2019. 39

[123] Willems, C., Dawoud, W., Klingbeil, T., and Meinel, C. (2009). Security in tele-lab—protecting an online virtual lab for security training. In *2009 International Conference for Internet Technology and Secured Transactions,(ICITST)*, pages 1–7. IEEE. 12

[124] Willems, C., Klingbeil, T., Radvilavicius, L., Cenys, A., and Meinel, C. (2011). A distributed virtual laboratory architecture for cybersecurity training. In *2011 International Conference for Internet Technology and Secured Transactions*, pages 408–415. IEEE. 11, 12

[125] Willems, C. and Meinel, C. (2008). Tele-lab it-security: an architecture for an online virtual it security lab. *International Journal of Online Engineering (iJOE)*, 4(2):31–37. 12

[126] Willems, C. and Meinel, C. (2011). Practical network security teaching in an online virtual laboratory. In *Proceedings of the International Conference on Security and Management (SAM)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer 3, 9

[127] Wurzinger, P., Platzer, C., Ludl, C., Kirda, E., and Kruegel, C. (2009). Swap: Mitigating xss attacks using a reverse proxy. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems*, pages 33–39. IEEE Computer Society. 29

[128] Xu, J., Guo, P., Zhao, M., Erbacher, R. F., Zhu, M., and Liu, P. (2014a). Comparing different moving target defense techniques. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 97–107. ACM. 105

[129] Xu, L., Huang, D., and Tsai, W.-T. (2014b). Cloud-based virtual laboratory for network security education. *IEEE Transactions on Education*, 57(3):145–150. 38, 39

[130] Yarom, Y. and Falkner, K. (2014). Flush+ reload: A high resolution, low noise, l3 cache side-channel attack. In *USENIX Security Symposium*, pages 719–732. 104

[131] Yu, A., Qin, Y., and Wang, D. (2011). Obtaining the integrity of your virtual machine in the cloud. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 213–222. IEEE. 88

[132] Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18. 101

[133] Zhong, R. (2018). Quora, the q. and a. site, says data breach affected 100 million users. https://www.nytimes.com/2018/12/04/technology/quora-hack-data-breach.html, accessed on 29 June 2019. 1

Acronyms

AIDE Advance Intrusion Detection Environment

API Application Programming Interface

AS Autonomous System

ASN Autonomous System Number

BIOS Basic Input/Output System

CBVL Container-Based Virtual Laboratory

CLI Command Line Interface

CPU Central Processing Unit

CRVL Crowd-Resourcing Virtual Laboratory

CTPlace An algorithm for Team Placement in a crowd-resourcing virtual laboratory

Docker A container technology for Linux that allows a developer to package up an application with all of the parts it needs

FSB Front Side Bus

GateOne An HTML5-powered terminal emulator and SSH client

GUI Graphic User Interface

HPI Hasso Plattner Institute

HTML Hyper Text Markup Language

HTTP Hypertext Transfer Protocol

ICMP Internet Control Message Protocol

IDS Intruder Detection System

IP Internet Protocol

Iptables A user-space utility program to configure the tables provided by the Linux kernel firewall and the chains and rules it stores.

JADE Java Agent Development Environment

kexec a system call to load and boot into another kernel from the currently running kernel

LAN Local Area Network

LFF Least-Full-first

MBR Master Boot Record

MFF Most-Full-First

MITM Man In The Middle

NAT Network Address Translation

Nmap Network Mapper, a free and open-source network scanner

Node.js An open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser

Open vSwitch An open-source implementation of a distributed virtual multi-layer switch

OpenNebula A cloud computing platform for managing heterogeneous distributed data center infrastructures

OS Operation System

P2P Peer-to-peer

PoC Proof of Concept

RAM Random Access Memory

SGX Software Guard Extensions

SSH Secure Shell

TCP Transmission Control Protocol

ACRONYMS

Team An isolated laboratory environment consists of VMs or Containers and virtual switch that was configured and built for a participant to practice a specific cybersecurity training scenario.

Tele-Lab Virtual Laboratory for IT Security e-Learning platform

TPM Trusted Platform Module

UDP User Datagram Protocol

vCPU virtual Central Processing Unit

VirtualBox A free and open-source hosted hypervisor for x86 virtualization, developed by Oracle Corporation

VM Virtual Machine

VPC Virtual Private Cloud

VPN Virtual Private Network

VTE Virtual Training Environment

Websocket A communications protocol, providing full-duplex communication channels over a single TCP connection.

XML-RPC Extensible Markup Language Remote Procedure Call

Xprobe2 A remote active OS fingerprinting tool