vLab: A Cloud based Resource and Service Sharing Platform

for

Computer and Network Security Education

by

Aniruddha Kadne

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2010 by the
Graduate Supervisory Committee:

Dijiang Huang, Chair
Wei-Tek Tsai
Gail-Joon Ahn

ARIZONA STATE UNIVERSITY

December 2010
ABSTRACT

Cloud computing systems fundamentally provide access to large pools of data and computational resources through a variety of interfaces similar in spirit to existing grid and HPC resource management and programming systems. These types of systems offer a new programming target for scalable application developers and have gained popularity over the past few years. However, most cloud computing systems in operation today are proprietary and rely upon infrastructure that is invisible to the research community, or are not explicitly designed to be instrumented and modified by systems researchers. In this research, Xen Server Management API is employed to build a framework for cloud computing that implements what is commonly referred to as Infrastructure as a Service (IaaS); systems that give users the ability to run and control entire virtual machine instances deployed across a variety physical resources. The goal of this research is to develop a cloud based resource and service sharing platform for Computer network security education a.k.a Virtual Lab.
To My Beloved Family
ACKNOWLEDGMENTS

I would like to express my gratitude to Dr. Dijiang Huang for giving me an opportunity to work on this cutting-edge research topic and for providing me with valuable guidance, encouragement and support. I remain highly obliged to Dr. We-Tek Tsai and Dr. Gail-Joon Ahn for the useful ideas and feedback they gave as part of my thesis committee.

It has been a wonderful experience working with all my fellow researchers in the Secure Networking and Computing (SNAC) Group and I would like to thank each one of them for helping and making the whole process of research so much fun. I would like to thank all my friends and teachers for making it all worthwhile.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF FIGURES</th>
<th>ix</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1  INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Background</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Organization</td>
<td>2</td>
</tr>
<tr>
<td>2  MOTIVATION AND SCOPE OF THE RESEARCH</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Survey and Analysis</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Proposed Solution</td>
<td>10</td>
</tr>
<tr>
<td>2.2 Technical Merits</td>
<td>11</td>
</tr>
<tr>
<td>3  SYSTEM ARCHITECTURE</td>
<td>13</td>
</tr>
<tr>
<td>3.1 Overview</td>
<td>13</td>
</tr>
<tr>
<td>3.1.1 Resource Layer</td>
<td>14</td>
</tr>
<tr>
<td>3.1.2 Cloud Layer</td>
<td>15</td>
</tr>
<tr>
<td>3.1.3 Application Layer</td>
<td>15</td>
</tr>
<tr>
<td>3.2 Role of Virtualization</td>
<td>16</td>
</tr>
<tr>
<td>3.2 Example Network Scenario</td>
<td>18</td>
</tr>
<tr>
<td>3.4 Summary</td>
<td>20</td>
</tr>
<tr>
<td>4  SYSTEM COMPONENTS</td>
<td>23</td>
</tr>
<tr>
<td>4.1 Web-based Portal</td>
<td>23</td>
</tr>
<tr>
<td>4.2 UserSpace Applet</td>
<td>25</td>
</tr>
<tr>
<td>4.3 virtualLabPortClient</td>
<td>27</td>
</tr>
<tr>
<td>CHAPTER</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>4.4 virtualLabPortServer</td>
<td>28</td>
</tr>
<tr>
<td>4.5 virtualLabPortConfigurationServer</td>
<td>30</td>
</tr>
<tr>
<td>4.5.1 Network scripts for VMs</td>
<td>30</td>
</tr>
<tr>
<td>4.5.2 Configuration scripts for VMs</td>
<td>31</td>
</tr>
<tr>
<td>4.5.3 IPTABLES rules for VMs</td>
<td>33</td>
</tr>
<tr>
<td>5 SYSTEM DESIGN</td>
<td>34</td>
</tr>
<tr>
<td>5.1 Web server- Database Subsystem</td>
<td>34</td>
</tr>
<tr>
<td>5.2 Cloud Subsystem</td>
<td>35</td>
</tr>
<tr>
<td>5.3 Virtual Lab Administration Subsystem</td>
<td>35</td>
</tr>
<tr>
<td>5.4 Putting It All Together</td>
<td>36</td>
</tr>
<tr>
<td>6 SECURITY ANALYSIS</td>
<td>39</td>
</tr>
<tr>
<td>6.1 XenServer Security</td>
<td>39</td>
</tr>
<tr>
<td>6.2 Virtual Lab Setup Security</td>
<td>40</td>
</tr>
<tr>
<td>6.3 Attacker on Internal Network</td>
<td>41</td>
</tr>
<tr>
<td>6.4 Attackers on External Network</td>
<td>41</td>
</tr>
<tr>
<td>7 PERFORMANCE EVALUATION</td>
<td>43</td>
</tr>
<tr>
<td>7.1 Hardware Platform</td>
<td>43</td>
</tr>
<tr>
<td>7.2 Sample Experimental Setups</td>
<td>44</td>
</tr>
<tr>
<td>7.2.1 Simple Network Configuration</td>
<td>44</td>
</tr>
<tr>
<td>7.2.2 Firewall Configuration</td>
<td>44</td>
</tr>
<tr>
<td>7.2.1 Web-server SSL, Authentication</td>
<td>45</td>
</tr>
<tr>
<td>7.2.1 Intrusion Detection</td>
<td>45</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1.</td>
<td>System Architecture Overview</td>
</tr>
<tr>
<td>2.</td>
<td>Example Network Scenario</td>
</tr>
<tr>
<td>3.</td>
<td>Virtual Lab Web Portal</td>
</tr>
<tr>
<td>4.</td>
<td>Userspace Applet</td>
</tr>
<tr>
<td>5.</td>
<td>Configuration Options Provide by Virtual Lab GUI</td>
</tr>
<tr>
<td>6.</td>
<td>virtualLabPortClient-Server Communication</td>
</tr>
<tr>
<td>7.</td>
<td>Code Snippet of Configuration File</td>
</tr>
<tr>
<td>8.</td>
<td>System Design</td>
</tr>
<tr>
<td>9.</td>
<td>Graphical overview of API classes for managing VMs, Hosts, Storage and Networking</td>
</tr>
<tr>
<td>10.</td>
<td>OSI Model</td>
</tr>
<tr>
<td>11.</td>
<td>Simple LAN over Ethernet</td>
</tr>
<tr>
<td>12.</td>
<td>Ethernet Frame Structure</td>
</tr>
<tr>
<td>13.</td>
<td>Multiple Ethernet Segments</td>
</tr>
<tr>
<td>14.</td>
<td>MAC address learning on Ethernet segment</td>
</tr>
<tr>
<td>15.</td>
<td>Multiple LANs in different collision domains</td>
</tr>
<tr>
<td>16.</td>
<td>Bridged Ethernet Segments</td>
</tr>
<tr>
<td>17.</td>
<td>XenServer Internal Network</td>
</tr>
<tr>
<td>18.</td>
<td>XenServer External Network</td>
</tr>
<tr>
<td>19.</td>
<td>Virtual Switch Inside XenServer</td>
</tr>
<tr>
<td>20.</td>
<td>XenServer Internal and External networks together</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>21.</td>
<td>XenServer Virtual Interfaces (VIFs)</td>
</tr>
<tr>
<td>22.</td>
<td>XenServer VLANs</td>
</tr>
<tr>
<td>23.</td>
<td>VLAN Trunking</td>
</tr>
<tr>
<td>24.</td>
<td>VM Migration</td>
</tr>
<tr>
<td>25.</td>
<td>NIC Bonds</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

1.1 Background

Cloud computing systems fundamentally provide access to large pools of data and computational resources through a variety of interfaces similar in spirit to existing grid and HPC resource management and programming systems. These types of systems offer a new programming target for scalable application developers and have gained popularity over the past few years. However, most cloud computing systems in operation today are proprietary, rely upon infrastructure that is invisible to the research community, or are not explicitly designed to be instrumented and modified by systems researchers. In this research, we employ XenServer Management API – to build a framework for cloud computing that implements what is commonly referred to as Infrastructure as a Service (IaaS); systems that give users the ability to run and control entire virtual machine instances deployed across a variety physical resources. The goal of this research is to create a cloud based resource and service sharing platform for Computer network security education, a.k.a Virtual Lab.

This project attempts to implement a virtual lab using the concept of cloud computing and virtualization. The cloud system in this project has been built using the Citrix Xen Server Management API (hereafter referred to as just "API"). Xen is a virtual machine hypervisor that allows several
guest operating systems to execute on the same computer hardware concurrently. The University of Cambridge Computer Laboratory developed the first versions of Xen. The Xen community develops and maintains Xen as free software; licensed under the GNU General Public License (GPLv2). Top level components of this, aggregate resources from multiple clusters i.e. collections of nodes sharing a LAN segment, possibly behind a firewall. Each cluster is controlled by a cluster controller for cluster-level scheduling and network control and a storage control for EBS-style block-based storage. It deploys its instances i.e. virtual machines on XEN hypervisors.

The prime objective of this research is to develop a resource and service sharing platform that provides segregated virtual systems for end users to build their own networking environments. In addition to this, it also aims at developing a user-friendly, easy-to-use graphical user interface that enables users to request for their virtual test environments remotely, just in similar fashion to accessing an email account over internet connection. This eliminates the time and location constraint to use this platform, which contributes towards increased productivity of the end users. We also plan to collaborate with multiple-university laboratory environment in future.

1.2 Organization

The rest of the document is organized as follows. Chapter 2 discusses about the motivation and scope of this research. Chapter 3
gives overview of the architecture of the platform. Chapter 4 describes
details of the basic building blocks of the implementation. Chapter 5 briefly
outlines the higher view of the system components and their interaction
with each other. Chapter 6 provides an analysis of the inherent security of
this platform. Chapter 7 conducts performance evaluation of the set up
and hardware devices. Chapter 8 presents the related work done in this
area. Chapter 9 outlines some of the features and enhancements that can
be made to the existing implementation. Interested readers can refer to
the Appendix A for more details about the cloud management software
a.k.a XenServer 5.6 used to build this tool.
CHAPTER 2

MOTIVATION AND SCOPE OF THE RESEARCH

Past few years have experienced tremendous rise and popularity of technologies like cloud computing and virtualization. Many organizations have already started implementing these technologies to further reduce costs through improved resource utilization. In essence, the combination of cloud computing and virtualization enables users to use applications on internet just as if they are browsing any other websites.

Computer and network Security (hereafter referred to as CNS) is an important area for undergraduate computer science and engineering education and it has been recognized as an area of national interest to strengthen our cyber infrastructure. To impart highest level of education facilities to the enrolled students, Arizona state University established a national Center of Academia Excellence in Information Assurance Education (CAE/ISE) certified by National Security Agency (NSA) and Department of Homeland Security (DHS). The most important part of learning every computer science course is the examination and experimentation with the technology. Almost every course that is offered in the field of computer science demands student to perform experiments in the lab set up to enrich their experience of getting so called 'hands-on'. This requires access to lab equipment that either most student don't have or have for very limited amount of time due to being shared with other classmates in order to keep overall cost of the set up within budget.
CNS offers a fantastic opportunity for strengthening our science, technology, engineering and mathematics (STEM) areas. It is a well known fact that United States does not produce enough computer security experts. One source of the problem is many universities do not have adequate hands-on experiment test beds that can effectively train students in computer security. As a result, students tend to lose interest very quickly since they do not get to actually perform the experiments by configuring the real devices.

2.1 Survey and Analysis

Our survey and analysis of the requirements suggests that any university that offers such courses faces three common limitations. These limitations are discussed in the following literature.

*Lack of computer networking equipments and space for various computer network settings* is the most common limitation faced by almost every university in the United States. For example, Arizona State University’s CNS laboratory consists of 21 Pentium IV desktops, one 48-port Cisco Switch and 4 Dell servers. We use desktops to emulate networking devices, such as routers, gateways and servers by running open-source software. This approach functions adequately when the emulated network size is small and the attack and countermeasure models are simple. For example, if we conduct a packet filter firewall, we require three computers: one is located in an internal network, one serves for the packet filter firewall and one serves for the outside server, which
can be shared by multiple project groups. Thus, at the most 10 groups of students can work on the packet filter firewall project simultaneously. However, for students to experiment with a firewall system including an interior/exterior router, DMZ and local network, the current configuration can support no more than 2 groups of students simultaneously.

Installing additional computers is usually not a viable solution due to the budget and laboratory space limitation. Instead, virtualization techniques are being used to run multiple computer images on one physical computer, thereby increasing the number of available computers. This approach, however, continues to have two limitations. First, we utilize ASU’s surplus computers to set up their CNS-IL, which is a cost-effective way for handling computing resources because most institutions regularly replace their old computers. Moreover, these surplus computers usually do not have powerful CPUs and adequate memory, thus putting limits on the number of running images. For example, in the CNS-IL, each computer has a 512MB memory, 40G hard drive, and 2.16GHz Pentium IV processor. In Spring 2006, however, based on student use and testing, it was found that the host computers were too slow to support more than two running images for most experiments. A second problem associated with use of additional computers is that the physical network barrier restricts the use of virtualization techniques, thus making the network setup inflexible. If we configure virtual machines (VMs) on the same physical computer, they can be interconnected through internal virtual
networks, e.g., VMware [2] and Xen [3] virtualization techniques, which can provide multiple internal host-only networks to emulate physically separate networks. However, if VMs are running on different host systems, the inter-connections among these VMs must go through a physical network switch. Thus, to separate the traffic between them from other VMs connected through the same physical switch, virtual LANs must be established in the switch, which makes it inflexible to reconfigure the networks. Moreover, the separation of each host system also creates a starvation situation, in which some computers can be overloaded by running many VMs while some are under-utilized at the same time.

Thus, a cost-effective and flexible virtualization solution is required, i.e., multiple virtual machines need to be created and managed via a cluster of physical machines, where the workload is uniformly shared among different physical systems in order to circumvent the starvation problem. This in turn avoids the situation where VMs are mapped to dedicate physical hosts statically. We call this virtualization capability as ‘clustered’ or ‘networked’ virtualization.

The next major limitation to set up such labs is the *Inability to simultaneously support projects for different IA courses*. Various courses offered to enhance the knowledge about network security; information assurance, data and computer system security, computer and network forensics etc involve more than 300 students in a given semester. Since, at least two of such courses will be offered during a single semester,
scheduling student access to such labs becomes a challenging issue since students involved is large but number of computers and laboratory space is limited. This issue is further compounded by the high security requirements of CNS-IL because the laboratory should be physically segregated as security attacks are experimented and such attacks cannot be propagated to the other systems in the public domain.

Another serious problem of current CNS-ILs is that IA courses usually require different computer and network configurations. Setting up separate laboratories for different IA classes will be able to reduce the network and system reconfigurations. we present the following three desired features for establishing a CNS-IL. (a) A CNS-IL should be accessed remotely to avoid the access schedule problem. To enable the remote access to the CNS-IL, traffic initiated from CNS-IL to the public domain must be disabled. This will prevent bad or testing traffic from entering the public network; (b) A CNS-IL should be able to share its idle computing and network resources with other CNS-IL users. This requires establishing an Alliance for students to be able to utilize the resources belonging to different CNS-ILs across boundaries of institutions. This will enable maximum utilization of well established CNS-IL resources; and (c) A standard procedure is needed to request/establish/destroy a test environment, which will require building a repository for indexing available security testing environments in different CNS-ILs.
Lack of an infrastructure to manage and utilize existing computer and network security experiments is the third critical limitation of setting up such CNS labs. Computer and network security is a rapidly developing area with numerous security defense and attack experiments and tools being proposed on a daily basis. Most security training requires that students perform hands-on exercises and experiments to truly appreciate the materials covered. Furthermore, institutions usually focus on a broad range of IA areas based on instructor expertise and existing lab resources. Often instructors have their own syllabus and project experiments that focus on different aspects of CNS. While adopting other institution’s CNS course materials and corresponding projects will ease the preparation of the course and enrich CNS education, the adoption process is inherently limited by the existing infrastructure, such as courseware setup, software deployments, and laboratory facilities. In addition, many universities also do not have student laboratories to practice CNS. Thus, it is desirable to establish an educational resource that includes a well-defined CNS infrastructure equipped with case studies, instructional materials, and test banks to enable other institutions to share and contribute related resources. This project proposes to develop a new CNS-IL infrastructure using the latest computing technology, i.e., Cloud computing to maximize the utilization of computer network resources and to create an educational database of attacks and defenses.
2.2 Proposed Solution

Taking into consideration the above mentioned limitations, we propose a new Computer network security laboratory platform that has following capabilities. The CNS-IL lab is designed to be virtualized in a clustered computing environment controlled by Cloud computing to fully utilize the computing and storage. The cloud will provide dynamic resource allocation to ensure effective system utilization through carefully monitoring of resource workload and available physical resources, and providing managed solutions on a continual basis. The most important aspect of this research is to develop a platform that students can access remotely without time and location constraints. Registered instructors will use, share and contribute the course materials and laboratory experiments in a collaborative manner. Registered student can set up their security testing environment by utilizing resources provided by the Cloud. The Cloud provides a segregated virtual environment for each participant, with the experiment confined to a virtual system, thus not affecting any system outside of the cloud as well as other virtual systems within the cloud.

To realize these capabilities we have identified three critical design facets to be included in our architecture. Firstly, and as discussed earlier in this document, our system was designed keeping Cloud at its core.. We are using cutting-edge Cloud computing software to monitor and provide virtual resources dynamically via a cluster of physical resources including computers, routers, gateways and firewalls. The Cloud system controls
networks, CPU memory and hard drives and provides segregated partitions so that each partition will not and cannot interfere with each other. The system will also confine any security attacks with the segregated partition only, preventing cyber attacks spilling over to public systems or networks. The second critical features that needs to be incorporated in the system is the ability to provide remote access to the registered users. Since the resources will provide security testing experimentation, it makes sense to provide access and benefits only to the registered users. Ability to publish and share the materials including lecture notes, test databases and blogs is regarded as an equally important feature during the development of this project. The cloud allows instructors and students to publish and share the material. The proposed cloud will host material publication contributed by all parties; however the materials will be evaluated by a committee to ensure relevancy, correctness and proper presentation.

2.3 Technical Merits

We attempt to establish a virtual Computer and network Security that incorporates a set of hands-on course projects into computer science and engineering undergraduate education. The proposed laboratory design can hugely reduce the learning curve of students. Our proposed research is characterized by following intellectual merits:

- It provides a virtual computer network environment with complete control of network nodes from the physical layer to the application
layer, which can greatly reduce network establishment overhead and provide desired level of fidelity for network security experiments.

- It can adopt clustering techniques to make efficient use of distributed computing resources thereby emulating the real computer networking experiments, in the best possible realistic manner, to test complicated network and attack scenarios.

- It inherently provides Infrastructure-as-a-Service (IaaS) that allows users to interface easily with others’ implementations and share development results.

- A rich set of experiment files and related course materials will be shared following the Web 2.0 principles through this project, which allows instructors to easily adopt new laboratory developments in security-based curricula with little learning and preparation overhead.

- This architecture is designed as a cost-effective solution using open-source software. It adopts clustering technologies that utilizes surplus college computing resources. Thus, it can be easily set up in colleges with limited budget and laboratory space.

- It provides an educational platform that allows researchers and students to experiment and learn a variety of modern technologies such as Cloud computing, wireless applications etc.
CHAPTER 3
SYSTEM ARCHITECTURE

In this section we discuss about the system architecture of virtual lab platform. It briefly discusses about the underlying Cloud platform, physical network setup, how virtualization fits on top it and ultimately the resource and service sharing it offers.

3.1 Overall Architecture

![System Architecture Overview](image)

Figure 1. System Architecture Overview

As shown in the figure, the overall architecture of the system can be divided in three distinct sub layers viz. Resources layer at the heart of the system, providing bear resources in the form of CPUs, Memory, Network links, Storage repositories etc. Then there is a Cloud layer as an
intermediate layer, with full control over the Resources Layer and finally the Application layer, residing on top of Cloud to provide the services. It works exactly like any stacked architecture would work, with every layer providing some services to the top layer while using services provided by the layer below it. Users usually talk to the application layer, while Cloud layer and the Resource Layer typically are transparent to the users.

Let’s discuss about each of the three layers in brief:

3.1.1 Resources Layer

This layer is a collection of physical devices that include powerful computer nodes, Gigabit network switches and Terabyte storage spaces for shared file system. Through hypervisor, the physical resources are clustered as a single logical device with aggregated computing and resources capabilities. Multiple computing nodes are network bootable. Usually all the resources are mapped to a master computer (appointed by the administrator, either randomly or based on some powerful configurationally advantageous capability) which then becomes the identity of the one large logical resource. This scheme is typically called the pooling of the resources, where the pool-master is responsible for sharing the CPUs, RAMs and other computing resources of all other computing nodes in the pool in a transparent fashion. Once booted, the computers access their files on the master using the NFS. The images for the VMs are provided through NFS and are mounted as network block
devices. All the experiment traffic is restricted in the experiment network. The cloud management traffic is transmitted through the control network.

3.1.2 Cloud Layer

This layer hosts the Cloud control software, and is used to combine all resources including computers, network and software together to provide a virtual machine with application customized platform, which make users think that they are using a single customized machine. And this layer can reallocate users’ request and balance load of servers that make the servers more efficient.

Typically Cloud layers can be categorized based on the type of services they offer viz. Application-as-a Service (AaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS). As described earlier, our Cloud layer provides the IaaS.

In essence, the cloud layer is formed using Resource virtualization, Storage Virtualization and Workspace Virtualization. When all of them combined together, they provide the needed service virtualization.

3.1.3 Application Layer

This is the top-most layer of the system which is actually visible to the users. It seamlessly runs on top of the Resources layer and Cloud layer to relay the services offered by them to the actual users of the system. In this layer users can establish their own environments for experiments. Students will need to use the application template, workflow and VMs that the system provides. They can easily revise the application template and workflow to personalize it. The key advantage offered by this
layer is integration of virtualization technology in cloud computing and Web 2.0 principles where sharing is at the core. Through cloud computing, we can maximally utilize the computing resources through virtualization techniques; using Web 2.0, we can improve the effectiveness of education by sharing and collaboration.

3.2 Role Of Virtualization

As shown in the Figure 2, hardware virtualization is built on top of the bare-hardware resources present in the Resources Layer. This virtual toolbox includes a number of virtual machines (VMs) that can be configured as workstations, servers, routers and networking devices. We identified that Citrix XenServer 5.6 allows us to achieve this with minimum effort and maximum flexibility. XenServer cluster is used to establish the fundamental physical platform for computing resource sharing. VMs run on the cluster to emulate various devices. VMs are interconnected using a mesh of virtual networks inside a node in the cluster and using virtual LANs across various nodes of the cluster. We discuss about this in greater details in Section. On top of this, in application layer, we use Web 2.0 principles to develop a web-based user interface and a web-based management interface, through which users can remotely access the system platform and perform their projects, just like sitting in front of the physical devices. Underlying are the networking and computing resource monitoring and management tools for dynamic resource allocation.
The proposed system provides inhabitation for many virtual components implemented at system level i.e. servers, clients, gateways, routers and firewalls are built on independent VMs with the full capability of an operating system. Internal networks are created by mapping the interface of a VM to appropriate virtual switch and thereby to the physical interfaces. Both virtual components (workstations, servers, routers, firewalls etc) can be requested by user through our specially designed Graphical User Interface (GUI), which allows users to create virtually every possible network topology that exists in present day world. The proposed approach also provides an option of creating topology templates for mass usage like that for class projects, in which case, an instructor can pre-configure the network topology and create a template for it. This template can then be used by the entire class to perform some fairly complicated network security experiments. In other case, users can create their own network topologies on a user-space provided on the GUI. In this way students of the class can quickly set up the testing environment by applying for the lease of the corresponding VMs and VNs. Since all VMs are initiated from an NFS repository over the network and they are run through our established clusters, multiple VMs can run directly within the same cluster and thus we can ease network configurations and reduce the scenario set up time. We discuss about the capability, working and features of this GUI in Section.
3.3 Example Network Scenario

Let us discuss an example network scenario which a user is trying to lay on the proposed virtual lab platform. Here a user wants to establish a security testing environment. Suppose the user requests the cloud to establish a network to test a denial of service (DoS) attack based on flooding of requests. To make the attack successful, the user needs to deploy Network Mapping to search available hosts; and then he may need to perform PortScanning to search vulnerable open TCP ports on the found hosts. To tests the robustness of the given network system to this attack, the user also needs to evaluate the packet filter firewall installed in gateway. In this experiment, the user requests an experiment scenario including at least two networks – one external network and one internal network that are separated by a gateway. The host is located in internal network and the attacker is located in the external network. The requested applications are packet builder, Nmap, ping, traceroute on the user’s computer, and iptables on the gateway.

![Diagram](image)

Figure 2. Example scenario

Figure 3 demonstrates the steps followed to request the desired network scenario from the proposed virtual lab platform.
1) The user remotely accesses to the Cloud and sends an experiment request, i.e., a service description for the requested experiment scenario, to the Cloud service broker running in the administration server.

2) The Cloud service broker checks the system repository managed by the scenario server on available memory, VNs, and CPU time in the cluster, and decides if the user’s request can be satisfied.

3) Based on the requested resources, available VMs stored in NFS VM repository are registered in the system repository.

4) The requested applications are also registered in the system repository. If the invoke VM does not contain the requested applications, the user can install them through the NFS service repository after the experiment environment being established.

5) The system repository returns the experiment scenario profile to the Cloud service broker.

6) The Cloud service broker needs to decide if the requested services should be composed according to the user’s role.

7) The experiment scenario composer running in the scenario server initiates the requested VMs and VNs. Once the experiment scenario is established, it reports to the Cloud service broker.

8) The Cloud service broker returns the “service complete” to the user.

9) Finally, the user starts the security experiment on the established system.
This testing network can be easily established by getting resource service from the system repository and service repository to allow VMs, VNs, and applications to be composed together, and then create a multi-hop networking scenario for the computer and network security test. After the experimentation, the user publishes the experiment scenario profile to allow others to reuse it if it has not been published before. Our proposed solution is flexible in that the user can specify options when requesting the service from the Cloud. For example, if the user specifies the software development option to “true”, no applications will be allocated to the VM, and the user needs to develop the corresponding attacks or countermeasures instead of using available applications. Also the user can reserve VMs for his use only.

3.4 Summary
The proposed virtual lab platform is essentially a form of distributed computing that allows the users the ability to take advantage of vast network of computing resources through the Web to complete their projects. If, for example, a user wants to set up a client server communication over a multiple hop network with the client located behind a fire walled gateway, she can initiate $n$ network devices i.e. a web-server, multiple routers, served as intermediate hops, a gateway hosting firewall and the client behind it. This experimental environment can be provided by the cloud based on available VM and network resources. Additionally, the cloud computing technology allows user to know ahead in time, how much
computing capability is available based on some system wide statistics maintained in the cloud management controller. In addition to this, the user can also configure a VM that exists within the cloud to meet the particulars of the jobs they are trying to accomplish. Plus, user can also go to different clouds and recreate the system needed to get the jobs done, making computational power of a commodity.

The broader impacts of this research will be realized across several aspects. This project will impact students at the undergraduate level via related courses in software engineering, computer networking and security, and distributed computing. Since this is an instructional laboratory improvement plan, its research components will also involve various issues surrounding network protocol monitoring and diagnosis, which can then be extended to our ongoing capstone projects and attract more undergraduate students to network security, software service architecture, distributed and Cloud computing. Our proposed laboratory improvements address two of the Arizona’s most critical pipelines issues expressed by Professor Jim Middleton, associate senior vice provost for STEM education at ASU. According to Middleton one bottleneck for STEM entry is the low enrollment in science and math subjects at the high school level and attrition of STEM majors at the undergraduate level [66]. Our project will motivate students to think about future careers in computer science and engineering areas, increase an awareness of computer and network security technologies, and improve the quality of CNS education.
We must note that the proposed Cloud-based CNS-IL is not just restricted in the area of CNS. It can be utilized broadly in all science and engineering areas. Our initial motivation of this project is to utilize the most advanced Cloud computing technologies to assist traditional classroom learning and hence stimulate students' interests in the STEM areas. Eventually, we expect the CNS-IL will set up a laboratory model for all CNS-related education that incorporates hands-on projects with in-classroom theory learning.
CHAPTER 4

SYSTEM COMPOENETS

This section mainly focuses on discussion about the implementation details of the proposed Virtual Lab platform.

The implementation of this thesis takes advantage of the features offered by variety of the programming languages, as appropriate. It involves development of number of modules in programming languages like ASP.NET framework C-sharp, Java, AWT, MS-SQL server, Socket programming APIs in Java, bash scripting and networking tools/configuration. The major part of the effort towards this thesis was devoted in developing following fundamental modules:

- Web-based portal
- UserspaceApplet
- virtualLabPortClient
- virtualLabPortServer
- virtualLabconfigurationServer
- AdminScripts

Putting it all together, Virtual Lab platform is built around 2000 LOC in various programming languages stated above.

Let’s start discussing about these modules on one-by-one basis.

4.1 Web-based portal

This is what user sees once he connects to the virtual lab platform remotely using a web-browser that has access to a moderate speed
internet connection. Figure 4 shows user view of this web page which presents the information about this project on its home page.

Figure 3. Web Portal

This portal provides various options like user account management, web-based video tutorials, publications and related document downloads etc in addition to the critical user space that allows user to create her network topology and requests resources from the cloud in most user-friendly and comfortable manner. We will discuss about the user space in
greater detail in next section. Apart from this, this portal is just as similar to any other web site on the internet and thus we terminate our discussion about this module here. Keen users are requested to explore various options provided by this portal by actually visiting the web-page.

4.2 UserspaceApplet

This section primarily describes the features of user space interface, leaving the implementation details to subsequent sections to be discussed. The prime objective of this module is to provide users with a platform independent, user-friendly, transparent graphical interface which lets them create variety of multi-hop, multi-purpose network topologies and run the experimentation without the constraints of time and location. This interface allows users to access the network elements in pretty much a similar fashion as to access an email account over a moderate speed internet connection and a simple web browser. To access these terminals remotely, the client machine needs to have SSH client software installed. Figure 1 depicts an example that imitates a real world scenario where a 2-op network is created to join two different departments of an organization, residing on different floors of a fictitious corporate office. In this interface, user has the facility to draw links between any two nodes, create nodes of types desktop, switch or a router. Each type of node is identified by a distinct color to enhance the comprehension of the network graph. In running on the node, hard drive of the node and the primary memory
addition to the type, there is a provision to select the operating system

![Figure 4. Graphical User Interface of Virtual Lab](image)

![Figure 5. Configuration Options Provide by Virtual Lab GUI](image)

a.k.a. RAM of the selected node (Refer Figure 2). Every physical networking switch has 4 ports by default and you can not select other parameters like operating system, hdd, ram for it. In addition to this, user can also configure the network interfaces added in the node using similar
configuration window, where she can select an interface to assign a static IP address or a dynamic IP address obtained from a DHCP server running in the local network of the node. Once user creates her desire network, she clicks submit button which, after running a validation control for valid hostname strings, IP addresses and connection links, relays the user's request to the Virtual Lab infrastructure management.

This module is entirely developed in AWT APIs in Java programming language that takes the advantages of object-oriented programming features to its fullest. At its heart, it uses a data structure that maintains all the details about the graph being created on the screen by the user. It dynamically adds and removes the edges and nodes from the userSpace by sensing the mouse events performed by the user in userSpace. It also has the functionality to translate the pictorial view of the network topology created by the user into a unique data format that can be conveyed to the cloud controller, to actually request the resources from it.

4.3 virtualLabPortClient

This is a simple administrative component in the cloud management network, that communicates with the virtualLabPortServer (shown as Port server in Figure 5), on user's behalf, to get required number of port numbers from it. We postpone the discussion about the role played by these port numbers to a later section. This section describes the procedure to access the virtual machines (VMs) for the
user’s network topology once they are ready to be used. The workflow is better understood with the help of figure 5.

This module has been implemented using simple Socket programming API’s in Java to create a TCP connection with the virtualLabServer. Once the connection is established, it talks to the server to receive appropriate number of unused or released ports from the pool of available ports. On successful reception of the port numbers, it closes the connection and relays the output to the user-session where it is tagged to the corresponding virtual machine in the firewall.

4.4 virtualLabPortServer

This is a fairly complex server program that is designed based on the principles of classic multi-threaded web-server. Its major role is to maintain a data structure that hosts a pool of port numbers, used to tag each VM of the user’s network (Explained later in details). It needs to ensure that it provides a synchronized access for this data structure to all the threads which simultaneously try to request/release the ports number from/to the pool. The server creates a separate thread for each incoming request and continues to listen on the designated port for more requests. In this way, it is able to cater more than one requests at the same time, with the help of independently running, dedicated threads. This improves the performance of this module to greater extent. As shown in Figure 5, there might be another threads running in the network which try to access
the pool data structure at the same time as that of thread in the discussion. Thus, the pool data structure has to be protected by some means to prevent the race conditions.

This module is completely developed in Java programming language and the Socket APIs provided by it. It creates a data structure that maintains a stack of released ports so as to make an efficient use of the ports released by the other users during past sessions. The synchronized access to this data structure is achieved using code synchronization techniques offered by the Java programming language.

As mentioned earlier, the server creates a dedicated thread for each incoming request on the designated port and then it becomes the responsibility of this thread to serve the request of the client on behalf of
virtualLabPortServer. Once the request is served, depending upon its type (request or release), the thread dies automatically.

4.5 virtualLabConfigurationServer

This is by far the most important module of the Virtual lab platform. It performs variety of functions such as creating network scripts for the user virtual machines (VMs), contacting port server, preparing firewall rules etc. This acts as a starting-point of the automated activities carried out to prepare the user VMs to be accessed remotely in a secure fashion.

This module is implemented as a Java program wrapped inside a shell script residing on the dedicated machine in the Virtual lab network (shown as Administration Server in Figure 5). Its primary responsibility is to collect the information regarding user’s network topology, its configuration details like host name strings, static IP addresses assigned to the interfaces of virtual machines, routing rules to be introduced in intermediate routers and the dynamic firewall rules that need to be embedded in the firewall of the Virtual Lab platform.

The functionality implemented in this module can be roughly divided in three major tasks as follows:

4.5.1 Preparation of Network Scripts for the Virtual Machines

In this sub section, the module works upon the input parameters it receives from the wrapper shell script, containing the configuration details for all the virtual machines contained in the user’s network topology. It typically involves parsing of the input parameters, to search for the
network configuration details. Once the module identifies such details, it simply creates interface file contents for each user virtual machine.

4.5.2 Preparation of Configuration Scripts for the Virtual Machines

The network scripts prepared in the above step have to be copied to the appropriate user Virtual Machines and the networking daemon has to be restarted for it to take the effect. This sub section of the server takes care of generating such a script called ConfigScript residing in the home directory of the user on Administration Server. There will be one common ConfigScript, for all the user virtual machines, that holds all the configuration commands to be executed on a specific user virtual machine, in order to prepare it to be accessed remotely and at the same time connecting it to the other user virtual machines in the fashion specified by the user request. For example, if a user-VM happens to perform the role of an intermediate router, then it is required to enable the forwarding functionality in the kernel of that VM. An example code snippet of one such ConfigScript is given below. It copies network script generated for two user -VMs to their respective locations and then restarts their network daemon. At the end of the file are some dynamic firewall rules that are responsible for facilitating remote access to these virtual machines over a secure SSH connection. We talk about iptables rules in the following section in details.
#!/bin/bash

#********* Configuration steps for 10.211.19.186 *********
scp -o StrictHostKeyChecking=no
/etc/virtualLabConfigDir/akadne/10.211.19.186_interfaces
10.211.19.186:/etc/network/interfaces
ssh -o StrictHostKeyChecking=no root@10.211.19.186
/etc/init.d/networking restart

#********* Configuration steps for 10.211.19.187 *********
scp -o StrictHostKeyChecking=no
/etc/virtualLabConfigDir/akadne/10.211.19.187_interfaces
10.211.19.187:/etc/network/interfaces
ssh -o StrictHostKeyChecking=no root@10.211.19.187
/etc/init.d/networking restart

#********* Firewall Rules *********
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 9002 -d 149.169.226.16 -j DNAT --to-destination 10.211.19.186
iptables -A FORWARD -i eth0 -p tcp --dport 9002 -d 10.211.19.186 -j ACCEPT
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 9003 -d 149.169.226.16 -j DNAT --to-destination 10.211.19.187
iptables -A FORWARD -i eth0 -p tcp --dport 9003 -d 10.211.19.187 -j ACCEPT

Figure 7. Code Snippet of configScript
4.5.3 Preparation of iptables rules for the Virtual Machines

This sub section of the virtualLabConfigurationServer is responsible for generating dynamic iptables rules that play vital role in making user virtual machines available over internet in a secure fashion. It basically involves two steps:

1) Contact the port server and get required number of unused/ released port number. Once obtained, these port numbers are mapped to the administrative IP address (explained later) of each Virtual machine in a random fashion. This port-number IP address mapping is then handed over to the subsection of the program that actually prepares the iptables firewall rules, explained in step 2.

2) Once the port-number IP address mapping is ready, it is used to generate the dynamic iptables rules and are appended at the end of the ConfigScript, as visible in the code snippet shown above.

Combining it all, the Wrapper shell script calls the virtualLabConfigurationServer that in turn generates the network files, generates the configuration commands for each of the virtual machine, contacts the virtualLabPortServer to get port numbers, maps them to the administrative IP addresses of each virtual machine and finally generates iptables rules. Thus, when the server thread terminates, it generates number of network files one per each user VM and one common ConfigScript for the user network.
CHAPTER 5
SYSTEM DESIGN

This chapter essentially continues our discussion from the previous chapter. It mainly focuses on the automated configuration steps followed once the user clicks on the submit button on the graphical user interface (userSpace) for the Virtual Lab. The physical setup for the Virtual Lab infrastructure management (refer Figure 3) can be logically modeled into three sub systems viz. Web-server – Database subsystem, Cloud Subsystem and Virtual Lab Administrating subsystem.

Following subsections briefly describe the roles played by these subsystems, before we describe how they interact with each other to fetch the resources requested by the user in efficient manner.

5.1 Web-Server – Database Subsystem
This subsystem is primarily responsible for presenting a user space in the web page rendered to the user browser. Once user submits her
topology, appropriate data values are stored in the database to keep record of user-run networks, to enable their use in future. During a normal request processing, the web-server communicates with the database multiple times at various stages to record relevant information regarding resources fetched from cloud subsystem. Web-server and MS-SQL database in Figure 1 form this subsystem.

5.2 Cloud Subsystem

This subsystem is at the core of the entire Virtual Lab management infrastructure. It seamlessly caters the requirements of multiple users at the same time. It is built on top of XenServer 5.6 open source cloud management software provided by Citrix systems. It provides multiple features of managing resource pool, network configuration, virtual instance execution, interfacing network storage with the cloud etc. Once a cloud (of resources) formed, it transparently interacts with other subsystems in model to provide resources from the cloud. The pool master, cloud network and the NFS from Figure 1 form this subsystem.

5.3 Virtual Lab Administrating Subsystem

It is responsible for connecting created virtual instances so as to implant the user’s network topology, by configuring network scripts, routing rules, host parameters etc. In addition to this it is also responsible for embedding appropriate firewall rules in the gateway of the infrastructure so as to pass selected traffic from the firewall to provide access to the individual nodes of the network. Gateway/Administration
server, Port server and DHCP server in figure 3 are the integral components of this subsystem.

5.4 Putting It All Together

This section describes four major steps followed between the submission of the request from the user and return of the appropriate resources connected in the required fashion, so as to mimic the requested network topology. It can be best explained with the help of Figure 3 and brief description of each step as follows:

Step 1: User contacts the web-server, via the fire walled gateway, on standard port 80, creates intended network graph (one similar to shown in Figure 1) and submits her request. The applet, running at the client-side from the Virtual Lab web-page, estimates this request and forwards it to the web-server. Java applet helps in creating an extremely quick responding user-canvas to create complicated topologies.

Step 2: This step can fundamentally be divided in following sub-steps. In step 2.1, web-server relays the evaluation of user’s request to the cloud management node a.k.a pool master. Pool master is responsible for estimating the availability of the resources in the cloud. Based on its estimation, it identifies physical nodes to run user’s virtual instances. It then reserves these resources against user’s session and assigns unique handles to them. One this is done, pool master returns these handles (step 2.2) to the web-server, which in turn stores it in the database to be used later in the session.
Step 3: This step is responsible for configuring user’s resources and creating administrative entries to provide a secure access to the individual virtual machines in the cloud. It is fundamentally divided into four sub-steps. In step 3.1, web-server acts a client of Administration server, submitting a request to configure the cloud access controls on user’s behalf. This request primarily provides information about the user account, internal IP addresses (meant for administrative purposes only) and static configuration details, if any, provided by user for every virtual node in his network. This step can be further sub-divided into sub-sub steps from 31.1 through 3.1.4 as follows. In step 3.1.1, the administration server contacts a port server, to get pairs of free ports for secure remote access to the individual virtual machines, one each for SSH console and VNC console. The port server creates a pool of available port numbers, which shrinks or grows as multiple users request or relinquish the cloud resources simultaneously. In step 3.1.2, the administrative scripts running on administrative server generate configuration scripts for every individual virtual machine in user’s request. This primarily includes networking configuration, hostname strings, routing table entries etc, as per user specified configuration and port numbers obtained in step 3.1.1 Next step, 3.1.3, uses scripts generated in previous step to actually configure the virtual machines in the cloud, over administrative network. Note that, this network is reserved for administrative and access traffic and it does not allow any other messages to pass, with the help of strict firewall rules. In
step 3.1.4, appropriate firewall ports are opened to allow SSH and VNC traffic to individual virtual machines. Firewall maintains sessions to keep track of mapping between incoming and outgoing traffic. Step 3.2 returns the port numbers obtained in step 3.1.1 to the web-server. User is now almost ready to get access to her network topology in the Virtual Lab infrastructure.

Step 4. This step stores the appropriate information about the session in database and renders the virtual machine (node in user’s network topology) - port number mapping on the response web page. Now user can securely access the network using Public IP and port numbers using SSH client software or VNC client software, as required.
CHAPTER 6
SECURITY ANALYSIS

In this section, we first describe the inherent features of XenServer edition [] that allows us to maintain some basic level of security. In the subsequent sections, we discuss about the extra care taken to protect the system from inside attackers.

6.1 XenServer Security

The XAPI tool-stack itself is written in a high-level, statically type-safe language known as Objective Caml (or OCaml). This guarantees that it is free from low-level memory corruption issues such as buffer overflows or integer overflows, making it much more robust against malicious attacks over the administration network. The SSL layer uses the popular stunnel package to provide industry-standard SSLv3 encryption. VM live relocation involves transferring the memory image of the VM while it is still running. Since a high performance transfer will minimize the performance impact on the running VM, and live relocation is only supported between machines on a local network, this transfer occurs in plain-text over port 80. Bear in mind that if you do configure XenMotion across WAN links that you will need to use IP-level security (e.g. IPsec) to encrypt the memory image.

It is possible to not bind an IP address to the administration network interface, which will mean that none of the administration functions will work from outside of the local console on the XenServer host. Be aware
that in this configuration you will not be able to create resource pools, import/export VMs, or otherwise take advantage of features such as e-mail alerting.

Resource pools are assigned a pool master which controls all the other hosts. All communication between resource pools is done over SSL, and hosts authenticate themselves to each other using a randomly generated symmetric key that is created at the time of pool creation.

A common way to isolate this administration traffic is to use Ethernet VLANs to segregate it from other nonadministration traffic. You can configure your routers to tag all traffic from the administration NICs with an administration VLAN tag. This VLAN can also be used for other appliance control traffic in your server farm, such as Citrix Provisioning Server or Citrix NetScaler. Note that remote clients such as XenCenter will need to connect to the administration network VLAN.

6.2 Virtual Lab Setup security
The system design of the Virtual Lab platform bolsters the security features provided by isolating the entire system from outside world by introducing a powerful firewall in between the outside word and the system entry-point. Although it sounds similar to the conventional approach followed to protect the systems, we make extensive use of iptables rules to develop a so called “White-List” of resources to be allowed to access. In addition to this, this “White-List” is continuously updated as different users
connect to or disconnect from the Virtual Lab platform. In what follows this, we describe about this two-fold security one-by-one.

6.2.1 Protection from intruders, attackers on internal network

This is specifically achieved by isolating the cloud-infrastructure from physical connections to any of the service ports that are connected either to the internet or to the internal routing system of the building in which the cloud-setup is situated. This alleviates the possibility of some positively motivated hacker-cum-students who try their skills on various internal machines just for fun. We want to avoid any such funny or serious incident. Both of our LAN segments viz. Virtual Lab Management Network and Cloud Traffic Network are formed using two physically separate switches, protected in a monitored server cabin. No one can get physical access to these switches except administrators of the system and thus protected from any form of physical intrusions.

6.2.2 Protection from intruders, attackers on the external network

Once the user VMs are ready to be accessed, the user needs to connect to the firewall of the system to access them. This places a restriction of passing the “White-list” maintained at the firewall to be able to access the user VMs. This can be best explained using the code snippet shown in section. If you observe the firewall rules section of the code snippet of ConfigFile here, you can quickly conclude that only ports that are specifically opened in the firewall of the gateway machine could be passed through, which eventually, deliver the traffic through it, to the appropriate user VM, after translating the port address to the
administrative IP address of the corresponding VM. If an intruder from outside tries to randomly connect to any port, it is more likely to be rejected by the strong firewall system. In the event of that port to be assigned to some other valid user, the intruder still need to guess the user credential assigned to the VM, to actually get access to the VM. This can be further improved by maintaining an additional module at the Gateway machine to authenticate users, with the help of Kerberos server, creating mappings between user and port numbers as services. This can be regarded as one of the future tasks for this project. It is very easy to install an intrusion detection system on the Linux-based gateway machine to track attack like port-scanning, and take protective actions.

Complimentary to the addition of dynamic rules to the gateway firewall (to maintain the “White-List”) we also remove the respective port entries from the firewall, once user logs-off from the system. This provides a fool-proof approach to protect the system by dynamically opening-closing only those ports which are being used for valid users.

Additionally, since users are required to access the VMs over an SSH connection, it also benefits from all the security features provided by the communication over secure SSH connection.
CHAPTER 7
PERFORMANCE EVALUATION

7.1 Hardware Platform

An ideal cloud platform should be built on top of a single powerful multi-processor server. Such cloud has infinite number of processors, infinite storage space and infinite memory to play with. The prime advantage of this approach is that present day commercial virtual machine softwares for such unified platforms provide efficient management and utilization of underlying resources. But this is just a fairy-tale as it is practically infeasible to have anything in infinite abundance. Thus, practically we have to form a collection of such resources and combine them together to form a logical platform that exhibits impression of such abundance of resources. We generally term it as Pool Of Resources.

For our virtual lab set up, which is still in its preliminary stages of developments and enhancements, we preferred using number of powerful multi-core servers available in the market. In addition to this, we use a number of single-core dedicated machines that serve their role in the automatic configuration management of the virtual resources. In what follows, we discuss about the hardware platform set up of virtual lab in details:

Our current hardware platform consists of three Dell PowerEdge R410 rack-servers connected together by a switched gigabit network. Every server runs its own copy of Citrix XenServer 5.6. Every rack-server
has 16 2.40 GHz processors, 32 GB of memory, 2 TB of storage space and 6 gigabit network interfaces. The other components of the set-up are as follows:

   One PC-based NFS server: This server provides storage for all virtual machine disk images.

   One PC-based gateway host: This server provides access control for the system through its firewall that is managed dynamically.

   One PC-based configuration server: This server is dedicated for the automation of users’ virtual machines’ configuration. It also hosts the port server daemon, described in chapter 4.

7.2 Sample Experimental Set-ups
Here we discuss few sample experimental setups that students can perform using virtual lab tool.

7.2.1 Simple Network Configuration
In this experiment, students simply get access a number of virtual machine in which they try their hands on simple network configuration stuff like setting IP addresses, netmasks, default gateways, routing table entries etc. This is a pretty simple experimental set up and serves as an introductory session for novices.

7.2.2 Firewall Configuration
This experimental set up needs access to at least three virtual machines, in which one VM serves as a firewall / gateway machine, one behind the firewall and another outside firewall. In this set up, students need to write firewall chains to allow the machine behind firewall to access
the machine outside it, without revealing its identity. This is a fun project and students usually enjoy configuring various protocols such as http, ftp, ssh etc through firewall.

7.2.3 Web Server- SSL, Authentication
This set up requires students to enable SSL service on a web server, generate public/private key pairs, create certificate requests etc. It typically involves access to more than 2-3 virtual machines.

7.2.4 Intrusion Detection
This experiment requires students to install an open-source intrusion detection tool called *snort* on one of the virtual machines. It then requires students to perform a number of attacks on the vm that has this tool and by properly configuring the snort rules, detect them.

7.3 Performance Analysis
We choose the experimental set up *Firewall Configuration* to perform a performance evaluation of our hardware platform. In ASU, a network security class is typically attended by 45 students per semester. As described earlier, the firewall configuration set up requires three virtual machines per student. Thus, in peak hours i.e. hours near deadline of the assignment, every student will try to run his set-up on the cloud and thus we will need to support 45 X 3 = 135 or even more virtual images simultaneously. Every virtual image would typically consist of 128 MB of memory, 8-20 GB of Hard Drive, at least 1 CPU and multiple network interfaces.
We ran 60 virtual images on the cloud and set some traffic among them across virtual interfaces embedded inside them. This traffic is carried over the cloud network shown in figure. The result were very exciting as we can see in figure that, running 60 virtual machines, all on one of the three rack-servers uses up only 7% of its total computing power. In addition to this, it uses around 27% of its total available memory and puts almost negligible strain on the high-speed, gigabit physical network interface card that hosts the virtual LANs for all these virtual machines. From the obtained results, we extrapolate that we can run at least 240 virtual machines of above stated configuration simultaneously on a single rack server in the virtual lab infrastructure. Thus, on three servers we can run $240 \times 3 = 720$ virtual machines simultaneously for a class of 45 students, which allocates around 16 virtual machines for every student. None of the experimental set up in any of the information assurance and network security classes offered at graduate level will need these many virtual machines for a student to perform his experimentation. Thus, this set up is well self sufficient to cater the needs of any information assurance and network security classes offered in the field of Computer and Network Security Education.
CHAPTER 8
RELATED WORK

This section attempts to briefly describe and appreciate the work done by various authors across different universities and research laboratories from the USA. We observed that <names> from Stony Brook University attempted to develop a similar infrastructure using VMWare virtualization technology.

[18] proposes an approach that uses datalink layer virtualization in addition to the virtualization of hosts involved in security experiments. It is implemented by rewriting every network packet created within a security experiment in such a way as to isolate these packets from the underlying network infrastructure. As a result, malicious packets can not escape outside the experimentation network, and therefore have no opportunities to exploit vulnerabilities in servers, firewalls or routers within production environments. Remote access can be provided by creating a communication tunnel from an experimenter’s workstation to the experimentation network.

This approach primarily focuses on the implantation of network virtualization that can be used to provide security in much the same way as host virtualization. In the same spirits a VMM mediates all the accesses made by a guest OS to host hardware, this approach relies on a packet rewriter deployed on the host machine that mediates all accesses to the physical network made by a guest VM. This approach believes that since
VMM is a much simpler piece of software that provides a much narrower interface to guest OS as compared to the complexity of an application-to-OS interface, it is believed to provide an adequate level of security for experiments involving malware. In the same manner, it rewrites every packet that goes out of the guest VM and thus can be relied upon to ensure isolation of the virtual networks from the underlying physical network.

In this, the researchers propose to encapsulate the guest VM’s network packets in a manner that they will no longer be interpreted by the underlying network fabric. In particular, consider a datalink layer packet $p$ from a host $A$ to another host $B$, where $A$ and $B$ are part of a security experiment. In V-NetLab, both $A$ and $B$ will be implemented as guests on host machines $A_h$ and $B_h$ respectively. The packet $p$ is intercepted by a packet rewriter (implemented using a kernel module) on host $A_h$, which generates a new data link layer packet $p'$ with the source address of $A_h$, destination address $B_h$, and a protocol identifier ETH P VNETLAB that is unused in the (physical) test bed. The payload of $p'$ is the entire packet $p$. On $B_h$, the kernel hands packets with the protocol identifier of ETH P VNETLAB to our packet rewriter, which inverts the above transformation and hands $p$ to the guest $B$. Due to the fact that $p'$ looks like any other datalink layer packet from $A_h$ to $B_h$, it is highly unlikely to compromise any components on the physical test bed that operate at these data link layer.

Moreover, since the protocol identifier of ETH P VNETLAB is unknown to
these components, they are unlikely to inspect or process its payload. As a result, components on the physical network are highly unlikely to be compromised (or affected in any way) by network traffic generated as part of security experiments. Additionally, the entire payload of $p'$ can be encrypted in order to ensure that its contents remain confidential, or to ensure that the resulting payload looks essentially random (i.e., uncorrelated with the original packet contents) and hence cannot predictably be used to exploit vulnerabilities on these devices/services.

According to [18], the advantage of virtualization at the data link layer is that it permits the use of any layer-3 protocol within the security experiments, including IP, ICMP, ARP etc. In addition to this, since the packets transmitted by guest OSes remain encapsulated on the physical network, it is possible for different virtual networks to use overlapping IP addresses without interfering with each other.

Planetlab [21] is a distributed laboratory that provides convenient management tools to startup and / or control a large collection of hosts that run identical software. Emulab [22] is another similar approach that provides light-weight virtualization, based on FreeBSD Jails, but this approach does not provide the degree of flexibility needed for our approach, where computers running different OSes may need to be hosted on the same physical machine. An alternative mode supported in Emulab is one where physical nodes on the testbed can be dedicated to run a custom OS image. his approach provides the desired degree of
flexibility to support security course assignments, but does not allow sharing of underlying hardware across multiple OSes. VNET [19] and VIOLIN [20] have some similarity with the approach presented in [18]. VNET approach is based on tunneling Ethernet packets over TCP/IP whereas VIOLIN uses an application-level virtual network architecture built on top of an overlay infrastructure such as a Planetlab. It uses UDP tunneling in the Internet domain to emulate the physical layer in the VIOLIN domain.

Our work tries to advantage of the knowledge base created by the above mentioned research in addition to the cutting-edge technological advancements, to lay out a solid, efficient platform to offer the resources from the cloud. Readers are encouraged to contact the author, should they have more interests in knowing about the implementation and configuration set-up of this platform, outside the context of this document.
CHAPTER 9

FUTURE WORK

Research presented in this document attempts to develop a proof-of-concept of a cloud platform that allows users to remotely access resources fetched from the cloud. Thus, it presents vast scope of improvisation and enhancement to the presented implementation, to make it more robust, efficient, user-friendly and importantly more secure. We identify following tasks that can be taken up by interested researchers to contribute towards the development of this amazing project.

A stronger authentication system can coupled with the existing platform to make it more robust against fake users. The best option for this should be a Kerberos server running in between the User and the platform, authenticating and authorizing users for services. The firewall port numbers uniquely tagged to a particular user at any given time, could be regarded as the services available to that user and thus other users trying to access the services (ports) can be denied the access at the authentication layer itself. In this way, even when a malicious user gets access to the VM credentials that belongs to other user, he will not be able to actually access the VM. Present authentication system does not take care of this type of attack.

Present implementation of the portal does not provide option to save their network topologies for future use. This can be incorporated in the portal by developing appropriate object serialization techniques and
reverse. In addition to this, there exist numbers of enhancement features like ability to edit a pre configured network, replicate an existing network, creation of generic network topology templates etc that can add versatility to this portal.

Presently, we propose this platform to be used by the students enrolled at Arizona State University alone. However, we plan to collaborate with number of other schools across the United States to incorporate data centers formed at respective schools to set up a distributed cloud platform, which provides remote access to students/users across number of schools, on free or pay-as-you-go basis.
REFERENCES


(1) ASU IA Lab
   http://snac.eas.asu.edu/IA-lab/
(2) Syracuse University SEED
   http://www.cis.syr.edu/~wedu/seed/index.html
(3) Stony Brook (SUNYSB) V-Lab
   http://seclab.cs.sunysb.edu/seclab/rprobsdesc__vnet.html
(4) Iowa State (IASTATE) University Xen World
   http://home.eng.iastate.edu/~hawklan/xw-assignments.html
(5) University of Houston-Clear Lake, Houston, TX
(6) Rochester Institute of Technology (RIT) NSSA Labs
   http://www.nssa.rit.edu/?q=node/49
(7) University of Illinois at Urbana-Champaign (UIUC) Security Lab
(8) University of New Mexico (UNM) IA Lab
   http://ia.mgt.unm.edu/labintro.asp,
   http://portal.acm.org/citation.cfm?id=1121341.1121375
(9) Idaho University VRAD Lab
   http://www2.cs.uidaho.edu/~cs481-54/documents/VRAD%20Lab_Operational_Spec_v10.doc
(10) Texas A&M University, College Station
(11) Georgia Tech Information Security Center (GT-IS C)
    http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01593876
(12) National Defense University (NDU)
   http://www.ndu.edu/irmc/ia/labs.html
(13) UMBC
(14) Binghamton University
   http://cait.binghamton.edu/cait/files/publ/Volynkin.TridentCom.v.1.2
   http://portal.acm.org/citation.cfm?id=10–29533.1029539
(15) University of Wisconsin-Eau Claire, Eau Claire, WI
   http://portal.acm.org/citation.cfm?id=1028174.971438
(16) Wofford College
(17) University of Toledo, Elyria, OH
(18) Kennesaw State University, Kennesaw GA
(19) Towson University, Towson, MD
(20) NYU, Polytech, http://isis.poly.edu/
   http://isis.poly.edu/~vlab/vlab_paper.pdf
(21) Purdue University
   http://www.cerias.purdue.edu/site/securitylab
(22) Carnegie Mellon University
   https://www.vte.cert.org/vteweb/default.aspx
(23) Brooklyn Polytechnic University
   http://isis.poly.edu/
(24) Universitaet Trier, Germany
   http://subs.emis.de/LNI/Proceedings/Proceedings56/GI-Proceedings.56-5.pdf
(25) University of Milan, Italy
   http://www.elab.usilu.net/FOSLET06/proceedings/01_damiani.pdf
(26) United States Academy at West Point
   http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1232395


[8] W. contributors, "Web 2.0,
   "http://en.wikipedia.org/w/index.php?title=Web_2.0&oldid=289440096,


[25] XenServer Software Development Kit Guide 5.5.0 Update 2
APPENDIX A

ABOUT XEN SERVER
This appendix briefly describes the features and capabilities of XenServer 5.6, the Cloud layer of the Virual Lab platform shown in Figure 2. We refer to [24],[25] hugely to summarize about the features provided by Citrix XenServer 5.6. Interested readers are directed to [25] for more details. Readers, interested in understanding only the workflow and analysis of this research, can very well skip this Appendix.

A.1 Object Model Overview

![Diagram of API classes for managing VMs, Hosts, Storage, and Networking]

Figure 9. Graphical overview of API classes for managing VMs, Hosts, Storage, and Networking

This section gives a high-level overview of the object model of the API. A more detailed description of the parameters and methods of each class outlined here can be found in the XenServer API Reference document.
We start by giving a brief outline of some of the core classes that make up the API.

A.1.1 VM

A VM object represents a particular virtual machine instance on a XenServer Host or Resource Pool. Example methods include start, suspend, pool_migrate; example parameters include power_state, memory_static_max, and name_label. (In the previous section we saw how the VM class is used to represent both templates and regular VMs).

A.1.2 Host

A host object represents a physical host in a XenServer pool. Example methods include reboot and shutdown. Example parameters include software_version, hostname, and [IP] address.

A.1.3 VDI

A VDI object represents a Virtual Disk Image. Virtual Disk Images can be attached to VMs, in which case a block device appears inside the VM through which the bits encapsulated by the Virtual Disk Image can be read and written. Example methods of the VDI class include "resize" and "clone". Example fields include "virtual_size" and "sharable". (When we called VM.provision on the VM template in our previous example, some VDI objects were automatically created to represent the newly created disks, and attached to the VM object.) SR An SR (Storage Repository) aggregates a collection of VDIs and encapsulates the properties of...
physical storage on which the VDIs' bits reside. Example parameters include type (which determines the storage-specific driver a XenServer installation uses to read/write the SR's VDIs) and physical_utilisation; example methods include scan (which invokes the storage-specific driver to acquire a list of the VDIs contained with the SR and the properties of these VDIs) and create (which initializes a block of physical storage so it is ready to store VDIs).

Network

A network object represents a layer-2 network that exists in the environment in which the XenServer Host instance lives. Since XenServer does not manage networks directly this is a lightweight class that serves merely to model physical and virtual network topology. VM and Host objects that are attached to a particular Network object (by virtue of VIF and PIF instances -- see below) can send network packets to each other. At this point, readers who are finding this enumeration of classes rather terse may wish to skip to the code walk-throughs of the next chapter: there are plenty of useful applications that can be written using only a subset of the classes already described! For those who wish to continue this description of classes in the abstract, read on. On top of the classes listed above, there are 4 more that act as connectors, specifying relationships between VMs and Hosts, and Storage and Networks. The first 2 of these classes that we will consider, VBD and VIF, determine how VMs are attached to virtual disks and network objects respectively:
A.1.4 VBD

A VBD (Virtual Block Device) object represents an attachment between a VM and a VDI. When a VM is booted its VBD objects are queried to determine which disk images (i.e. VDIs) should be attached. Example methods of the VBD class include "plug" (which hot plugs a disk device into a running VM, making the specified VDI accessible therein) and "unplug" (which hot unplugs a disk device from a running guest); example fields include "device" (which determines the device name inside the guest under which the specified VDI will be made accessible).

A.1.5 VIF

A VIF (Virtual network InterFace) object represents an attachment between a VM and a Network object. When a VM is booted its VIF objects are queried to determine which network devices should be created. Example methods XenServer Software Development Kit Guide Overview of the XenServer API 9 of the VIF class include "plug" (which hot plugs a network device into a running VM) and "unplug" (which hot unplugs a network device from a running guest). The second set of "connector classes" that we will consider determine how Hosts are attached to Networks and Storage.

A.1.6 PIF

A PIF (Physical InterFace) object represents an attachment between a Host and a Network object. If a host is connected to a Network (via a PIF) then packets from the specified host can be transmitted/received by the corresponding host. Example fields of the PIF
class include "device" (which specifies the device name to which the PIF corresponds -- e.g. eth0) and "MAC" (which specifies the MAC address of the underlying NIC that a PIF represents). Note that PIFs abstract both physical interfaces and VLANs (the latter distinguished by the existence of a positive integer in the "VLAN" field). PBD A PBD (Physical Block Device) object represents an attachment between a Host and a SR (Storage Repository) object. Fields include "currently-attached" (which specifies whether the chunk of storage represented by the specified SR object) is currently available to the host; and "device_config" (which specifies storage-driver specific parameters that determines how the low-level storage devices are configured on the specified host -- e.g. in the case of an SR rendered on an NFS filer, device_config may specify the host-name of the filer and the path on the filer in which the SR files live.)

A.2 XenServer Networking

It’s important to understand that XenServer networking operates at Layer 2 of the OSI.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Application layer</td>
</tr>
<tr>
<td>6</td>
<td>Presentation layer</td>
</tr>
<tr>
<td>5</td>
<td>Session layer</td>
</tr>
<tr>
<td>4</td>
<td>Transport layer</td>
</tr>
<tr>
<td>3</td>
<td>Network layer</td>
</tr>
<tr>
<td>2</td>
<td>Data link layer</td>
</tr>
<tr>
<td></td>
<td>• LLC sublayer</td>
</tr>
<tr>
<td></td>
<td>• MAC sublayer</td>
</tr>
<tr>
<td>1</td>
<td>Physical layer</td>
</tr>
</tbody>
</table>

Figure 10. OSI Model

This means it’s independent of any L3 addressing, such as IP. As we’ll see, XenServer acts as an L2 virtual switch.
Simple Ethernet Segment

In this simple Ethernet segment, how do nodes A and C talk to each other?

Figure 11. Simple LAN over Ethernet

Ethernet defines a frame (not a packet) which is the carrier of the data payload from the upper layers.

Figure 12. Ethernet Frame Structure

This frame is placed on the wire by layer 1 of node A, and is picked up by node C. If we looked at layer 2 of node C, we would see the same frame that node A transmitted. So:

- Each Network Interface Card or NIC for each node has a unique address, usually burnt in at the factory. This is known as the Media...
Access Control address, or simply the MAC address.

- When preparing a frame for transmission the destination MAC address must be known. Getting the target address to layer 2 is taken care of by layer 3, for example ARP when using IP addressing.

- Each frame has a source and destination address. The frame is seen by all nodes on the segment, but discarded by all but the NIC that has this destination address.

- In our example, node C knows who sent the frame, because of the built-in source MAC address.

Ethernet Segments

Several Ethernet segments can be joined together with a device known as a hub. A hub is also known as a repeater hub, because it repeats any frame it receives on one port out of all its other ports. This in effect turns the connected segments, into one big segment. This has scalability problems due to the way frames are transmitted.

Collisions

Nodes A & B can place a frame on the network at any time, in hopes that the circuit is not busy. If the circuit is busy, perhaps because both nodes are trying to transmit at the same time, then both nodes will back off and retry after a random period of time. As the number of nodes increase, though, the chance of two frames from two different nodes colliding becomes a significant limiting
performance factor for the segment.

Collision Domains

Similar to a hub, a switch, which is sometimes called a switched-hub, connects multiple segments. The difference is that it learns which port, on the switch, a MAC address belongs to by looking at each frame's source MAC address. When it needs to send a frame to that MAC address in the future, it doesn't repeat the frame on all ports, it sends it to the port down which the frame belongs. For the duration of transmitting the frame, this effectively creates a segment between the two ports – the source and destination – so that the two segments can talk at wire-speed. This keeps the collisions confined to each segment.

Addressing
Even though node A addresses node C by its MAC address, this still leaves the problem of how A knows the MAC address of C in the first place.

![Figure 14. MAC address learning on Ethernet segment](image)

Layer 3 of the 7-layer OSI defines a network addressing scheme, most commonly the Internet Protocol (IPv4). Here both nodes A and C have been given unique IP addresses. Although not obvious here, we haven't just replaced one addressing scheme with another. IP can route packets from one node in one Ethernet segment to another node in another segment across the world.

As previously stated: an Ethernet frame has a destination address, and only the NIC with *that* address will accept the frame. There is one of two exceptions to this rule, and it’s called a broadcast frame. A broadcast frame is one whose destination MAC address has all the bits set, and every NIC on the segment will process the frame.
There is a Layer 3 to Layer 2 mapping protocol called the Address Resolution Protocol (ARP) that takes advantage of broadcast frames. ARP is used to translate L3 addresses like an IP address into an L2 address like a MAC address. In the above example network, node A has an IP address of 192.0.0.1, and node C has an address of 192.0.0.3. With TCP/IP, node A would want to communicate with node C by IP address and not MAC address, but Layer 2 still needs to address that payload to node C by MAC address. In this case, node A at Layer 3 uses ARP to map node C’s IP address to a MAC address, so that L3 can tell L2 how to address the frame. But first how does ARP know the MAC address of node C? At first it doesn’t, so ARP sends out an *ARP request* – a broadcast frame – with a payload of the IP address for node C. All nodes in the segment will process the ARP request, but only one will send an *ARP response* if they have that IP address – which in this case should only be node C. The ARP response will contain the MAC address. But this would be inefficient if we had to preempt every real frame a wanted to send to C with a broadcast. For this reason, each node in the segment will keep a map of IP address to MAC address translations that ARP will look at first. This is known as the “ARP cache” or “ARP tables”. In both a Windows and Unix shell, you can interrogate that machine’s ARP cache with the “arp –a” command.

Routing
A router connects networks at layer 3, usually IP. The router – or
gateway – also has an IP address. In fact a router is a device that can
have several IP addresses, and his job is to route packets from one
TCP/IP network to another. Each interface – or port – on the router will
have a MAC address.

Usually, at a minimum, each node on a network has two IP
addresses it knows about: its own address and the address of the router
(usually called the default gateway). Users usually use TCP/IP addresses
that are passed down from L7, for example when browsing web sites.

In this example, if node NA wants to send a TCP/IP packet to node
PB, what needs to happen? With routing, node NA will know if the
destination IP address is contained on his subnet, or not. In this case it's
not, so node NA at L3 knows he needs to send his TCP/IP packet through
the router to get to node PB. In this case, though, the destination MAC
address in the Ethernet frame will actually contain the address of the port
on the router to which network N is connected. Node NA discovers the
MAC address for that router port by using ARP in the usual way – the
ARP request will actually contain the IP address of that router port to which network N is connected.

The router will take that IP addressed packet and place it on the appropriate port for the destination IP address. In this example the port on the router that is attached to network P will now have to use ARP in the usual way to find the MAC address of node PB.

This is a very simple example, and it can get a lot more complicated by adding switches and more routers. These basics are the same in all cases, though.

Bridging

A bridge, in the case of XenServer networking, is the same as a switch, except its implemented in software on the XenServer host. The bridging software XenServer uses is the standard Linux implementation, with no special code from Citrix. There is therefore plenty of documentation available online.

Figure 16. Bridged Ethernet Segments
XenServer Internal Network

This is a simple picture of how XenServer networking works. Each virtual machine has a virtual NIC, which is connected to a virtual network, which is controlled by the XenServer host: Domain zero. The network shown acts just like a physical Ethernet segment, and works at layer 2 of the OSI – there is no TCP/IP configuration needed here. Just like a real network, the way the network is utilized is completely up to the way the virtual machines are configured.

Figure 17. XenServer Internal Network

If they need to communicate using TCP/IP then the individual OSes on those VMs need to be setup, just like with real machines on a network. Configuration doesn't just apply to TCP/IP. The virtual machines will almost certainly need services such as DNS and DHCP. The above picture depicts, what is known in XenServer, as an Internal Network: It has no connection to the outside world.

The only difference between a XenServer Internal Network, and a XenServer External Network is that we can connect the virtual Ethernet to
a real NIC. This means that the virtual machines can now take advantage of real services provided by other servers outside of the XenServer host. In the Unix world, the NICs, real or virtual, are called interfaces. XenServer calls a real interface a Physical Interface or PIF and each virtual interface in a virtual machine is a VIF.

In the OS running on a VM, the VIF looks and operates like a locally installed PIF. In Windows, the device driver name may be different, depending on if the paravirtualized tools have been installed or not. In reality, XenServer networking is accomplished by connecting the VIFs and optionally 1-PIF to a virtual switch or bridge. If you remember, a switch learns the MAC addresses of the nodes connected to each of its ports, thereby reducing the amount of traffic on its other ports. It’s the same with a bridge.
In the real world a switch is used to reduce the number of collision
domains on attached segments. In XenServer networking, a bridge is
used to connect virtual machines together not networks. It’s also used to
connect those virtual machines to the outside world.

You can create several internal networks, but (in the following
network) the only way to route between network Q and network R, is to
have a machine that has 2 VIFs, one in each network. Each VIF can
belong to only one network, but a virtual machine can have many VIFs. A
virtual machine on internal network Q can gain access to the external
network R, by routing through one of the virtual machines that has one VIF
on each network. Internal networks N and P will not be able to
communicate with each other, or the PIF.

So how does this transpire in reality?

Each virtual machine has a unique ID, and in the above example
the Windows VM has an ID of 1, and the Linux VM has an ID of 2.
The virtual machines, assuming that the PV Tools have been installed, will see an Ethernet NIC in terms of their own environment. This means that the Windows VM will see the “Citrix XenServer PV Ethernet Adapter” in the device manager and the Linux VM will see “eth0”.

On the host-side, what is actually the other side of the virtual Ethernet cable joining these VMs to the bridge; the Ethernet interfaces that XenServer sees will have a naming convention. Each interface on VM1 – or guest1 – will begin with V-I-F-1-dot, followed by a sequence number.

![XenServer Host (Dom0)](image)

Figure 20. XenServer Internal and External networks together

that represents the interface number on that VM. So in this case, the first adapter on VM1 will have an interface name of VIF1.0 inside the XenServer host. Whatever frames are transmitted by VM1 on his virtual NIC, will be received by VIF1.0. Any frames transmitted by VIF1.0 will be received by the VM1 virtual NIC. The real NIC on the XenServer host will be “eth0” and will be plugged into the bridge as well. Virtually speaking, this would not be from his RJ-45 interface, but from his bus interface. The
RJ-45 interface is a real interface used to connect the XenServer host to the outside world. In this setup, the bridge is also seen as an interface. If it's joining members of an external network together, its name will begin with “xenbr”, the “br” meaning bridge. If it's joining members of an internal network together, its name will begin with “xapi”.

XenServer Networking: VLANs

In the case of creating a VLAN, every distinct VLAN will get its own bridge. Also, the (pseudo) PIF will have a dot separated name to include
the vlan tag number and, when on the real network, the bridge name will start with “xapi”. Apart from that, everything else will be the same as normal external network. It’s not possible to create an internal VLAN network.

In this example there are 8 virtual machines as guests on a single XenServer host. Also on this host are 3 VLANs, with tags 5, 65 and 4000. There are also a couple of machines that have a regular non-vlan (untagged) external network, so their bridge is wired directly to the PIF. The interfaces (eth0.5, eth0.65, eth0.400, etc) for the VLANs are actually created in Linux as virtual interfaces. These interfaces simply tag each frame with the appropriate VLAN number before they are moved on to the PIF.

All frames emanating from a guest VLAN will leave the PIF, and hence will be seen by the outside world, as tagged. This means when

Figure 23. VLAN Trunking

created in Linux as virtual interfaces. These interfaces simply tag each frame with the appropriate VLAN number before they are moved on to the PIF.
having multiple VLAN guest networks, a frame may be tagged differently depending on the source VLAN. For this reason, you are required to (eventually) connect the PIF of the XenServer to a VLAN trunk port that supports 802.1Q encapsulation, when VLANs are in-play. When using VLANs the XenServer Host handles all interpretation of the VLAN tags. Any frames sent to guests that are part of a VLAN will, even so, remain untagged.

XenMotion

During a migration of a VM from one host to another, any memory pages that change on the source host are copied to the destination host. This process is repeated until the number of pages to copy is minimal, and the VM can be started on the destination host.

But in this example setup, the external real switch device is expecting the MAC address of the VM to be on one port, while it’s actually just migrated to another port. In this case, the last step of the migration is for the
destination host to update any external devices with a gratuitous ARP packet. A “garp”, is nothing more than a regular ARP request, but with the MAC and IP address already filled-in. This serves as an update to any external devices' arp cache, and there would be no response.

NIC “Teaming” or “Bonding”

NIC bonds can improve XenServer Host resiliency by using two physical NICs as if they were one. If one NIC within the bond fails the host's network traffic will automatically be routed over the second NIC. In XenServer 4.1, NIC bonds work in an active/passive mode, with only one physical NIC ever in use.

![NIC Bonds Diagram](image)

Figure 25. NIC Bonds

This virtual interface (in the above case Bond0) will then look like a regular PIF to any XenServer External Network.

- XenServer NIC bonds completely subsume the underlying physical devices (PIFs). In order to activate a bond the
underlying PIFs must not be in use, either as the management interface for the host or by running VMs with VIFs attached to the networks associated with the PIFs.

- XenServer NIC bonds are represented by additional PIFs. The bond PIF can then be connected to a XenServer network to allow VM traffic and host management functions to occur over the bonded NIC.
- The bond interface itself can be made to have its own MAC address; otherwise it inherits the MAC address of the first listed PIF when created.
B.1 Functions in Summary.aspx.cs

```csharp
public void prepareSQL(out String sqlVertex, out String sqlInterfaces, out String sqlNetworks)
```

This method is mainly responsible for converting the user topology created on the canvas of the applet into appropriate database queries. It does so by parsing out the string input fetched from the user session variable "graph", which contains the string form of the user topology. The function parses this string and segregates its contents into three distinct types of SQL queries viz. sqlVertex, sqlInterfaces and sqlNetworks. These queries contain configuration information about the nodes, interfaces and the networks created by the user on the canvas of applet.

```csharp
protected void acceptButton_Click(object sender, EventArgs e)
```

This is the method called on the click of Accept button of Summary.aspx page. It is mainly responsible for calling a number of functions to actually lay out the user defined topology on cloud platform. It distinctly performs following functions:

1. Call `prepareSQL()` to parse the user graph and prepare SQL queries.
2. Call `XenConnection()` to securely connect to the cloud infrastructure and reserve a session on behalf of the user.
3. Call `dbConnection()` to securely connect to the MS-SQL database and reserve a session on behalf of the user.
4. Execute queries prepared in Step 1 on the database to store the user topology information in the database against its unique session id obtained in Step 2. This allows to make use of this information later in the session.

5. If Step 4 succeeds, it then contacts the cloud infrastructure to actually initiate the process of reserving the resources on behalf of the user, against the unique session id obtained in Step 2.

6. Once the resources are successfully reserved in the cloud infrastructure, it then configures the VMs (for user defined static IP addresses, Hostname strings, routing rules etc) by calling connectSSH().

7. If step 6 succeeds, it then simply redirects the control to the next page, which displays VM-Ports mapping.

   protected Session xenConnection()

   This is a simple method which creates a dedicated connection to the cloud infrastructure on behalf of the user and returns its handle to the called program.

   protected SqlConnection dbConnection()

   This is another simple method which connects to the MS-SQL database on behalf of the user and returns its handle to the called program.

   protected bool addNetworksInfo(SqlConnection conn, String sqlNetworks, String userSession)

   This method executes an SQL-query that enters the information about networks/VLANs to be created on behalf of user. It receives this query as
an input parameter, along with the XenServer connection and database connection obtained in Step 1,2,3 respectively of method acceptButton_Click().

```csharp
protected bool addVertexInfo(SqlConnection conn, String sqlVertex, String userSession)
```

This method executes an SQL-query that enters the information about the VMs to be created on behalf of the user. It receives this query as an input parameter, along with the XenServer connection and database connection obtained in Step 1,2,3 respectively of method acceptButton_Click().

```csharp
protected bool addInterfacesInfo(SqlConnection conn, String sqlInterfaces, String userSession)
```

This method executes an SQL-query that enters the information about the configurations of interfaces attached to various VMs of the user topology. It includes details like static IP addresses, LANs to which these interfaces etc.

It receives this query as an input parameter, along with the XenServer connection and database connection obtained in Step 1,2,3 respectively of method acceptButton_Click().

```csharp
protected bool createNetworks(SqlConnection conn,out String updateQuery, XenAPI.Session userSession)
```

This method actually interacts with the cloud infrastructure using XenServer connection obtained in Step 2 of acceptButton_Click(), to
reserve the resources like XenServer network handles. Once it does it successfully, it returns true else false.

```csharp
protected bool attachVLANs(SqlConnection conn, XenAPI.Session userSession)
```

This method calls a stored procedure in the MS-Sql database to attach already created, free, unused VLANs to every network handles created for the user topology. This activity is done at the database itself and the opaque references related to VLANs are updated in the networksInfo table.

```csharp
protected bool createVertices(SqlConnection conn, out String updateQuery, XenAPI.Session userSession)
```

This method interacts with the cloud infrastructure and to reserve VMs on behalf of the user. While doing this, it takes care of all the parameters set by the user, like operating system, RAM, HDD etc about a VM. After successful creation of a VM, it saves its handle in the database against user's session to be used in the future.

```csharp
protected bool configureInterfaces(SqlConnection conn, out String updateQuery, XenAPI.Session userSession)
```

This method interacts with the cloud infrastructure and reserves VIFs on behalf of the user. Every VIF has to be attached to a corresponding VLAN on one side and appropriate VM on the other side. By this time, these entries have already been updated in the interfacesInfo table. Thus, it becomes very easy to use these handles and create a VIF using
XenServer API call. After successful creation of these VIFs, this method updates their handles in the database and returns true else false.

```csharp
protected bool updateIP(SqlConnection conn, out String updateIPQuery, XenAPI.Session userSession)
```

The prime responsibility of this method is just to query the XenServer internal database and get the IP addresses of the VMs set to run from `createVertices()`.

These IP addresses then become administrative IPs for these VMs, used for configuration of VMs.

```csharp
protected bool connectSSH(SqlConnection conn, out String updatePortsQuery, XenAPI.Session userSession)
```

This method is responsible for initiation of configuring the VMs created by now and set running. By configuration I mean, it is responsible for setting static IP addresses to all the VIFs of any given VM, their hostname strings, routing rules of the routers in the network topology excite does this by executing a shell script, which receives the username and interfaces’ details string as input parameters. At the end of the execution of this shell script, all the VMs are configured appropriately and VM-to-port mappings are returned. Next part of this method then parses this mapping string and updates the ports number against corresponding VM handle in the database.

**B.2 Functions in virtualLabPortServer.java**
This piece of code creates a daemon caller virtualLabPortServer that listens on port 8888 for requests to fetch ports. On reception of such requests, it creates a thread and dedicates that thread to the incoming request and continues to listen on the port 888 for more requests

B.3 Functions in virtualLabPortServerThread.java
This piece of code is mainly responsible for analyzing the user’s requests (whether it is to get ports or free ports) and then access the synchronized data structure that contains pool of unused, freed ports. In response, it returns these numbers to the user in the form of a concatenated string of the form \textit{port}_n,\textit{port}_n+1,\textit{port}_n+2.

B.4 Functions in virtualLabPortClient.java
This is a wrapper class that does the job of creating a TCP connection to the virtualLabPortServer and requesting for the resources.

\textit{public String exec(String action, String param)}

This method receives the type of request and parameter in the form an integer number corresponding to the type. If it is a get request the param is equal to the number of port numbers to be requested and if it a free request, the param contains a string of port numbers to be returned to the pool.

B.5 Functions in getConfig.java
This piece of code does the job of generating configuration script for user network topology. Basically such script contains ip address configuration commands, commands that add routing table entries, hostname setting
commands etc. In addition to this, it contacts the virtualLabPortServer by creating an instance of virtualLabPortClient and obtains required number of port numbers. It then maps these port numbers to the user VMs on random basis and returns this mapping to the caller program.