DESIGN OF A VIRTUAL COMPUTER LAB ENVIRONMENT FOR HANDS-ON INFORMATION SECURITY EXERCISES[*]

Nathaniel Gephart, Benjamin A. Kuperman
Computer Science Department
Oberlin College
Oberlin, OH 44074
440-775-8556
ngephart@cs.oberlin.edu, benjamin.kuperman@oberlin.edu

ABSTRACT

There is an increasing demand from both students and industry for more computer science students to have experience in information security. One area that is difficult for smaller colleges to implement is hands-on security exercises for students. This paper describes a method and our design considerations for constructing a virtual computer lab environment to be used for such courses. This system uses a single server to host virtual machines that can be configured into small networks and made available for student assignments. These machines for each student group can communicate amongst themselves but are isolated from each other and from the rest of the campus network. Students can be given administrative access to the machines without compromising the security of the campus network.

While our initial focus is on an information security course, this type of system would also be useful for laboratory exercises in operating systems, networking, and system administration classes. This software is under active development and is being made freely available for use in other courses and at other institutions.

INTRODUCTION AND OVERVIEW

Over the past several years there has been an increasing demand from both students and industry sponsors for more graduates with experience in information security. According to the U.S. Bureau of Labor and Statistics, growth in the employment of

[*] Copyright © 2010 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

computer scientists and computer network, systems, and database administrators will be driven in part by the increasing need for information security [2, 3].

In academia, many of the professors with the relevant background experience (either graduate research or other specialization) have positions at larger research oriented institutions. Few smaller computer science programs offer specializations in information security, but a growing number of schools offer it as an elective. In support of these non-specialists, there is a growing wealth of information available for interested professors in the forms of textbooks, prepared course material, research literature, professional training books, and web resources all dedicated to information security. One area that is difficult for a smaller institution to implement is meaningful hands-on laboratory exercises to accompany a security course. For many schools, constructing a dedicated security lab is not feasible due to limited resources including money, space, machines, time, and staffing. This paper describes a method and the design considerations made to implement such laboratory exercises through the use of virtualization and virtual machine techniques, with a particular emphasis on being implemented at 4-year institutions.

We believe that there is a need for realistic, hands-on laboratory exercises in information security courses. Many instructors believe that experiential learning significantly improves understanding and retention of material by students. For many security laboratory assignments, students will need to have privileged access to machines (root or administrator access), a situation that is not easily provided for by a general purpose CS lab environment. Some exercises involve analyzing or controlling the communication between multiple machines increasing the number of machines needed. Further, each student group will be attempting to accomplish the same set of tasks (changes, patching, configuration, etc.) requiring them to each have their own set of machines to work on. Finally, a number of interesting security assignments involve the use of network tools or generation of traffic (worm spreading, port scanning, etc.) which should be isolated from the campus network and care must be taken to keep one group's traffic from interfering with others. In the past, this has been done through the construction of dedicated, closed network laboratories [4] or through mobile carts with laptops [6, 9].

DESIGN CONSIDERATIONS AND DECISIONS

We made a number of important design decisions during the development of this Virtual Computer Lab (VCL). These decisions can be broken down into the following broad categories: physical environment, virtualization platform, security/isolation, and system management. First, the physical environment describes the limitations we encountered and the equipment purchased. The platform selection concerns the style of virtualization hypervisor as well as licensing. Security is focused on the problems of how to manage isolation of worlds, users, and access controls. Finally, system management considers how to automate certain world-related tasks: the management of virtual world state, handing in of worlds for grading, and templates for rapid deployment of labs.

Physical Environment

Before discussing the software environment and our configuration of it, we first want to outline the computing environment at our institution as it motivated much of the design of this project. As is probably the case at many other liberal arts colleges, we have a dedicated computer lab that is available for use by all CS classes. It was viewed as unacceptable, even detrimental, to task a number of these machines for use solely as part of a security course as they are in high demand during our introductory lab sessions. A second possibility would be to purchase and deploy a set of computers dedicated for use in a security course. Space is at a premium – there is not room in the existing lab nor is there vacant space available for a separate lab. It would also have been expensive to purchase a set of machines to be used for a class that is only offered every other year.

Our idea to overcome these limitations was to purchase a single server that could be housed in the campus server room and have it host a set of virtual machines that could be used for the lab. We then could take advantage of the bandwidth and high-speed connections available on our LAN to allow the existing lab machines to remotely access the virtual computers using remote desktop software.

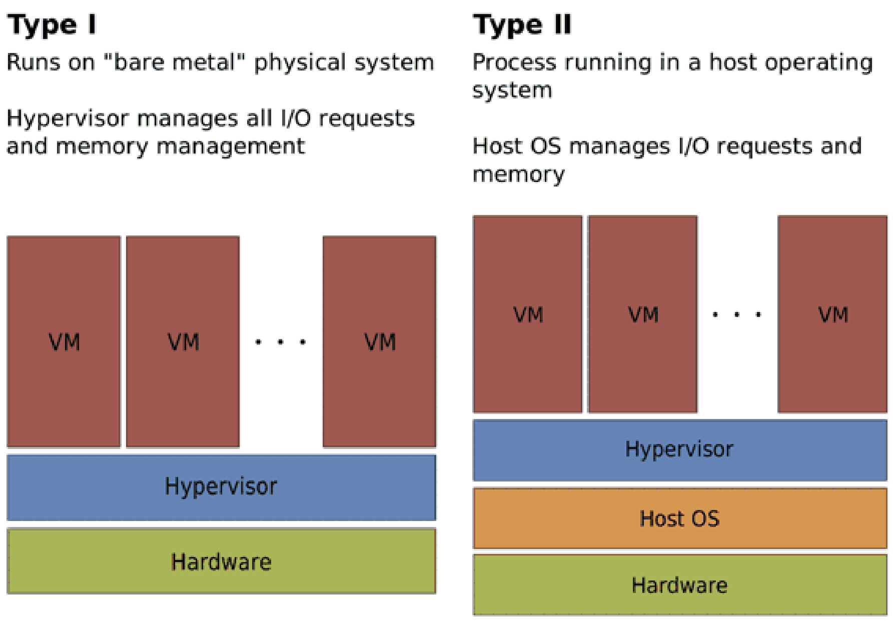
Type I
Runs on "bare metal" physical system
Hypervisor manages all I/O requests and memory management
VM
VM
. . .
VM
Hypervisor
Hardware
Type II
Process running in a host operating system
Host OS manages I/O requests and memory
VM
VM
. . .
VM
Hypervisor
Host OS
Hardware

Figure 1: The two basic types of virtualization hypervisors

In terms of hardware selection, almost all modern x86 CPUs have hardware support for virtualization. A typical security exercise might involve 1 to 3 machines with very light loads, so the processor speed need not be excessive. Space is a greater concern (both memory and disk), as each virtual machine requires a full disk image of the computer and dedicated memory while executing. For our budget of $3,000 we purchased a HP ProLiant ML150 G6 Non-Hot Plug server with a 2.40 GHz Quad-Core Intel Xenon Processor, 24GB of PC3 RAM, and four 250GB disk drives configured as a single 1TB logical volume. For the price of 2 or 3 stand-alone lab machines, we are able to purchase server-class hardware with sufficient resources for virtualization. In our opinion, the specific components are less important than ensuring that there is copious amounts of

RAM available and that some considerations regarding disk space and disk access time are made.

Virtualization Platform

Our first consideration was what virtualization platform to use. We examined a number of available solutions, both commercial and open source. The Xen project [10] and Sun Microsystem's VirtualBox [7] were the most promising open source solutions, while Citrix XenServer [11] and VMware ESXi [8] were the most attractive commercial packages.

Hypervisors exist in two flavors: Type II or host-based hypervisors, and Type I or "bare metal" hypervisors. The latter are made up of a thin software layer with the sole purpose of virtualizing the underlying hardware. The former require the presence of a full-fledged host operating system to manage I/O operations and provide other basic system services. Bare metal hypervisors boast greater speed and security because of the reduced complexity and size of the virtualization code. However, they can also be somewhat limited in their support for a variety of hardware platforms. See [5] for additional details on virtualization.

Licensing was also an important concern as it encompassed questions such as whether we needed professional, paid support of the platform, access to official support documentation, or more advanced feature sets. Since we were developing an environment around one of these hypervisor platforms, having access to support and documentation was essential. The commercial solutions we considered offered basic documentation for their products on their websites, but for troubleshooting and technical support, it was necessary to purchase a support contract. Introducing monthly or yearly expenses for technical support would immediately make our environment significantly less sustainable and decrease its attractiveness as a potential solution for other schools. In contrast, the websites for both the Xen project, and VirtualBox offered access to extensive documentation. Both also had active message boards, IRC chat rooms, and mailing lists where we could ask questions as they arose. In addition to the cost of professional support, the commercial solutions both offered entry level products for free, but required additional licensing to take advantage of larger hardware platforms (more CPUs, RAM, etc.) and advanced features. This was, again, a deterrent because it required recurring cost for us and others to implement our environment.

Ultimately, we chose the open source Xen project to serve as our virtualization platform. We found that it offered the best solutions to the above platforms of the four we considered. Xen is also the basis for the XenWorlds project at the University of Iowa [1], with whom we were collaborating. Because Xen is a very complex system with many close ties to the Linux kernel, we also wanted to find a distribution of Linux that included packages for easy installation of Xen and its dependencies. The number of such distributions is growing steadily; however, we chose to use Debian as the base for our system. Debian's package management tools are well-tested and easy to use, and Debian was one of the first to include Xen packages in its repositories.

Security

The security and isolation of the system was a high priority. We designed this system keeping in mind that these worlds may be used by students to work with potentially harmful software and attack techniques. We therefore required that worlds be given extremely limited or no access to the outside network. Further, we wanted each student group to be isolated from each other so that the network traffic they each generate would not interfere with other groups.

Worlds were isolated by the creation of virtual bridges, one per world, in the Dom0 of the Xen platform. Virtual machines that were together on the same network bridge could communicate with each other, but no other systems. This configuration of bridges in the Dom0 allows easy administration, particularly in that it is easy to move a VM to another world by simply deleting its interface from one bridge and adding it to another. Also, network traffic of all the VMs is handled entirely within the Xen platform; no traffic ever goes out on a physical interface. Problems that arise because of the isolation requirement included how to allow students and administrators to upload files to worlds, and how to handle automatic configuration of network settings.

Another significant set of problems was that of user management and access control. Our isolation requirement dictated that students be limited to accessing only those worlds specifically designated to them. Xen is designed such that the administrator can control and connect to all virtual machines running on the system. Our system has an initial implementation of procedure-oriented access control that mirrors the Unix user management system. This is done through a menu system that only offers a limited subset of actions to the students, and no direct command line access to the Xen server. Each

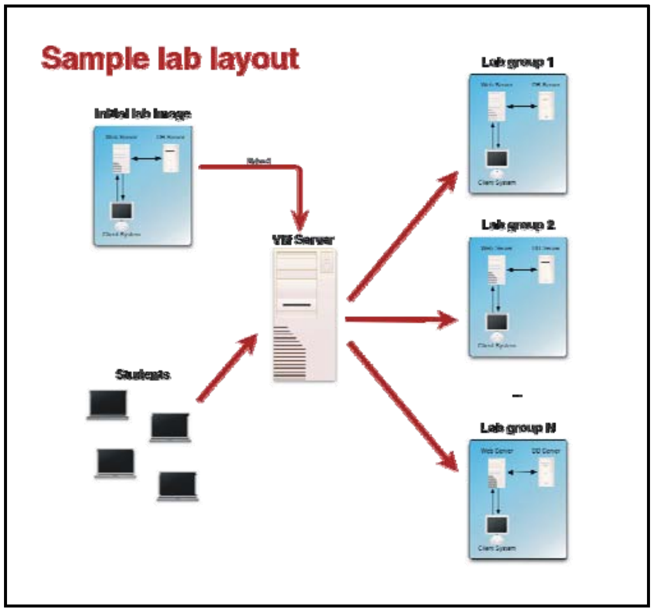

Figure 2: An example of how labs can be deployed using this system. An instructor creates an initial image and the scripts create a copy of this virtual network for each group in the class. Students access the virtual machines

student is given a username in the system along with a unique numeric identifier. The worlds are also identified by name and number. Part of the configuration of the system is a mapping of user IDs to world IDs. A student, thus, should be able to access the worlds allocated to him but no others. When a student logs in the system presents him with the option to log in to only those worlds allocated to him.

System Administration

In a system of this complexity, in which groups of virtual machines are created and destroyed on a regular basis, administration becomes an issue. Questions considered included the following: automated management of virtual machine creation and destruction, how to hand in lab assignments, and automatic provisioning of virtual machines through templates.

Administration of the entire system is through a set of scripts written in Python. Included in the setup package for Iowa State's XenWorlds environment were a number of Python scripts to handle the administration and configuration of XenWorlds worlds. We were able to use much of this code as a base, adapt it to our purposes, and extend it in ways we found necessary.

In order to ease distribution of lab assignments to students, we decided the system had to be capable of creating several worlds at once based on a template or configuration file. The XenWorlds system uses the configuration file format and classes that are part of the Python standard library. In these files, administrators specify the number of nodes in a world as well as the specific disk and network configurations for each node. By pointing the configuration scripts at the instructor created configuration file, the system could create the necessary virtual machines, link them together, and start them on the server. At this point, our system is not yet capable of that level of automation – there are still many tasks that need to be handled by the administrator manually – however, our vision is that running one command would allow the creation and configuration of a world and would not require any user intervention.

In order to handle this kind of configuration automatically, we needed a scheme for assigning IP and MAC addresses to each virtual machine in each world. Since worlds are isolated from each other, it is sufficient to assign the nodes of all the worlds the same MAC and IP addresses. However, this would present significant problems for the administrator if ever two worlds needed to be linked together or VMs needed to be moved from one world to another. Thus, we decided on a scheme that assigned addresses based on the world owner's ID, the world assignment number, and the node number. These three values were sufficient to give all virtual machines running on the server at a given time unique addresses and avoid any potential configuration problems that might occur when moving VMs. A problem we encountered with automatic configuration arose when we tried to setup a Microsoft Windows VM. We were unable to find a way to assign MAC or IP addresses to the VM without manually logging into each machine and specifying the settings. This may be a significant barrier if we decided to use Windows for one of our lab exercises.

Another of our design goals was the ability to manage virtual machine and world state. This goal ties in closely with a number of our other objectives including handing in assignments and templates. It was our hope to include functionality that would allow a student to restore her world to its original state, wiping out any progress she has made since the lab was distributed. This functionality is really a specific case of being able to rollback the state of the world to any predefined save point. There exist several projects that implement this functionality including UnionFS and various copy-on-write solutions. Xen itself can work with COW snapshots of filesystems created in lvm logical volumes. Copy-on-write also promises huge improvements in disk usage because only the changes made to the filesystem need to be stored. Using COW systems, a particular lab setup would need one or more base images. Using those, each virtual world's nodes would be provided with a snapshot as its root filesystem which would keep only the changes the student made to the filesystem instead of a complete copy of the base image.

We also considered handing in of lab assignments. Since our system still uses full copies of disk images per student per lab, handing in entire virtual machines with associated disk images is only feasible over a fast network connection. With COW snapshots, such a hand in scheme is much more viable. We also discussed alternative, more traditional methods of handing in assignments such as lab write-ups. At present, our system does not have any hand in functionality built into the system. This is an area that we will continue to explore.

Finally, to lessen the effort required to create and distribute a lab, we wanted to develop a number of prebuilt disk images. These images would allow rapid deployment of VMs with specific functionality, such as a web server, database server, and attack platform (with various scanning and offensive tools).

CONCLUSIONS AND FUTURE WORK

In this paper we have presented our design for a virtual computer lab to be used for information security exercises at a liberal arts college. To explain our design, and to help others who are designing similar projects, we included a list of items that we had to consider during the construction of this system. By using virtualization, we can reduce the expense in both dollars and space for establishing a hands-on system laboratory.

While our presentation of this system has focused only on information security labs, the lab environment lends itself to use in other courses. Operating systems courses could have students compile and run their own kernels implementing features such as schedulers or virtual memory. Networking courses can have students implement sniffers, route packets, or create and modify network topologies without interfering with the rest of the campus. And information systems courses can have students install and administer a set of servers. With a common file format such as the Open Virtualization Format, it may be possible to create a repository of a wide variety of laboratory exercises to be shared among the academic community.

We are continuing to refine and add additional features to the scripts that make this lab- oratory setup possible. We are concurrently developing a set of security laboratory exercises that we will distribute with the system and use in our own information security class. In the summer of 2010, we will be hosting a regional workshop to give faculty at other liberal arts colleges an opportunity to learn how to install and administer this system at their own colleges.

AVAILABILITY

Additional documentation, source code, and laboratory exercises for this system are available online at http://www.cs.oberlin.edu/~kuperman/research/vmlab/.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant number DUE-0830877 and Oberlin College. We would like to thank the reviewers for their time and comments, the Oberlin College Computer Science Department, and OCCS system administrator Nate Daniels.

REFERENCES

[1] Anderson, B., Joines, A., Daniels, T., Xen Worlds: leveraging virtualization in distance education, *ITiCSE '09: Proceedings of the 14th annual ACM SIGCSE conference on innovation and technology in computer science education*, 293-297, 2009.

[2] Bureau of Labor and Statistics, Occupational Outlook Handbook, 2010-11 Edition, chapter Computer Scientists, 2010, http: //www.bls.gov/oco/ocos304.htm, [cited 22 March 2010].

[3] Bureau of Labor and Statistics. Occupational Outlook Handbook, 2010-11 Edition, chapter Computer Network, Systems, and Database Administrators, 2010, http://www.bls.gov/oco/ocos305.htm, [cited 22 March 2010].

[4] Hill, J., Jr., Carver, C., Humphries, J., Pooch, U., Using an isolated network laboratory to teach advanced networks and security, *SIGCSE '01: Proceedings of the thirty-second SIGCSE technical symposium on computer science education*, 36–40, 2001.

[5] Matthews, J., Dow, E., Deshane, T., Hu, W., Bongio, J., Wilbur, P., Johnson, B., *Running Xen: a Hands-On Guide to the Art of Virtualization*, Upper Saddle River, NJ: Prentice Hall, 2008.

[6] Robila, S., Wachsmuth, B., Scharff, C., Popyack, J., Mobile instructional laboratory environments and their use in computing sciences, *Journal of Computing Sciences in Colleges, 23*, (3), 114–118, 2008.

[7] VirtualBox, http://www.virtualbox.org/, [cited 22 March 2010].

[8] VMware ESXi with VMware Go, http://www.vmware.com/ products/esxi/, [cited 22 March 2010].

[9] Wagner, P., Phillips, A., A portable computer security workshop, *Journal on Educational Resources in Computing, 6*, (4), article 3, 2006.

[10] Xen Hypervisor, http://www.xen.org/, [cited 22 March 2010].

[11] Citrix XenServer, http://www.xensource.com/, [cited 22 March 2010].