Infrastructure Tools for Efficient Cybersecurity Exercises

Jim Marquardson
jimarqua@nmu.edu
College of Business
Northern Michigan University
Marquette, MI 49855, USA

Abstract

Academics are responding to the call from industry for graduates armed with cybersecurity skills. A common challenge that educators face is creating effective cybersecurity curriculum that prepares students with practical skills upon graduation. While hands-on exercises are a powerful method for teaching and assessing cybersecurity skills, these exercises can be difficult to create, require large infrastructure investment, or waste valuable learning time simply configuring the learning environment. In recent years, industry has demanded increased infrastructure automation. These tools have matured and help make provisioning and configuring hardware and software easier. Infrastructure automation tools can help cybersecurity educators create exercises that can scale without adding substantial burden on the educators.

Keywords: Cybersecurity, Infrastructure automation, Curriculum design and development, Computer Security

1. INTRODUCTION

Malicious actors know how to find and exploit weaknesses in systems. They not only know the definitions of key cybersecurity concepts, but they know how to operationalize those concepts. To effectively defend against malicious actors, it is critical that cybersecurity students can apply the skills they learn in the classroom. The National Security Agency / Department of Homeland Security Centers of Academic Excellence in Cyber Defense program defines knowledge units with skills that cybersecurity students should obtain in their degree programs (NSA / DHS, 2013). It is important to note that the knowledge units often list outcomes where students should be able to use, apply, operate, configure, install, and analyze key cybersecurity technologies. These active verbs reinforce the need for cybersecurity students to have practical, hands-on skills.

There are many challenges when creating technical exercises to teach and assess cybersecurity skills. Clearly, some infrastructure is required. Students must have computers with administrative privileges so they can install software and make configuration changes. Often, dedicated cybersecurity labs with networking equipment and servers are needed. Lab networks may need to be segmented from the general campus network. Capital and operational costs musts be considered when designing the infrastructure to support a cybersecurity program. While private labs provide an excellent space for conducting cybersecurity exercises, maintaining infrastructure and configuring systems can be a major burden. For example, giving students an exercise where they configure firewalls on a server might require one new server be created for each student in the class. If done manually, setting up a single lab exercise can be extremely taxing on educators.

Information technology has long aimed to increase operational efficiency in organizations. The increasing adoption of cloud computing has spurred developments in infrastructure automation. Cloud computing is sold with the promise of quick and easy provisioning and deprovisioning of computing resources to scale with demand. To reduce the cost of provisioning resources, cloud providers have leveraged existing tools and built tools to automate infrastructure. Educational institutions can leverage these same infrastructure automation tools within their private infrastructures.

To date, few educators have published clear, actionable strategies for creating cybersecurity exercises that meet student needs without excessive burden on the educator to maintain infrastructure. The following sections in this paper describe some of the most popular tools for virtualization and infrastructure automation that educators can use to develop cybersecurity exercises. Practical recommendations are given to help educators known when adoption of the tool can be beneficial. The purpose of these overviews is to introduce the tool, the benefits the tools provide, and a high-level overview of key concepts.

2. VIRTUALIZATION

Virtualization technologies allow one or more guest virtual machines to run on a physical host. Virtualization can be broken down into two main categories: client-side virtualization and server-side virtualization. The advantages and potential drawbacks of adopting each solution in an educational environment will be discussed in the following sections.

Client-side Virtualization

With client-side virtualization, students run one or more guest operating systems on their computer in a similar way to how they run applications. Changes made inside the virtual machine do not affect the host operating system. A student running Microsoft Windows as her primary desktop operating system could run a Windows Server virtual machine and a Linux virtual machine to observe the interaction of those two systems. The top virtualization platforms for desktop systems are VMWare Workstation Player (*VMWare Workstation Player*, 2017) and Oracle's VirtualBox (*VirtualBox*, 2017).

VMWare Workstation Player is proprietary software. It should be noted that VMWare has changed its licensing model and product offering several times over the past several years. Currently, VMWare Workstation must be licensed for use in the classroom. It is impossible to know if VMWare will continue to support the product or change its licensing structure. VMWare Player has proven to work well for desktop virtualization, but building on top of a closed platform can be problematic.

VirtualBox is an open source desktop virtualization platform maintained by Oracle. VirtualBox is free and supports a wide variety of guest operating systems such as Windows desktop editions, Windows server editions, Linux, BSD, and Solaris. Because VirtualBox is open source, the community could feasibly support the product if Oracle stopped updating the platform. Because of these reasons, VirtualBox became the platform of choice when developing our cybersecurity curriculum.

There are several key advantages of using client-side virtualization for cybersecurity exercises. First, students can run cybersecurity exercises completely on their personal computers. Students would not need access to a dedicated campus computer lab, thereby reducing the infrastructure investment required and increasing accessibility. Another benefit is the ability to segment cybersecurity traffic from the rest of the network. It is possible to run client-side virtualization without any network connectivity, making it a good choice for students with network connectivity challenges.

Perhaps one of the key benefits of running cybersecurity exercises through virtual machines rather than using students' host operating systems is the ability to segment network traffic. A virtual machine can be configured so that network traffic never leaves the client machine. This segmentation prevents students from accidentally performing malicious actions on the network. Two key networking modes available in VirtualBox will be explained. With network address translation mode enabled, a virtual machine can connect to the internet, but the virtual machine cannot interact with other virtual machines running on the same guest. With internal networking, virtual machines on the same guest can communicate with each other but cannot connect to the internet. Students may need to switch network modes during exercises. For example, a student might use the network address translation mode to connect to the internet and download software packages, then switch to the internal networking mode to communicate with other virtual machines and prevent traffic from reaching other networks accidentally.

Some drawbacks exist in client-side virtualization that prevent it from being the definitive solution in cybersecurity exercise development. First, student computers may be limited by hardware capabilities. A single cybersecurity exercise might require several virtual servers to be running simultaneously. The exercises that can be conducted may be constrained by RAM, CPU, or hard disk. Modern Windows Server operating systems require at least 2GB RAM to run reasonably well, so running three virtual servers may be infeasible on older hardware. Each virtual server can take between 1-10GB on average. Computers with smaller solid-state disks typically perform well but often sacrifice capacity. Some hardware may simply not have enough capacity to install multiple operating systems. Second, supporting a wide number of devices can be challenging. Though modern virtualization software can be installed on Windows, MacOS, and Linux, differences between client configurations can lead to time spent troubleshooting virtualization software. For example, some laptops have virtualization features disabled in the BIOS by default which can be corrected, but can cause confusion. Also, anti-virus has interfered with the successful installation of virtualization software leading to problems completing exercises.

Server-side Virtualization

Microsoft's Hyper-V (*Hyper-V*, 2017) and VMWare vSphere (*vSphere*, 2017) are two popular server virtualization platforms used in the data center. At a high level, both platforms run virtual servers on top of physical hardware. Unlike client-side virtualization platforms that run virtual machines on top of a complete operating system layer, modern server-side platforms run virtual machines on a thin hypervisor layer which gives virtual machines more direct access to hardware resulting in better performance.

The VMWare vSphere Hypervisor is a free product with limited functionality that is installed directly on server hardware. Administrators install the vSphere Client on their computers and connect to the server to manage virtual machines. While vSphere Hypervisor is a low-cost option, key productivity features are missing (such as the ability to clone an existing server). Like VMWare vSphere Hypervisor, Microsoft Hyper-V Server 2016 is a free, thin virtualization layer that sits on top of the physical hardware. Hyper-V can also be enabled on a modern Windows Server by enabling the Hyper-V role.

Both vSphere and Hyper-V allow administrators to over-commit resources to accommodate more virtual hardware. For example, a physical server might have 32GB RAM. On that physical server, 32 virtual servers can be created and assigned 2GB RAM each. Because servers rarely use all available RAM, each virtual server should run fine despite the overallocation. Using either platform, licenses for Windows Server guest operating systems must be obtained as usual.

The VMWare and Microsoft platforms are both mature and good candidates for deploying virtual servers in a private cybersecurity lab. Choosing one platform over the other will likely be driven by vendor preference, licensing costs, or compliance with established information technology standards.

Major benefits of server-side virtualization include centralized control, scalability, and policy enforcement. Because administrators have complete control over the infrastructure, exercises can be designed and tested in a stable environment. Administrators can control all aspects of the environment from operating system versions, firewalls rules, and software installed. Students are less likely to have to spend time troubleshooting extraneous issues in a tightly controlled environment. Next, leveraging the same tools as large cloud providers, infrastructure can be built to scale computing capacity to meet student needs so that issues of students' computers lacking sufficient resources are negated. Lastly, because the infrastructure is centrally managed, policy regarding network traffic and acceptable use can be carefully monitored.

A clear drawback of server-side virtualization is initial investment. Hardware must be purchased, configured, and actively managed throughout its lifecycle. Dedicated lab administrators may need to be hired to manage the computing environment. Lastly, increased control comes at the cost of increased administrative burden. With client-side virtualization, students typically manage their own infrastructure. With server-side virtualization, the educator is responsible. Server-side virtualization can mean more time creating and configuring the infrastructure.

3. INFRASTRUCTURE AUTOMATION

Virtual machines frequently need specific software and configurations. Manual installation and configuration can be time consuming and error prone. Fortunately, infrastructure automation tools exist to streamline the process of creating virtual machines, installing software, and making configuration changes. The following sections describes some of the top tools that have emerged that can help educators be more effective.

Vagrant
Vagrant is a tool that makes provisioning and deprovisioning virtual machines efficient (*Vagrant*, 2017). Though Vagrant can be used to provision virtual machines on several virtualization platforms, emphasis will be given here on its integration with VirtualBox. A core concept in vagrant is a Vagrant box—a partially configured virtual machine used in lieu of an operating system DVD image for the creation of a virtual machine. The typical workflow for creating a virtual machine using VirtualBox without Vagrant involves downloading an ISO, creating a blank virtual machine, attaching the ISO, botting the virtual machine, following the installation prompt, and waiting for the operating system to be installed. The entire process can take dozens of clicks and 30 minutes of waiting.

Using Vagrant, a virtual machine can be created with just two commands: vagrant init [version]; vagrant up. Table 1 shows the commands needed to create a virtual machine in VirtualBox using Vagrant and connecting through SSH. The entire process takes approximately 4 minutes to complete.

```
C:\temp> vagrant init ubuntu/xenial64
C:\temp> vagrant up
C:\temp> vagrant ssh
```

Table 1: Creating an Ubuntu Server Virtual Machine with Vagrant

Creating virtual machines with Vagrant requires fewer steps, fewer decisions, and completes in less time. With Vagrant, the cost of breaking a virtual machine accidentally or intentionally is low because they can be recreated easily.

In addition to creating generic servers, instructors can create Vagrant configuration files that carry out post-installation configurations automatically. A Vagrantfile is a Ruby file that tells Vagrant which virtual machines to create along with basic configuration settings. A single cybersecurity exercise, such as address resolution protocol spoofing, might require three virtual machines. Table 2 shows a sample Vagrantfile that defines three virtual machines and assigns them private IP addresses. The Vagrantfile could be distributed to students via a learning management system. Then, students would copy the file to their hard drive, navigate to the folder in the command prompt, then run the command "vagrant up." The three virtual machines would then begin the boot process without any additional input required by the students.

A Vagrantfile that installs the Moodle learning management system on a single virtual server as part of the provisioning process is shown in Appendix A. Vagrant can leverage configuration management tools such as Ansible and Chef (described in the subsequent sections), but much can be done using shell scripting. The advantage of shell scripting is that special configuration management software is unnecessary. The downside to shell scripting is the potential time requirement needed to write and troubleshoot custom scripts.

The main benefit of Vagrant is the ease with which virtual machines can be created. The drawbacks include the learning curve to create Vagrantfiles. Vagrant also introduces another tool that must be updated. Both the Vagrant software and the boxes must periodically be updated to fix bugs and obtain the latest patches from operating system vendors. The learning curve for Vagrant is low for basic usage. Advanced Vagrant features can be introduced over time.

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure(2) do |config|

config.vm.provision "shell", inline: "echo
Starting victim, middle, and server"

config.vm.define "victim" do |victim|
  victim.vm.box = "ubuntu/trusty64"
  victim.vm.host_name = "victim"
  victim.vm.network "private_network",
     ip: "192.168.10.10"
end

config.vm.define "middle" do |middle|
  middle.vm.box = "ubuntu/trusty64"
  middle.vm.host_name = "middle"
  middle.vm.network "private_network",
    ip: "192.168.10.50"
end

config.vm.define "server" do |server|
  server.vm.box = "ubuntu/trusty64"
  server.vm.host_name = "server"
  server.vm.network "private_network",
     ip: "192.168.10.100"
end

end
```

Table 2: Vagrantfile Defining Three Servers

Ansible

Ansible is RedHat's infrastructure automation tool that aims to automate simple and complex infrastructures (*Ansible*, 2017). Administrators use the Ansible language to define playbooks. Playbooks describe system configurations that should be applied to target systems by the Ansible automation engine. A paid add-on, the Ansible Tower can monitor and apply playbooks on a large infrastructure. Ansible does not require custom agents to run on the server; instead it applies all changes over SSH.

The playbook in Table 3 demonstrates the human readable format of the Ansible language. An administrator with some Linux experience can understand that the playbook ensures that the Apache web server and PostgreSQL database servers are running. The example comes from the Ansible online documentation ("Intro to Playbooks," 2017).

```
---
- hosts: webservers
  remote_user: root

  tasks:
  - name: ensure apache is at the
latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2
dest=/etc/httpd.conf

- hosts: databases
  remote_user: root

  tasks:
  - name: ensure postgresql is at the
latest version
    yum: name=postgresql state=latest
  - name: ensure that postgresql is
started
    service: name=postgresql
state=started
```

Table 3: Sample Ansible Playbook

Ansible must be installed on a Linux control node—this is the only machine that needs Ansible installed. Unfortunately, the Linux requirement means that students running Windows or MacOS operating systems cannot use Ansible to create virtual machines on their own computers. For this reason, Ansible is most applicable for private labs where the instructor wants full control over the infrastructure.

An example will help illustrate the value of Ansible. Suppose an instructor wants to configure 50 virtual servers to ensure that they have nmap installed for a port scanning exercise. First, the instructor must create the Ansible playbook. Next, the instructor would run the following command to apply that playbook to all hosts defined in the playbook: "ansible-playbook cyber-exercise-1.yml." Ansible would login to each virtual server and install nmap if needed.

Key benefits of Ansible include a low learning curve compared to other infrastructure management tools, lack of an agent required on the remote servers, and scalability. However, the introduction of any configuration management tool requires that the administrator be trained in its use. And as mentioned previously, the Ansible controller must run Linux, though the controller does not have to be a dedicated server.

Chef

Chef is one of the first widely used infrastructure automation platforms (*Chef*, 2017). Chef is open source software created by Chef Software, Inc. Like Ansible, Chef requires a master controller to communicate with nodes. One difference is that Chef's master must be a dedicated server, whereas the Ansible client can be run from any Linux computer. Also, whereas Ansible performs configuration changes on nodes using SSH, Chef communicates to custom Chef agents on the nodes.

Chef's documentation and tutorials are extensive. However, the learning curve for using Chef is steeper than Ansible. For example, the introduction tutorial for infrastructure automation is estimated to take eight hours. The new Chef user can quickly become overwhelmed with cooking related tools—recipes, knives, supermarkets, kitchens, etc. But the maturity of the platform and extensive features make it a robust solution for complex infrastructure needs.

Chef starts with a cookbook. Cookbooks are written in the Ruby programming language—a dynamic language with similar readability to Python. A cookbook contains one or more recipes, files, libraries, attributes, and additional environment information. Recipes instruct chef how to configure the system. Cookbooks are registered on the chef server and pushed out to chef nodes.

To create a cybersecurity exercise using Chef, an instructor would first need to create a cookbook. The Chef Development Kit comes with command line tools to create a skeleton cookbook. Recipes must be added to the cookbook. Table 4 shows a sample recipe to install the Apache web server.

This recipe would be saved in a Ruby .rb file within the cookbook folder.

```
package "apache2" do
  action :install
end
```

Table 4: Sample Chef Recipe

When the recipe is complete, the knife tool is used to upload the cookbook to the chef server. Then, the knife command can be used to run the recipe on nodes.

Because Chef an Ansible are similar tools, their benefits can be directly compared. Chef's advantages include extensive documentation, and active community, and a mature, tested solution. Compared to Ansible, Chef has a steeper learning curve, is more complex, and uses more computing resources. Chef is more likely than Ansible to force the educator to spend time managing the configuration management.

4. CONCLUSIONS

Advances in virtualization and infrastructure automation tools make it easier than ever to develop and deploy cybersecurity exercises. No single tool can solve all infrastructure and configuration management challenges. For the organizations that lack dedicated lab environments, the Vagrant and VirtualBox combination is a mature, flexible solution for creating virtual environments on the fly quickly. For organizations with dedicated lab environments, Ansible, Chef, and server-side virtualization platforms are solutions that should be explored. Ansible is a lighter weight solution for organizations that want to take their first steps using configuration management. Chef would be an appropriate choice for more complex configuration scenarios, though the steeper learning curve than Ansible should be taken under consideration.

The technologies shared in this paper have been evaluated for fit in a cybersecurity program. It is hoped that educators continue to share cybersecurity exercise best practices so that the discipline can move forward quickly to meet the increasing need for qualified professionals.

5. REFERENCES

Ansible. (2017). Retrieved from https://www.ansible.com/

Chef. (2017). Retrieved from https://www.chef.io

Hyper-V. (2017). Retrieved from https://www.microsoft.com/en-us/cloud-platform/server-virtualization

Intro to Playbooks. (2017). Retrieved June 14, 2017, from http://docs.ansible.com/ansible/playbooks_intro.html#playbook-language-example

NSA / DHS. (2013). *NSA / DHS National Centers of Academic Excellence in Cyber Defense (CD) Knowledge Units* (pp. 1–73). Retrieved from https://www.iad.gov/NIETP/documents/Requirements/CAE-CD_Knowledge_Units.pdf

Vagrant. (2017). Retrieved from https://www.vagrantup.com/

VirtualBox. (2017). Retrieved from https://www.virtualbox.org/

VMWare Workstation Player. (2017). Retrieved from http://www.vmware.com/products/player/playerpro-evaluation.html

vSphere. (2017). Retrieved from https://www.vmware.com/products/vsphere.html

Appendices and Annexures

Appendix A – Additional Tables and Figures

Vagrantfile that Installs Moodle
The following code can be used to install the Moodle learning management system as part of the virtual machine provisioning process. The Vagrantfile could be distributed to students via a course website. Student would download the Vagrantfile to their computers, open a command prompt, navigate to the folder with the Vagrant file and run "vagrant up." Vagrant would then create a new virtual machine, download required packages, and complete the Moodle installation in the background. When the process completes, students could open a web browser and view the Moodle installation at http://localhost:8888.

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/xenial64"
  config.vm.network "public_network", ip: "192.168.1.99"
  config.vm.synced_folder "./", "/vagrant_share"

  #Provisioning instructions leveraged from:
  # https://docs.moodle.org/31/en/Step-by-step_Installation_Guide_for_Ubuntu
config.vm.provision "shell", inline: <<-SHELL
sudo su
echo "nameserver 8.8.8.8" > /etc/resolv.conf #Fix DNS resolving "bug" in Xenail
apt-get update
debconf-set-selections <<< 'mysql-server mysql-server/root_password password mysqladmin'
debconf-set-selections <<< 'mysql-server mysql-server/root_password_again password mysqladmin'
apt-get -y install mysql-server
apt-get -y install apache2 mysql-client php7.0 libapache2-mod-php7.0
apt-get -y install graphviz aspell php7.0-pspell php7.0-curl php7.0-gd php7.0-intl php7.0-mysql
php7.0-xml php7.0-xmlrpc php7.0-ldap php7.0-zip
echo "Installing php-mbstring and php-soap (optional Moodle components)"
apt-get -y install php-mbstring php-soap
service apache2 restart
apt-get -y install git-core
cd /opt
git clone git://git.moodle.org/moodle.git
cd /opt/moodle
git branch --track MOODLE_32_STABLE origin/MOODLE_32_STABLE
git checkout MOODLE_32_STABLE
cp -R /opt/moodle /var/www/html/
mkdir /var/moodledata
chown -R www-data /var/moodledata
chmod -R 777 /var/moodledata
chmod -R 0755 /var/www/html/moodle

echo "default_storage_engine = innodb" >> /etc/mysql/mysql.conf.d/mysqld.cnf
echo "innodb_file_per_table = 1" >> /etc/mysql/mysql.conf.d/mysqld.cnf
echo "innodb_file_format = Barracuda" >> /etc/mysql/mysql.conf.d/mysqld.cnf
service mysql restart
mysql -u root --password=mysqladmin -e "CREATE DATABASE moodle DEFAULT CHARACTER SET utf8
COLLATE utf8_unicode_ci;"
mysql -u root --password=mysqladmin -e "create user 'moodleadmin'@'localhost' IDENTIFIED BY
'moodleadmin';"
```

```
mysql -u root --password=mysqladmin -e "GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,CREATE
TEMPORARY TABLES,DROP,INDEX,ALTER ON moodle.* TO moodleadmin@localhost IDENTIFIED BY
'moodleadmin';"

echo "<?php  // Moodle configuration file

unset(\\$CFG);
global \\$CFG;
\\$CFG = new stdClass();

\\$CFG->dbtype    = 'mysqli';
\\$CFG->dblibrary = 'native';
\\$CFG->dbhost    = 'localhost';
\\$CFG->dbname    = 'moodle';
\\$CFG->dbuser    = 'moodleadmin';
\\$CFG->dbpass    = 'moodleadmin';
\\$CFG->prefix    = 'mdl_';
\\$CFG->dboptions = array (
  'dbpersist' => 0,
  'dbport' => '',
  'dbsocket' => '',
);

\\$CFG->wwwroot   = 'http://127.0.0.1:8888';
\\$CFG->dataroot  = '/var/moodledata';
\\$CFG->admin     = 'admin';

\\$CFG->directorypermissions = 0777;

require_once(__DIR__ . '/lib/setup.php');
" >> /var/www/html/moodle/config.php
echo "Changing the Document root"
sed -i "s/DocumentRoot \\\/var\\\/www\\\/html/DocumentRoot \\\/var\\\/www\\\/html\\\/moodle/"
/etc/apache2/sites-available/000-default.conf
service apache2 restart
echo "The installation has completed."
echo "Open a browser on your host and go to http://127.0.0.1:8888 to complete configurations."
SHELL

end
```