



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Network Monitoring and Threat Detection In-Depth (Security 503)"
at <http://www.giac.org/registration/gcia>

Using Vagrant to Build a Manageable and Sharable Intrusion Detection Lab

GIAC (GCIA) Gold Certification

Author: Shaun McCullough, cybergoof@gmail.com

Advisor: Adam Kliarsky

Accepted Date: September 2, 2016

Abstract

This paper investigates how the Vagrant software application can be used by Information Security (InfoSec) professionals looking to provide their audience with an infrastructure environment to accompany their research. InfoSec professionals conducting research or publishing write-ups can provide opportunities for their audience to replicate or walk through the research themselves in their own environment. Vagrant is a popular DevOps tool for providing portable and repeatable production environments for application developers, and may solve the needs of the InfoSec professional. This paper will investigate how Vagrant works, the pros and cons of the technology, and how it is typically used. The paper describes how to build or repurpose three environments, highlighting different features of Vagrant. Finally, the paper will discuss lessons learned.

1. Introduction to DevOps

In a recent survey by RightScale, 74% of respondents said they implemented some form of DevOps in their organization in 2016 (Weins, 2016). Gartner defines DevOps as “represents a change in IT culture, focusing on rapid IT service delivery through the adoption of agile, lean practices in the context of a system-oriented approach. DevOps emphasizes people (and culture), and seeks to improve collaboration between organizations and development teams. DevOps implementations utilize technology -- especially automation tools that can leverage and increasingly programmable and dynamic infrastructure from a life cycle perspective” (Woods, 2016).

In that definition, the key phrase is “improve collaboration between operations and development teams”. While working as a web application developer in the 1990’s, there were strict processes and purposeful barriers in place to move an application from the development team to the operations team. We had multiple problems with the development teams working off a different architecture configuration from that of the operations team. Development teams might throw their new software “over the wall” to the IT ops teams, who had to figure out how to place this round peg in their square hole infrastructure. To fix this, organizations implemented Sequential Engineering which put in place processes, oversight, change management meetings, and complex staging areas (Wikipedia, n.d.). This took time, cost money, and kept teams from truly being agile.

As Gartner explained, DevOps processes will utilize technology to improve collaboration and automation. The Vagrant software was created to bridge those teams together, and get IT operations and developers working off the same infrastructure configuration (HashiCorp, n.d.).

2. Vagrant Basics

Vagrant is a tool for building complete virtual environments. It supports a workflow that focuses on automation and lowers the environment setup time (HashiCorp, n.d.). Vagrant was started in January of 2010 by Mitchell Hashimoto as a side project. In late 2012, HashiCorp was formed to provide an entire suite of products to support a full DevOps workflow. As a bonus, most of the products from HashiCorp are open source.

Shaun McCullough, cybergoof@gmail.com

Vagrant provides an easy to configure, reproducible and portable work environment built on top of industry standard virtual machine (VM) providers (HashCorp, n.d.). Vagrant providers can be VirtualBox, VMWare, Amazon Web Services (AWS), or any number of other providers. Vagrant also supports many standard provisioning tools such as Chef, Puppet, Ansible, or even straight shell scripts. The beauty of Vagrant is that it allows separation of the underlining VM provider technologies from the chosen provisioning tools, and manages these configurations through a standardized Vagrantfile.

For the developer, Vagrant will isolate dependencies and configurations within a single, disposable, consistent environment while still supporting all the tools a user needs. Vagrant is just a tool, and it's up to the DevOps team to build the right workflow and processes to support it, but Vagrant certainly helps.

2.1. A Docker Comparison

When talking about DevOps support tools, Docker is one of the more well-known capabilities. Although Docker has a similar goal as Vagrant, the approach is very different. Docker's goal is to achieve agility and control for Development and IT Operations teams to build, ship and run any app, anywhere (Docker, n.d.). Docker supports DevOps processes by providing a generalized container for an app to run on, and that container can be shipped and run on any number of Docker supported infrastructures such as AWS (Amazon, n.d.), Azure (Iain Foulds, n.d.), or DigitalOcean (Digitalocean, n.d.). A quick Google search yielded a large number of cloud services that support Docker Containers.

The main difference in the Vagrant and Docker approaches is that Vagrant deploys a full virtual environment, while Docker deploys the bare minimum "container" to support running an app onto a standardize infrastructure. A particularly interesting topic on www.stackoverflow.com discussed the difference between Docker and Vagrant (Hykes, 2013). In this thread, the authors of Docker and Vagrant chimed in to discuss their views of what was different about their products. Both authors are obviously champions of their own product, and the feature lists have grown in the last 3 years since

Shaun McCullough, cybergoof@gmail.com

the post was made, but their responses provide a good insight into their goals of their projects.

To test out a single application, such as using a Docker instance of Snort (Coolacid, n.d.) to monitor a vulnerable instance of Wordpress (Vibioh, 2015), or running malware analysis apps then Docker will work great (Zeltser, 2014). However, for an InfoSec professional researching and sharing a realistic and feature rich target environment, then the container architecture of Docker does not adequately match a real target host environment. Furthermore, Docker containers use the resource isolation features of the Linux kernel such as cgroups and kernel namespaces while avoiding the overhead of the entire virtual machine (wikipedia, n.d.). Therefore, Docker apps run inside Linux containers. Building a test environment that simulates psexec lateral movement or monitoring for Mimikatz credential harvesting on Windows machines is impossible at this time.

Microsoft has started investing in the Docker movement by tooling the Windows Server 2016 to support Windows containers through Docker. Using the Windows Docker Service on Windows Server 2016 will support Windows container images (Peterson, Windows Container Images, 2016). However, the container sizes are huge (Peterson, Windows Containers on Windows Server, 2016), seeming to limit the idea of lite weight containers for each app.

I have developed software projects using Docker and Vagrant in the past for my job. I found the Docker projects were more complicated when the software stack, networking, or non-volatile data storage was more complicated and required multiple applications. At its core, Vagrant uses Virtual Machine infrastructures which is a well-known and well used container. You do need an entire OS running on a VM, but that tradeoff is acceptable. This author believes Vagrant provides an easier, more realistic implementation for a lab environment and will focus this paper on Vagrant technologies.

2.2. Using Vagrant

Vagrant is deceptively easy to use. One can create Vagrant environments to deploy to any number of provider systems such as AWS, Digital Ocean, Google,

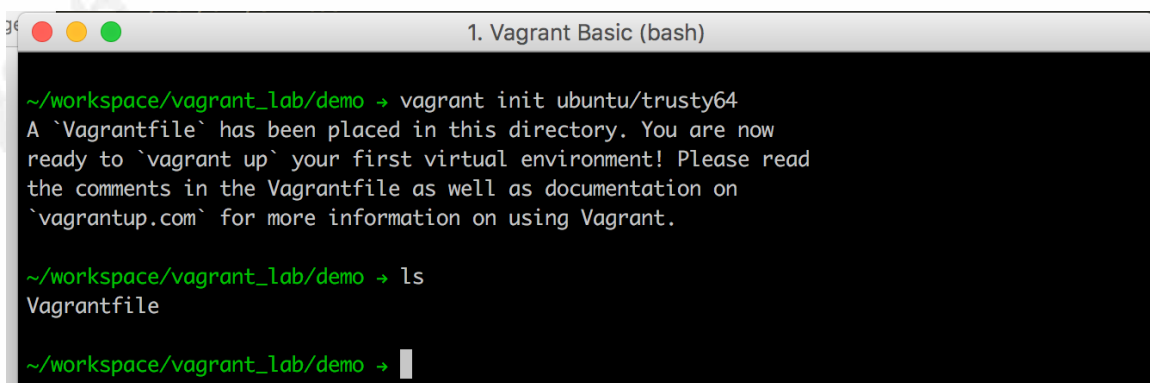
Shaun McCullough, cybergoof@gmail.com

OpenStack (Hashimoto, n.d.). For an InfoSec professional looking to share infrastructure environments across a wide range of audience members, local host VirtualBox technology provides a low cost and ease of use. VMware Fusion or Workstations is also an option, but requires the purchase of a Vagrant Connector for \$79 in addition to the cost of the VMWare product (HashiCorp Pricing, n.d.)

To get started using Vagrant, install VirtualBox (www.virtualbox.org/wiki/Downloads) and install Vagrant (www.vagrantup.com/docs/installation/). Vagrant builds an environment from a VM image with added metadata, packaged into a Vagrant box file (HashiCorp Boxes, n.d.). A box file is built for a particular provider and can be used by anyone to bring up an identical working environment. That is the real power of Vagrant, the ability to easily box up the entire work environment for easy sharing. A Vagrant box file could be the entire server already provisioned and configured for a specific research scenario.

Vagrant provides an easy mechanism to search and download box files that have already been created. HashiCorp's Atlas website at atlas.hashicorp.com/boxes/search provides an easy way to search for box files based on deployment provider, operating system, or keywords. The labs in this paper rely on the Trusty64 base image provided by Ubuntu available for download at atlas.hashicorp.com/ubuntu/boxes/trusty64.

Since the box file is available directly from HashiCorp, the local Vagrant application can pull what is needed from the website. First, create a working directory for the new build environment. This directory is the Vagrant project root directory. Then simply initiate the new Vagrant box file with “vagrant init ubuntu/trusty64” (Figure 1).



```

1. Vagrant Basic (bash)

~/workspace/vagrant_lab/demo → vagrant init ubuntu/trusty64
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.

~/workspace/vagrant_lab/demo → ls
Vagrantfile

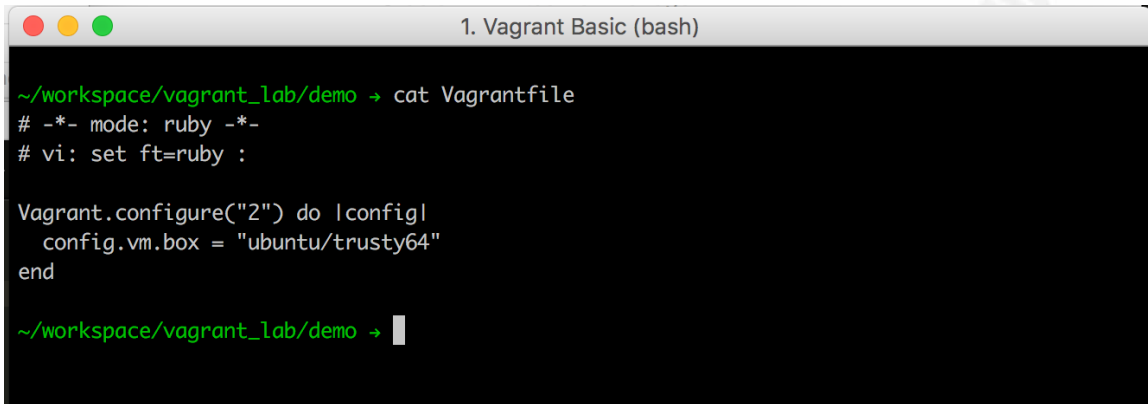
~/workspace/vagrant_lab/demo →

```

Figure 1: Initialize Vagrant environment

Shaun McCullough, cybergooof@gmail.com

Running the initialize command created a file called *Vagrantfile*. The *Vagrantfile* is the instruction book Vagrant uses to build an environment (Figure 2). The *Vagrantfile* is used to identify what box file to use, how to configure the environment's network, how to sync folders between the host and the guest, and specific information for each supported provider, VirtualBox in this case. Vagrant provides more details in its help documents online at www.vagrantup.com/docs/vagrantfile.



```
1. Vagrant Basic (bash)

~/workspace/vagrant_lab/demo → cat Vagrantfile
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/trusty64"
end

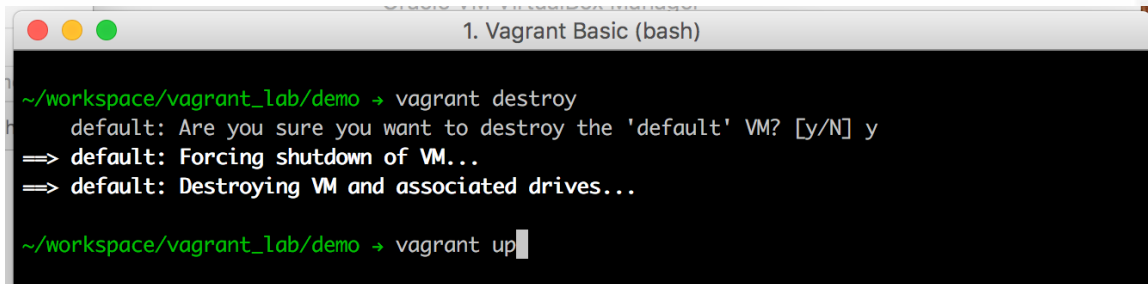
~/workspace/vagrant_lab/demo →
```

Figure 2: Inside Vagrantfile

The *Vagrantfile* identifies the base box file for the environment. To stand up the environment, simply run “vagrant up”. Vagrant looks for the *Vagrantfile* in the current directory and sees the need for the Ubuntu/Trusty64 box. If it has not been downloaded before, Vagrant will reach out to the atlas.hashicorp.com site and download this particular box file.

This lab uses the Ubuntu/Trusty64 base system to build and provision a specific environment. The original trusty64 box file, that was downloaded to the local host, will never change when provisioning a system; it is always the original Trusty64 base box file.

Vagrant is meant to easily create disposable and reloadable environments. To completely wipe away and rebuild an environment just requires two easy commands, “destroy” and then “up” again (Figure 3).

A terminal window titled "1. Vagrant Basic (bash)" with a dark background and green text. The prompt is ~/workspace/vagrant_lab/demo. The user enters 'vagrant destroy'. The output shows a confirmation message: 'default: Are you sure you want to destroy the 'default' VM? [y/N] y'. The user enters 'y'. The output continues: '=> default: Forcing shutdown of VM...' and '=> default: Destroying VM and associated drives...'. The user then enters 'vagrant up' and the prompt returns to ~/workspace/vagrant_lab/demo.

```
~/workspace/vagrant_lab/demo → vagrant destroy
default: Are you sure you want to destroy the 'default' VM? [y/N] y
=> default: Forcing shutdown of VM...
=> default: Destroying VM and associated drives...

~/workspace/vagrant_lab/demo → vagrant up
```

Figure 3: Destroy and up Vagrant Environment

Section 3 of this paper demonstrates some of the ways that an InfoSec professional can build and share full working environments that are easy for the audience to acquire and start using.

3. Vagrant Labs

3.1. Vagrant as a Shareable Infrastructure

An InfoSec professional can spend countless hours researching new intrusion detection techniques, documenting an improved analysis approach, or building exercises for students to work through. Vagrant can provide a simple and repeatable process so the audience can immediately download, provision, and begin interacting with exact copies of the environment for themselves.

When learning to detect an attack on a web server, the Damn Vulnerable Web Application (DVWA) is a feature rich application that lets a user work through 18 different web application attacks in nine categories. Once installed and running in an environment, monitoring the network traffic to this web application would allow someone to learn how to analyze a web application attack.

Installing the DVWA can be tricky, especially if the audience is unfamiliar with standing up PHP/MySQL applications. With over 60 lines of instructions and commands, it might take a while to get it right (nightmare-rg, n.d.).

Some Vagrant users have recreated the DVWA in Vagrant, which makes it simple to stand up a full DVWA image. This Lab demonstrates setting up an available DVWA Vagrant image to get an environment started quickly, and how to capture pcap files to analyze the attacks.

First, in a working directory, clone the DVWA Github project from user nightmare-rg with the command “git clone https://github.com/nightmare-rg/dvwa-vagrant.git”. Next, change directories into the newly created Vagrant project directory “dvwa-vagrant”. Cloning the project brought down a *Vagrantfile*, a *bootstrap.sh* file, and the source code needed to run the DVWA (Figure 4). The *Vagrantfile* identifies the

location of the box file to use, the ports that must be forwarded from the guest to the host, and VirtualBox specific configurations (Figure 5).

```
sans@lab
$ git clone https://github.com/nightmare-rg/dvwa-vagrant.git
Cloning into 'dvwa-vagrant'...
remote: Counting objects: 652, done.
remote: Total 652 (delta 0), reused 0 (delta 0), pack-reused 652
Receiving objects: 100% (652/652), 1.01 MiB | 0 bytes/s, done.
Resolving deltas: 100% (84/84), done.
Checking connectivity... done.

sans@lab
$ cd dvwa-vagrant/

sans@lab
$ ls
bootstrap.sh*  dvwa.sql  dvwa-1.0.8/  README.md  Vagrantfile  z_custom.ini
```

Figure 4: Building DVWA environment

```
Vagrant.configure("2") do |config|

  config.vm.box = "trusty"
  config.vm.box_url = "https://cloud-images.ubuntu.com/vagrant/trusty/current/trusty-server-cloudimg-amd64-vagrant-disk1.box"

  config.vm.network :forwarded_port, guest: 80, host: 8088

  config.vm.provider :virtualbox do |vb|
    vb.gui = false
  end

  config.vm.provision :shell, :path => "bootstrap.sh"
end
```

Figure 5: DVWA Vagrantfile

DVWA uses the basic Ubuntu Trusty box file. The Vagrant file then calls the “shell” provision command to run bootstrap.sh from inside the newly created VM (Figure 6). DVWA could have picked from any of the 13 described provisioning technologies (HashiCorp Provisioning, n.d.). Building images for a production environment will probably require a more sophisticated provisioning system such as Puppet, Ansible, or

Chef. However, a shell based provisioning is very easy to read and understand by the audience.

```
set -o nounset                                # Treat unset variables as an error

apt-get update

sudo debconf-set-selections <<< 'mysql-server-5.5 mysql-server/root_password password vmdev
sudo debconf-set-selections <<< 'mysql-server-5.5 mysql-server/root_password_again password
mdev'
sudo apt-get update
sudo apt-get -y install mysql-server-5.5 php5-mysql apache2 php5 vim

rm -rf /var/www/*
mkdir -p /var/www
ln -s /vagrant/dvwa-1.0.8 /var/www/html

mysql -u root -pvmdev < /vagrant/dvwa.sql

mkdir -p /etc/php5/conf.d
ln -s /vagrant/z_custom.ini /etc/php5/conf.d/z_custom.ini

sudo service apache2 restart
```

Figure 6: DVWA's bootstrap.sh file

Once the project is downloaded, starting the environment is easy as the command “vagrant up” (Figure 7).

```
sans@lab
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'trusty'...
==> default: Matching MAC address for NAT networking...
==> default: Setting the name of the VM: dvwa-vagrant_default_1468690297026_12753
==> default: Clearing any previously set forwarded ports...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
default: Adapter 1: nat
==> default: Forwarding ports...
default: 80 (guest) => 8088 (host) (adapter 1)
default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
default: SSH address: 127.0.0.1:2222
default: SSH username: vagrant
default: SSH auth method: private key
default: Warning: Remote connection disconnect. Retrying...
```

Figure 7: Provisioning DVWA environment

It only took three commands and less than three minutes to go from nothing to a fully functioning web server running DVWA with Apache. Once the provisioning is finished, going to “<http://localhost:8088>” brings up the DVWA webpage (Figure 8).

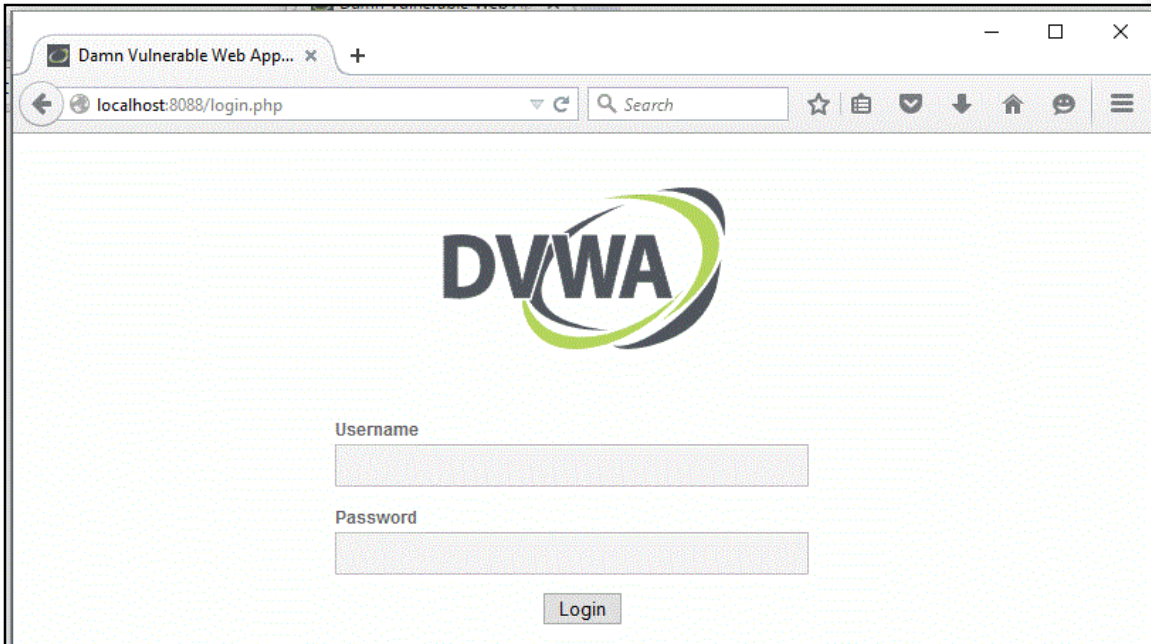


Figure 8: DVWA Webpage

Capturing the network traffic between the host system and the DVWA takes a few more commands. VirtualBox provides tools to capture all network traffic to a VM and store it in a pcap file. First, use Vagrant to halt the Virtual Machine in order to make the network modifications (Figure 9).

```
sans@lab
$ vagrant halt
==> default: Attempting graceful shutdown of VM...
```

Figure 9: Halt Vagrant environment

Next, use the program *VboxManage.exe* to modify the Virtual Machine to turn on NIC Tracing and output it to a file in *c:\temp* (Figure 10).

```
sans@lab  
$ 'c:\Program Files\Oracle\VirtualBox\VboxManage.exe' modifyvm dvwa --nictrace1 on --nictracefile1 c:\temp\dvwa.pcap
```

Figure 10: Turn on NIC tracing

Now, all network traffic coming across the first interface will output to *dvwa.pcap*. Restart the DVWA with “vagrant up” again, re-open the DVWA web application from the host browser, and perform a very basic SQL Injection (Figure 11).

Vulnerability: SQL Injection

User ID:

ID: ' OR '1'='1
First name: admin
Surname: admin

ID: ' OR '1'='1
First name: Gordon
Surname: Brown

ID: ' OR '1'='1
First name: Hack
Surname: Me

ID: ' OR '1'='1
First name: Pablo
Surname: Picasso

Figure 11: Perform SQL Injection

Opening the pcap file in Wireshark, the HTTP Get with the SQL Injection attack is easily identified (Figure 12). Using a Vagrant environment as the target environment, and collecting the pcap from a network interface can allow the audience to analyze attack, and setup for the environment took a matter of minutes.

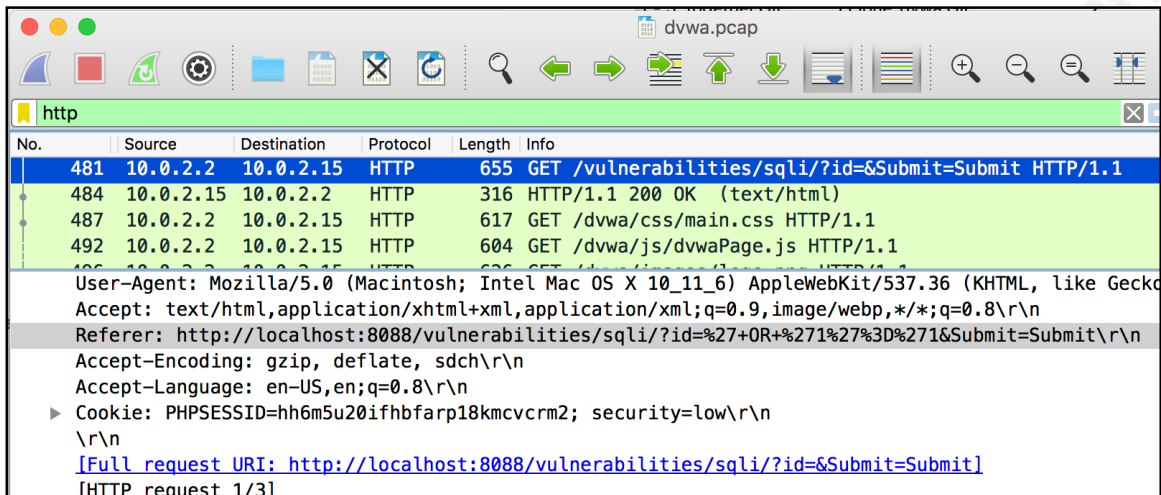


Figure 12: Wireshark output

3.2. Creating and Sharing a Vagrant Environment

The last lab demonstrated how easy it was to download and setup a Vagrant environment to run a web application attack, then capture and analyze the network traffic. Only a few commands to stand up and setup the entire environment. Using the prepackaged and tested Vagrant environment enables the audience to quickly get to the heart of the research without troubleshooting the infrastructure and server installation problems. This lab will investigate how to create a new Vagrant environment from scratch and package it up for easy distribution.

The target server will be an Apache web server running on Ubuntu/Trusty64. Snort will be installed on the server to monitor network connections. In this lab, a second Kali2 VM will be running to conduct a denial of service (DoS) attack against the Apache server.

Creating a new Vagrant environment starts with a *Vagrantfile* (Figure 13). In order to use the second Kali VM to attack the server, Vagrant needs to use a *private_network* and assign an IP Address. The *private_network* is the same as *Host-only Adapter*.

```
~/workspace/vagrant_lab/lab2 → cat Vagrantfile
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/trusty64"
  config.vm.network "forwarded_port", guest: 80, host: 8080
  config.vm.network "private_network", ip: "192.168.33.10"
  config.vm.provision :shell, :path => "bootstrap.sh"
end

~/workspace/vagrant_lab/lab2 →
```

Figure 13: Simple Vagrantfile

In this lab, the provisioning work will be done through a bootstrap file that Vagrantfile will run in a shell command. The file *bootstrap.sh* does all the configuration and setup of the server (Figure 14). First, it updates the base Ubuntu operating system. Next, it will install the Apache2 web server. Last, it installs the necessary Snort applications. Since this paper is more focused on how Vagrant can be used in a lab environment, rather than a focus on Snort, the Snort configuration is simple.

```
#####
#!/usr/bin/env bashs

# Updating and prepping the base image
sudo apt-get update -y
sudo apt-get upgrade -y
sudo apt-get install -y build-essential

# Installing apache2
sudo apt-get install -y apache2

# Turn off gro and lro
sudo apt-get install -y ethtool
sudo echo "post-up ethtool -K eth0 gro off" >> /etc/network/interfaces
sudo echo "post-up ethtool -K eth0 lro off" >> /etc/network/interfaces

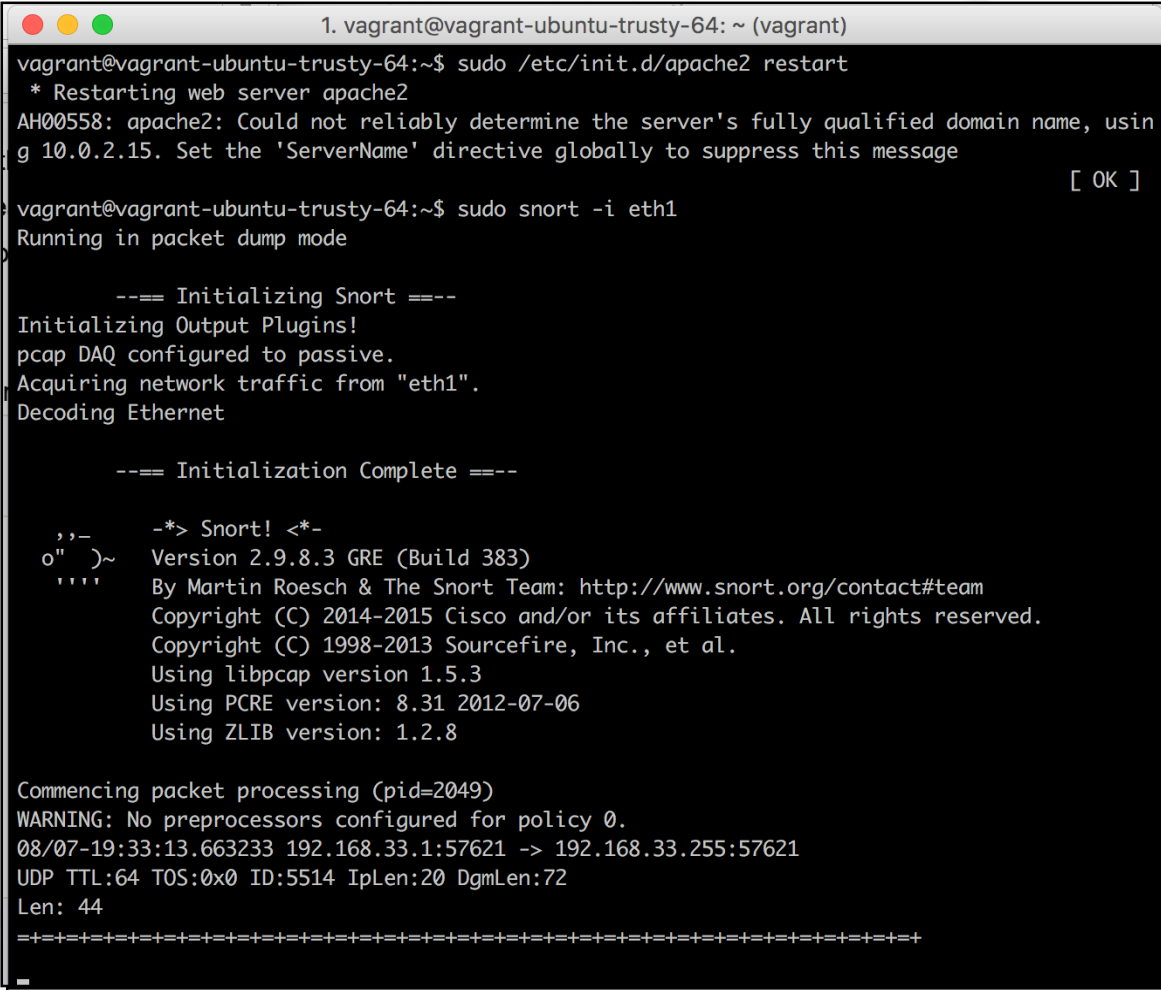
sudo apt-get install -y libpcap-dev libpcrc3-dev libdumbnet-dev bison flex
sudo apt-get install -y zlib1g-dev liblzma-dev openssl libssl-dev
# Installing Snort
mkdir /vagrant/snort_src && cd /vagrant/snort_src
wget https://www.snort.org/downloads/snort/daq-2.0.6.tar.gz
tar -xvzf daq-2.0.6.tar.gz
cd daq-2.0.6
./configure
make
sudo make install

cd /vagrant/snort_src
wget https://snort.org/downloads/snort/snort-2.9.8.3.tar.gz
tar -xvzf snort-2.9*.tar.gz
cd snort-2.9.*
./configure --enable-sourcefire
make
sudo make install
# update shared libraries
sudo ldconfig

# Create symlink to the snort binary in /usr/sbin
sudo ln -s /usr/local/bin/snort /usr/sbin/snorts
~/workspace/vagrant_lab/lab2 →
```

Figure 14: Lab2 Vagrantfile

The following command, “vagrant up”, will start up the VM, update the networking, and run the *bootstrap.sh* file from within the VM. Once complete, the command “vagrant ssh” from the host computer will create an SSH session between the host and the Vagrant VM using the Vagrant created user “vagrant”. Next, just start Apache and Snort to start monitoring the interface (Figure 15). From the host web browser, go to <http://192.168.33.10> to see that apache is working and available (Figure 16).



```

1. vagrant@vagrant-ubuntu-trusty-64: ~ (vagrant)
vagrant@vagrant-ubuntu-trusty-64:~$ sudo /etc/init.d/apache2 restart
* Restarting web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
10.0.2.15. Set the 'ServerName' directive globally to suppress this message
[ OK ]
vagrant@vagrant-ubuntu-trusty-64:~$ sudo snort -i eth1
Running in packet dump mode

    === Initializing Snort ===
Initializing Output Plugins!
pcap DAQ configured to passive.
Acquiring network traffic from "eth1".
Decoding Ethernet

    === Initialization Complete ===

    ,,-      -*> Snort! <*-
    o"  )~    Version 2.9.8.3 GRE (Build 383)
    '""      By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
              Copyright (C) 2014-2015 Cisco and/or its affiliates. All rights reserved.
              Copyright (C) 1998-2013 Sourcefire, Inc., et al.
              Using libpcap version 1.5.3
              Using PCRE version: 8.31 2012-07-06
              Using ZLIB version: 1.2.8

Commencing packet processing (pid=2049)
WARNING: No preprocessors configured for policy 0.
08/07-19:33:13.663233 192.168.33.1:57621 -> 192.168.33.255:57621
UDP TTL:64 TOS:0x0 ID:5514 IpLen:20 DgmLen:72
Len: 44
=====

```

Figure 15: Restarting Apach2 and Snort

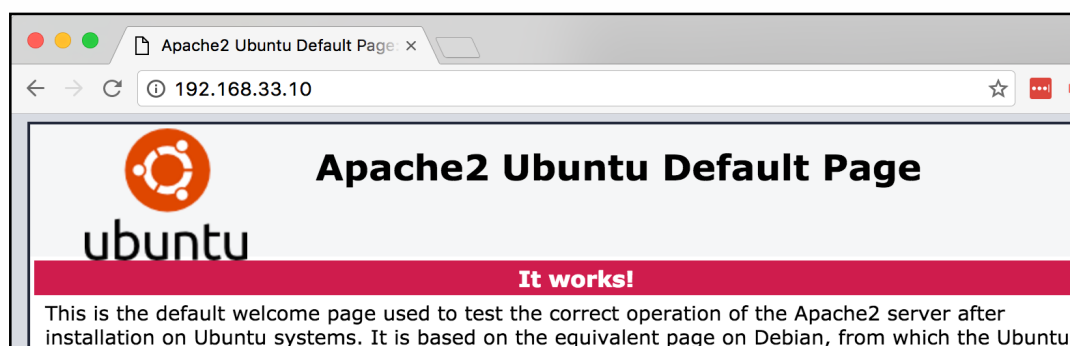


Figure 16: Apache2 default webpage

Now, Snort is monitoring the interface and should detect the denial of service (DoS) attack. The `apache_mod_isapi` attack in Metasploit should be detected by Snort, but will not actually DoS your newly installed Apache server (Rapid7, 2016). Figure 17 shows the commands to run the `apache_mod_isapi` DoS attempt in Metasploit. Figure 18 is the snort output when snort is monitoring, but without any rules loaded.

```

root@kali: ~
File Edit View Search Terminal Help
msf auxiliary(apache_mod_isapi) > use auxiliary/dos/http/apache_mod_isapi
msf auxiliary(apache_mod_isapi) > set rhost 192.168.33.10
rhost => 192.168.33.10
msf auxiliary(apache_mod_isapi) > exploit

[*] Causing the ISAPI dll to be loaded and unloaded...
[*] Triggering the crash ...
[*] Auxiliary module execution completed
msf auxiliary(apache_mod_isapi) > exploit

[*] Causing the ISAPI dll to be loaded and unloaded...
[*] Triggering the crash ...
[*] Auxiliary module execution completed
msf auxiliary(apache_mod_isapi) > $$

```

Figure 17: Metasploit auxiliary apache_mod_isapi

```

=====
WARNING: No preprocessors configured for policy 0.
08/21-22:03:10.240474 192.168.33.1:65522 -> 192.168.33.10:80
TCP TTL:64 TOS:0x0 ID:25997 IpLen:20 DgmLen:52 DF
***A***F Seq: 0xEA557EA4 Ack: 0x3A65926A Win: 0x1015 TcpLen: 32
TCP Options (3) => NOP NOP TS: 175963503 578
=====

08/21-22:03:10.240513 192.168.33.10:80 -> 192.168.33.1:65522
TCP TTL:64 TOS:0x0 ID:36907 IpLen:20 DgmLen:64 DF
***A**** Seq: 0x3A659445 Ack: 0xEA557EA5 Win: 0x21F TcpLen: 44
TCP Options (6) => NOP NOP TS: 578 175963503 NOP NOP Sack: 59989@32420
=====

08/21-22:03:10.240547 192.168.33.10:80 -> 192.168.33.1:65522
TCP TTL:64 TOS:0x0 ID:36908 IpLen:20 DgmLen:64 DF
***A**** Seq: 0x3A659445 Ack: 0xEA557EA5 Win: 0x21F TcpLen: 44
TCP Options (6) => NOP NOP TS: 578 175963503 NOP NOP Sack: 59989@32420
=====

```

Figure 18: Snort detecting DoS attack

Creating a *Vagrantfile* and *bootstrap.sh* file that provisions a basic Ubuntu image into the desired target environment is nice, but Vagrant can support an even easier build and share solution. Vagrant can package the entire VM, provisioned and configured as desired, into a new Vagrant box file and made available on <https://atlas.hashicorp.com> for anyone to use, hosted for free.

Returning to the host terminal, to the directory with the *Vagrantfile*, use the command “vagrant package” to package up a new box file (HashiCorp Package, n.d.). In the case of Figure 19, the file will be named *lab2.box*.

```

1. vagrant@vagrant-ubuntu-trusty-64: ~ (bash)

~/workspace/vagrant_lab/lab2 → vagrant package --output lab2.box --vagrantfile Vagrantfile
=> default: Attempting graceful shutdown of VM...
=> default: Clearing any previously set forwarded ports...
=> default: Exporting VM...
=> default: Compressing package to: /Users/alchemist/workspace/vagrant_lab/lab2/lab2.box
=> default: Packaging additional file: Vagrantfile

~/workspace/vagrant_lab/lab2 → ls
Vagrantfile*  bootstrap.sh*  lab2.box      snort_src/

```

Figure 19: Command to package box file

This *lab2.box* file is a new Vagrant box file that can be used to create a whole new, fully provisioned Vagrant environment. After creating an account on atlas.hashicorp.com, a new box file can be uploaded and made available to the world. Atlas hosts the box file for free, provides the ability to tag the box file with labels or keywords for easier discovery, and supports basic versioning (Figure 20). Downloading, installing and running the image is as simple as calling “vagrant init” on the desired box file (Figure 21).

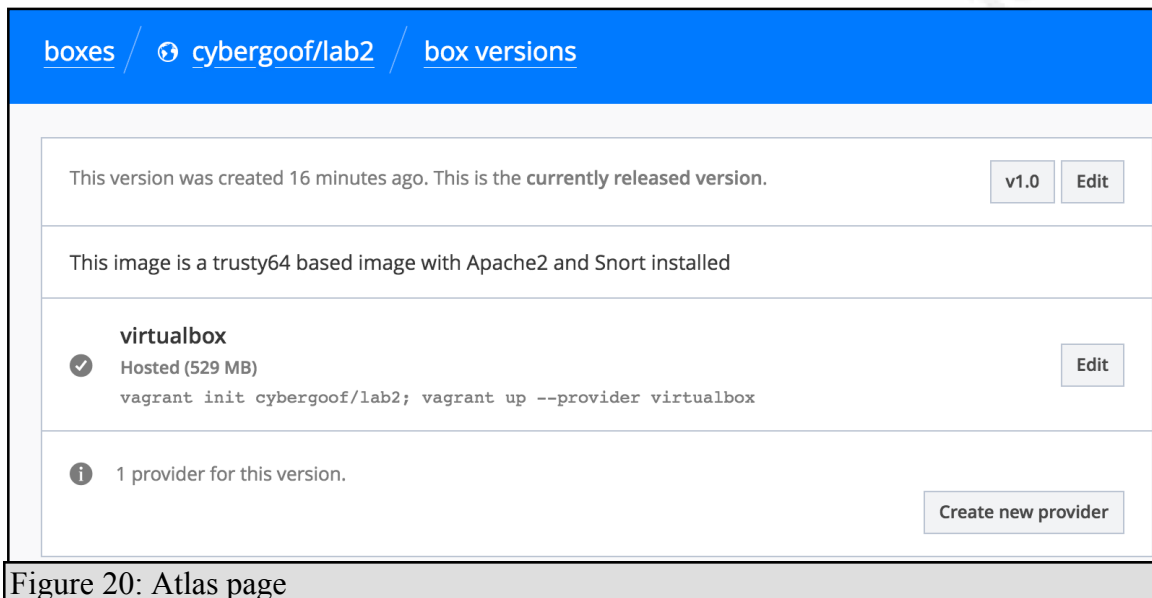


Figure 20: Atlas page

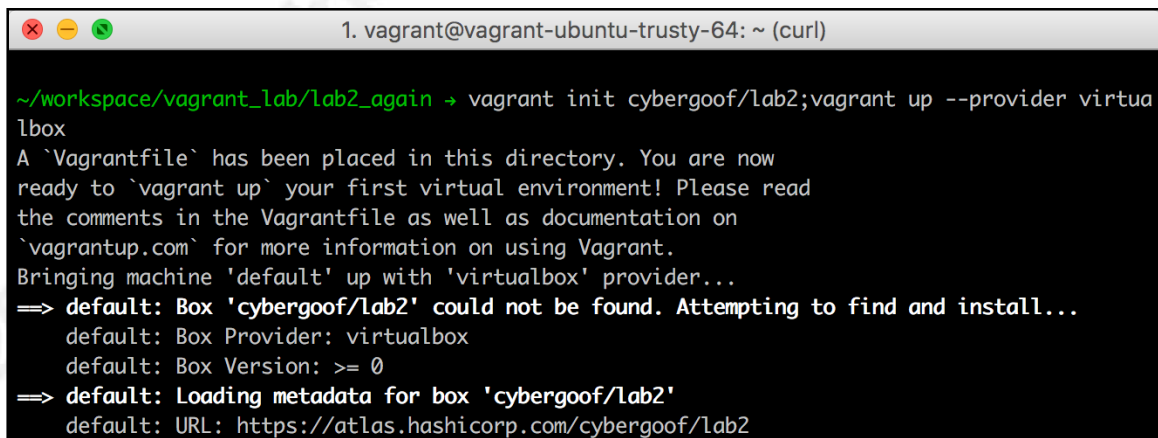


Figure 21: Initializing new environment

This lab showed how easy it is to package an environment and share it with anyone running Vagrant and VirtualBox. Either building off a base operating system

Shaun McCullough, cybergooft@gmail.com

(OS) so the audience can make any necessary changes, or simply providing the entire installed and configured server as a packaged box file. InfoSec professionals can package their complete environment and provide it without any need for additional provisioning. This can eliminate provisioning problems if someone were to attempt to run the research years later and the original software packages are no longer available. At this point, this paper has shown how to use Linux based guest operating systems. The final lab will show how to use Windows as a guest OS and some of the problems it brings.

Shaun McCullough, cybergooof@gmail.com

3.3. Working with Windows

Besides the Linux environments in the previous lab, Vagrant also supports Microsoft Windows VM's, but there are not as many available Vagrant environments as in Linux. Microsoft does not allow redistribution of their operating system by users (Microsoft, 2015). Searching the Atlas site does show some Windows based Vagrant box files. Microsoft itself uses Atlas to distribute box files, but at the time of this paper, only Windows 10 was provided at <http://atlas.hashicorp.com/Microsoft>. For InfoSec professionals needing Windows 7 and 8.1, Microsoft's Developer website has box files available at <https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/> (Figure 22).

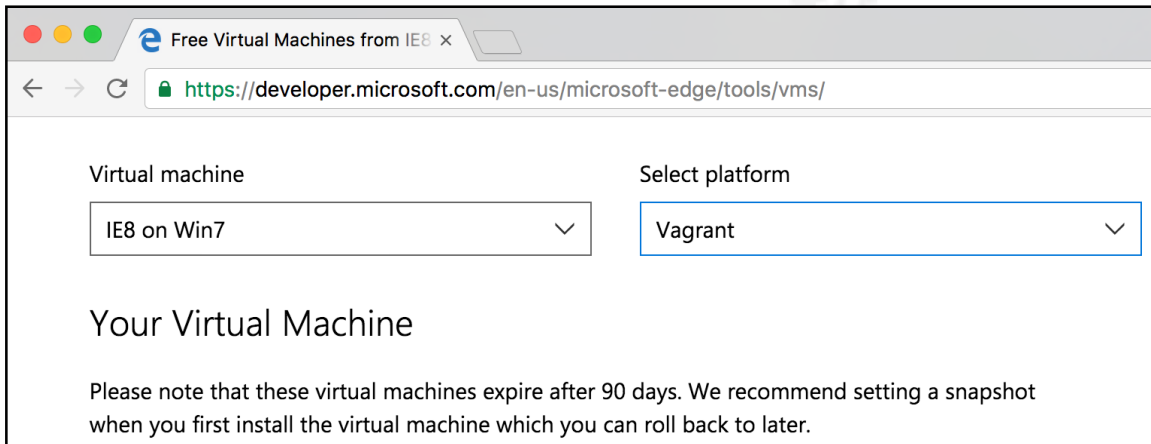
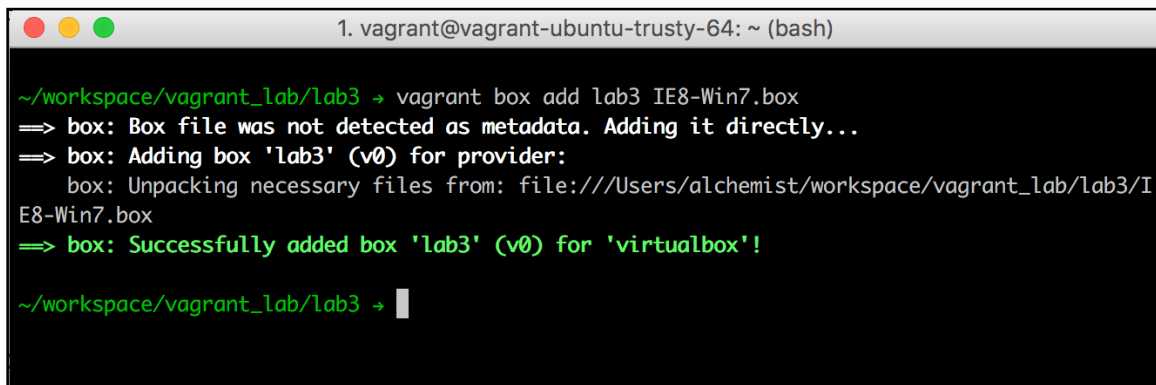


Figure 22: Downloading Windows Vagrant environment

This lab will use a Vagrant box file from the Microsoft Developer website. Once the box file is unzipped, add the newly downloaded Vagrant box file to the local Vagrant repository. The command “vagrant box add lab3 IE8-Win7.box” will do this (Figure 23).

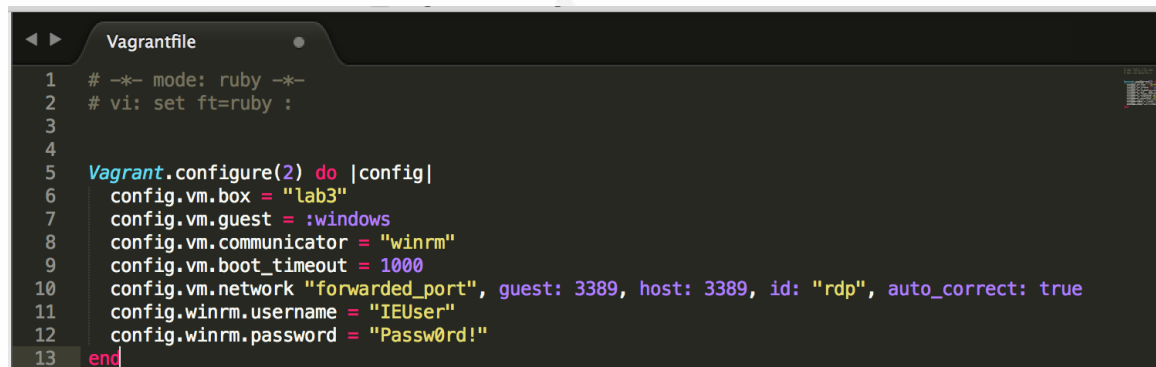


```
1. vagrant@vagrant-ubuntu-trusty-64: ~ (bash)
~/workspace/vagrant_lab/lab3 → vagrant box add lab3 IE8-Win7.box
⇒ box: Box file was not detected as metadata. Adding it directly...
⇒ box: Adding box 'lab3' (v0) for provider:
    box: Unpacking necessary files from: file:///Users/alchemist/workspace/vagrant_lab/lab3/IE8-Win7.box
⇒ box: Successfully added box 'lab3' (v0) for 'virtualbox'!

~/workspace/vagrant_lab/lab3 →
```

Figure 23: Add box file to Vagrant

The box file is now called *lab3*. Now, initializing the Vagrant environment is as simple as “vagrant init lab3” and a new generic *Vagrantfile* has been created. Oddly, Microsoft did not configure the Windows 7, 8.1 or 10 box files to support Vagrant provisioning. The *Vagrantfile* must be changed to use WinRM, network ports opened and the proper username/password set. Also, once started, Windows will need to be configured with the proper networking type and turn on WinRM.



```
Vagrantfile
1 # -*- mode: ruby -*-
2 # vi: set ft=ruby :
3
4
5 Vagrant.configure(2) do |config|
6   config.vm.box = "lab3"
7   config.vm.guest = :windows
8   config.vm.communicator = "winrm"
9   config.vm.boot_timeout = 1000
10  config.vm.network "forwarded_port", guest: 3389, host: 3389, id: "rdp", auto_correct: true
11  config.winrm.username = "IEUser"
12  config.winrm.password = "Passw0rd!"
13 end
```

Figure 24: Windows Vagrantfile after changes

After changing the *Vagrantfile* as show in Figure 24, issue the “vagrant up” command. The guest Windows OS GUI is now available. However, looking at the host terminal, Vagrant is failing to authenticate with the newly started Windows VM (Figure 25).

```

=> default: Booting VM...
=> default: Waiting for machine to boot. This may take a few minutes...
default: SSH address: 127.0.0.1:2222
default: SSH username: vagrant
default: SSH auth method: private key
default: Warning: Authentication failure. Retrying...
default: Warning: Authentication failure. Retrying...
default: Warning: Authentication failure. Retrying...
default: Warning: Authentication failure. Retrying...

```

Figure 25: Windows Vagrant environment starting

The guest Windows OS needs WinRM configured in order for Vagrant to interact with it. As soon as the Windows GUI is available, interact with the GUI and follow the configuration based on feedback from Andre Pereira (andreptb, 2015). First, set the networking to either Home or Work (Figure 26).

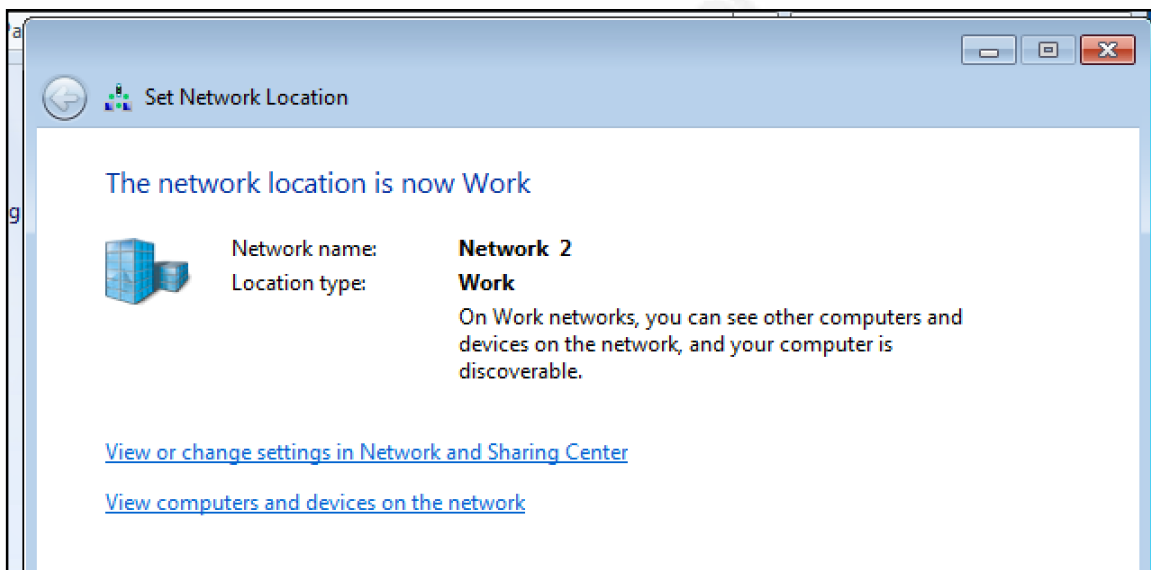


Figure 26: Windows networking set to "work"

Next, run the following script as administrator on the guest Windows OS (Figure 27).


```
@echo off
set WINRM_EXEC=call %SYSTEMROOT%\System32\winrm
%WINRM_EXEC% quickconfig -q
%WINRM_EXEC% set winrm/config/winrs @{MaxMemoryPerShellMB="300"}
%WINRM_EXEC% set winrm/config @{MaxTimeoutms="1800000"}
%WINRM_EXEC% set winrm/config/client/auth @{Basic="true"}
%WINRM_EXEC% set winrm/config/service @{AllowUnencrypted="true"}
%WINRM_EXEC% set winrm/config/service/auth @{Basic="true"}
```

Figure 27: Script to run on Windows guest

Returning to the host terminal, it should start communicating and eventually complete with success (Figure 28). The configuration can be verified by going to the guest Windows OS validating that a shared directory named *c:\vagrant* is now there. This directory is shared with the base Vagrant root directory on the host.

```
=> default: Waiting for machine to boot. This may take a few minutes...
default: WinRM address: 127.0.0.1:55985
default: WinRM username: IEUser
default: WinRM execution_time_limit: PT2H
default: WinRM transport: negotiate
=> default: Machine booted and ready!
=> default: Checking for guest additions in VM...
default: The guest additions on this VM do not match the installed version of
default: VirtualBox! In most cases this is fine, but in rare cases it can
default: prevent things such as shared folders from working properly. If you see
default: shared folder errors, please make sure the guest additions within the
default: virtual machine match the version of VirtualBox you have installed on
default: your host and reload your VM.
default:
default: Guest Additions Version: 5.0.4
default: VirtualBox Version: 5.1
=> default: Mounting shared folders...
default: /vagrant => /Users/alchemist/workspace/vagrant_lab/lab3
```

Figure 28: Successful Vagrant environment provisioning

Sharing research involving a Windows OS is not nearly as easy as the first two labs. However, it only takes a few more steps to configure the Microsoft OS, and writing the research steps with the Vagrant Windows OS will ensure that all audience members are starting with the same configuration. The shared *c:\vagrant* directory can be used to provide scripts that will perform all the necessary configurations.

Shaun McCullough, cybergooof@gmail.com

4. Putting it into Practice

4.1. The Pitfalls of Vagrant

There are a few limitations when strictly using a Vagrant and VirtualBox solution for sharing environments. When attempting to setup a lab network at work with a Windows VM and a Linux VM operating in Bridge mode, so that the Linux VM could monitor the subnet, Vagrant itself had problems interacting with the Windows VM over the primary NAT adapter. This appeared to be a limitation in VirtualBox rather than Vagrant. VMWare's virtual switch technology may provide a more realistic and effective networking environment; however, the Vagrant/VMWare connector must be purchased (HashiCorp Package, n.d.). The cost is not prohibitive at \$79, but that will limit the potential audience for particular labs.

When I first started using Vagrant, I was primarily working from a Windows Host running Linux and Windows VM environments. I found some inconsistencies in Vagrant errors, more complications when trying to interact with Vagrant over SSH, and a couple of software bugs only found in the Windows implementation (cybergoof, 2016). I found the Vagrant workflow works a little better from a Mac OSX host. This is not a show stopper, but something to keep in mind.

Anyone can create and share a Vagrant environment, and the quality of that environment is up to the individual. There is no rating system for Vagrant images, like the Stack Exchange (www.stackexchange.com) use of a merit and badge system to support quality questions and answers. Unlike a Github project where the source code is exposed, a packaged Vagrant box file could harbor less than desirable software. However, in the context of InfoSec professionals providing packaged environments to support an exercise or research, the audience will have to decide the trustworthiness of the source. In general, I would recommend anyone providing the a fully packaged environment through Atlas also provide a *Vagrantfile* and bootstrap scripts in Github so that the audience can build, and alter, the configuration as desired.

Vagrant does have a larger ecosystem of tools to support DevOps. Vagrant and Packer (www.packer.io) is heavily focused on the development, while HashiCorp cloud

Shaun McCullough, cybergoof@gmail.com

services such as Atlas (atlas.hashicorp.com) and Terraform (www.terraform.io) can help deliver environments for multiple provider hosting options and even manage them. These products could be used by InfoSec professionals who needs a more sophisticated lab environment, is part of a larger teams, or is building and supporting organized training.

4.2. Conclusion

Vagrant appears to be a great tool for InfoSec professionals looking for an easy to use, free, and portable toolset to package and share consistent environments. Researchers can build Vagrant images that demonstrate their research, best practices, or exercises for others to use and participate. Vagrant easily uses VirtualBox and supports Windows or Linux host, enabling a wide range of exercise environments. More sophisticated network based environments may be more difficult with just VirtualBox, but implementation in VMWare or AWS may be appropriate for those advanced audiences.

References

- Amazon. (n.d.). *Docker Basics*. Retrieved from Amazon Web Services:
<http://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html>
- andreptb. (2015, 02 16). *Setup ModernIE Vagrant Boxes*. Retrieved from GitHub:
<https://gist.github.com/andreptb/57e388df5e881937e62a>
- Coolacid. (n.d.). *docker-snort*. Retrieved from GitHub:
<https://github.com/coolacid/docker-snort>
- cybergoof. (2016, 01 27). *Atlas Push Not Working on Windows*. Retrieved from GitHub:
<https://github.com/mitchellh/vagrant/issues/6938>
- Digitalocean. (n.d.). *Docker on Digital Ocean*. Retrieved from DigitalOcean:
<https://www.digitalocean.com/community/tutorials/how-to-use-the-digitalocean-docker-application>
- Docker. (n.d.). *Why Docker*. Retrieved from Docker: <https://www.docker.com/enterprise>
- HashiCorp Boxes. (n.d.). *Boxes*. Retrieved from Vagrant Up:
<https://www.vagrantup.com/docs/boxes.html>
- HashiCorp Package. (n.d.). *Package*. Retrieved from Vagrant Up:
<https://www.vagrantup.com/docs/cli/package.html>
- HashiCorp Packer. (n.d.). *Packer*. Retrieved from Packer: <https://www.packer.io/>
- HashiCorp Pricing. (n.d.). *Pricing & Purchase*. Retrieved from Vagrant Up:
<https://www.vagrantup.com/vmware/#buy-now>
- HashiCorp Provisioning. (n.d.). *Provisioning*. Retrieved from Vagrant Up:
<https://www.vagrantup.com/docs/provisioning/>
- HashiCorp. (n.d.). *Why Vagrant*. Retrieved from Vagrant Up:
<https://www.vagrantup.com/docs/why-vagrant/>
- Hykes, M. a. (2013). *StackOverflow*. Retrieved from StackOverflow:
<http://stackoverflow.com/questions/16647069/should-i-use-vagrant-or-docker-for-creating-an-isolated-environment>

- Iain Foulds. (n.d.). *Using the Docker VM Extension to deploy your environment*. Retrieved from Microsoft Azure: <https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-linux-dockerextension/>
- Microsoft. (2015, 08 15). *License Terms*. Retrieved from Windows.net: http://modernievirt.blob.core.windows.net/vhd/release_notes_license_terms_1_5_15.pdf
- mitchellh. (n.d.). *Vagrant*. Retrieved from Github: <https://github.com/mitchellh/vagrant/wiki/Available-Vagrant-Plugins>
- nightmare-rg. (n.d.). *dvwa-vagrant*. Retrieved from GitHub: <https://github.com/nightmare-rg/dvwa-vagrant>
- Peterson, N. (2016, 5 25). *Windows Container Images*. Retrieved from msdn.microsoft.com: https://msdn.microsoft.com/en-us/virtualization/windowscontainers/management/manage_images
- Peterson, N. (2016, 5 27). *Windows Containers on Windows Server*. Retrieved from msdn.microsoft.com: https://msdn.microsoft.com/en-us/virtualization/windowscontainers/quick_start/quick_start_windows_server
- Rapid7. (2016). *Apache Range Header DOS*. Retrieved from Rapid7: https://www.rapid7.com/db/modules/auxiliary/dos/http/apache_range_dos
- Vibioh. (2015, 06 18). *vibioh/wordpress*. Retrieved from Docker Hub: <https://hub.docker.com/r/vibioh/wordpress/builds/bvvpvgxfwd2svkdjwxdpnpx/>
- Weins, K. (2016, May 11). *Cloud Management Blog*. Retrieved from RightScale: <http://www.rightscale.com/blog/cloud-industry-insights/new-devops-trends-2016-state-cloud-survey>
- Wikipedia. (n.d.). *Traditional Engineering*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Traditional_engineering
- Woods, V. (2016, April 6). *The Science of DevOps Decoded*. Retrieved from Gartner: <http://www.gartner.com/smarterwithgartner/the-science-of-devops-decoded/>
- Zeltser, L. (2014, 12 10). *Running Malware Analysis Apps as Docker Containers*. Retrieved from SANS Digital Forensics and Incident Response Blog: <https://digital-forensics.sans.org/blog/2014/12/10/running-malware-analysis-apps-as-docker-containers>

Shaun McCullough, cybergoof@gmail.com