A Using the Cloud for Malware Analysis

There may be several reasons to use a cloud service as a platform for malware analysis.

1. The user's own laptop (or desktop) computer may not have enough CPU power, or memory, to run several VMs at once
2. The user's computer may not support virtualization, and therefore not be able to run VMWware or VirtualBox at all. This is the case for some Apple products at the time of this writing.
3. The user may want to analyze malware in the cloud in order to provide a extra level of protection in the possible but unlikely case the malware is able to escape.
4. The user may want to have access to the same malware analysis environment from a variety of physical locations, such as home, work, or when on travel.

Cloud service providers, or CSPs, offer their customers access to a variety of virtual machines. This variety may include a choice of operating systems, number of CPUs, memory, disk storage, and software configurations. Selecting the right configuration is mostly a trade-off between capability and budget. Many CSPs offer a free tier of services, and charge more depending on the size of the virtual machine, and the level of access desired. For the student malware analyst, free or at most "pay as you go" is probably the best choice.

Before describing how to set up a Windows desktop suitable for malware analysis using a particular cloud service, we need to discuss nested virtualization. Nested virtualization can be described as the ability to run a virtual machine inside another virtual machine. If one acquires a Windows or Linux desktop running in the cloud, that desktop is actually a VM running under the control of a hypervisor (such as Xen), and it's the hypervisor that is running on real, bare metal hardware. At this time, not all cloud service providers make nested virtualization available.

Whatever CSP we use, we intend to connect to the cloud platform using a virtual network controller, or VNC. Several VNC servers are freely available on the Internet. The Remote Desktop Protocol, or RDP, would be an option were we to connect to a Windows VM from a Windows PC. For the sake of licensing if no other reason, Linux VMs seem to be the more common CSP offering. In what follows, we describe the creation of a Linux VM, and how to connect to it using VNC.

A.1 Google Compute Engine

The Google Cloud Platform (GCP), and the Google Compute Engine in particular, is the example we use. The place to start is the GCP Console, which is accessed at https://console.cloud.google.com/. The GCP documentation suggests that more testing has been done with nested virtualization on Debian, so that's the Linux distribution we use in the following. But we caution the reader that systems change, and what worked for us may not work for them. Details large and small may need to be updated. We found this web site to be helpful [16]. We decided to use the xfce4 desktop, as described at https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-vnc-on-debian-10

From the GCP Console, create a new VM instance, taking note of its external IP, which in this example is 34.162.107.197. The IP is ephemeral, in the sense that GCP may assign a different external IP the next this VM is started. We configured a new virtual machine instance with the following parameters:

- 4 cores and 16 GB RAM (e2-standard-4)
- latest version of Debian
- a default disk, at least 100 gigabytes, since OVA files and VMs are big
- enable display device (which may not be necessary)
- allow default access to all Cloud APIs (which may not be necessary)
- enable HTTP and HTTPS

Take note of the zone in which the machine is created, which happens to be "us-central1-a". Start this VM, and connect with ssh. The resulting shell is a root shell, but it lacks a password. So the obvious first step is to set a root password using: `sudo passwd root`.

To configure the xfce4 desktop, from the newly password-protected shell, issue these commands. Installing tasksel takes a while, but it makes installing xfce4 much easier.

```
sudo apt-get update && sudo apt-get upgrade
sudo apt install tasksel -y
sudo tasksel
```

[16] https://tecadmin.net/how-to-install-vnc-server-on-debian-10/

Invoke tasksel as shown above, select the xfce4 option, and tab down to "Okay". Once xfce4 is installed, install a VNC server as shown below. This too will take a few minutes. Once all this is done, we can invoke the vncserver. This last command will prompt for a password. Test it using netcat. If you need to install netcat, do so as shown. Try to connect to port 5901. If that succeeds, kill the server because some additional configuration work needs to be done. The process is summarized here:

```
sudo apt install tightvncserver
sudo apt-get install autocutsel
sudo apt-get install dbus-x11
sudo apt-get install -y netcat
vncserver
(prompts and confirms a password)
nc localhost 5901   -- gives a short RFB message in reply
vncserver -kill :1
```

Use a text editor such as `vi` or `nano` to modify `.vnc/xstartup`

```
#!/bin/sh
xrdb $HOME/.Xresources
startxfce4 &
```

We should now be able to connect to our new Debian instance using a VNC client, such as VNC Viewer. In the GCE version of Debian, VirtualBox is not installed by default. So we visit the Debian site for package files and instructions[17]. Having downloaded a Debian binary from the Virtualbox web site, we referred to the instructions to see how to load some repositories and header files. To actually run VirtualBox, however, it turns out that nested virtualization needs to be explicitly enabled for the Debian instance using the GCP shell. Stop the newly created Debian instance, and following the directions at GCP[18] we end the ssh session and issue this command:

```
gcloud compute instances export VM_NAME \  % "instance-6"
  --destination=YAML_FILE_PATH \ % "yaml.txt"
  --zone=ZONE % "us-central1-a"
```

Still within the GCP shell, use a text editor to add the following to the YAML file:

[17] https://wiki.debian.org/
[18] https://cloud.google.com/compute/docs/instances/nested-virtualization/

```
advancedMachineFeatures:
  enableNestedVirtualization: true
```

Save that file, and:

```
gcloud compute instances update-from-file VM_NAME \ % "instance-6"
  --source=FILE_PATH \      % "yaml.txt"
  --most-disruptive-allowed-action=RESTART \
  --zone=ZONE   % "us-central1-a"
```

using the same substitutions as before. We then connect to the machine using ssh, and confirm that the command `grep -cw vmx /proc/cpuinfo` returns something non-zero. (The GCP documentation says to make sure the VM is running on an Intel Haswell or later.)

A.2 Running VirtualBox

We can now attempt to run VirtualBox on the new Debian instance. We found it useful to have small OVA files for Ubuntu and Windows Server available for such testing. When trying to run a new VM, VirtualBox might complain about the VirtualBox Linux kernel driver being not loaded or not set up correctly. The `/sbin/vboxconfig` command may in turn complain about missing header files. Correct this as follows:

```
sudo apt update
sudo apt upgrade
sudo apt install linux-headers-$(uname -r)$
sudo /sbin/vboxconfig
```

Given the size of OVA files, and that we may need several of them, we decided to add a standard disk, which we need to format. The GCP console has links to instructions. We set up the disk /dev/sdb to be mounted at /mount/disks/disk1, with fstab configured to mount it automatically at boot time. (In a situation where several students in the same class may need read-only access to a set of ISO or OVA files, the disk can be shared between virtual machine instances belonging to the same GCP project.)

We now have the ability to run virtual machines, using VirtualBox, but those machines will not be able to access the Internet until they can acquire IP addresses. From a Debian instance shell, issue the command `VBoxManage modifyvm aNewVM --natdnsproxy1 on` where "aNewVM" is replaced with

the VM's name in VirtualBox. This will allow the guest operating systems to go out to the Internet for updates, and web browsers running on the guest will be able to reach the Internet. See Figure 20.

Once the various guest systems are updated and stable, it's not a bad idea to export them as VirtualBox appliances, that is, as OVA files. Likewise, once the GCP instance is able is able to support VNC connections and run VirtualBox successfully, creating an image of that instance as a backup is appropriate.

The process of starting a VM, connecting to it with VNC, and using VirtualBox to create virtual machines can be summarized as follows:

- Go to the GCP Compute Engine VM Instances page
- Select the VM instance to be started, and start it
- Note the external IP address `<extIP>`
- Use SSH (right menu) to connect to that VM once it has started
- from that shell, run `vncserver -geometry 1600x1000`
- use VNC Viewer to connect to ¡extIP¿:5901
- when the remote desktop appears, use VNC Viewer properties to zoom as much as 200 %
- invoke VirtualBox from the new desktop
- if creating or importing a new VM into VirtualBox, issue the VBoxManage modifyvm command as shown above.
- Keep the CPU and memory configuration minimal, e.g. one CPU and four GBs of RAM. More can be added later, once the new VirtualBox VM is stable
- Set paravirtualization to default, unless experience shows different. Linux prefers KVM, while Windows prefers Hyper-V.
- Network should be NAT, cable connected
- install Guest Additions as desired
- do whatever you need to do...take snapshots as desired
- shut VirtualBox down cleanly

- close the VNC Viewer session
- if desired, run `vncserver -kill :1` to stop that server
- MAKE SURE to shutdown the VM instance on the Instances page

When building a new VM, whether from an ISO image or an OVA file, it's probably better to build the new VM on an ordinary (non-cloud) desktop. Let it do updates there. Once the new system is updated and stable, create an OVA file and move that up to the cloud, where it can be unpacked and used.

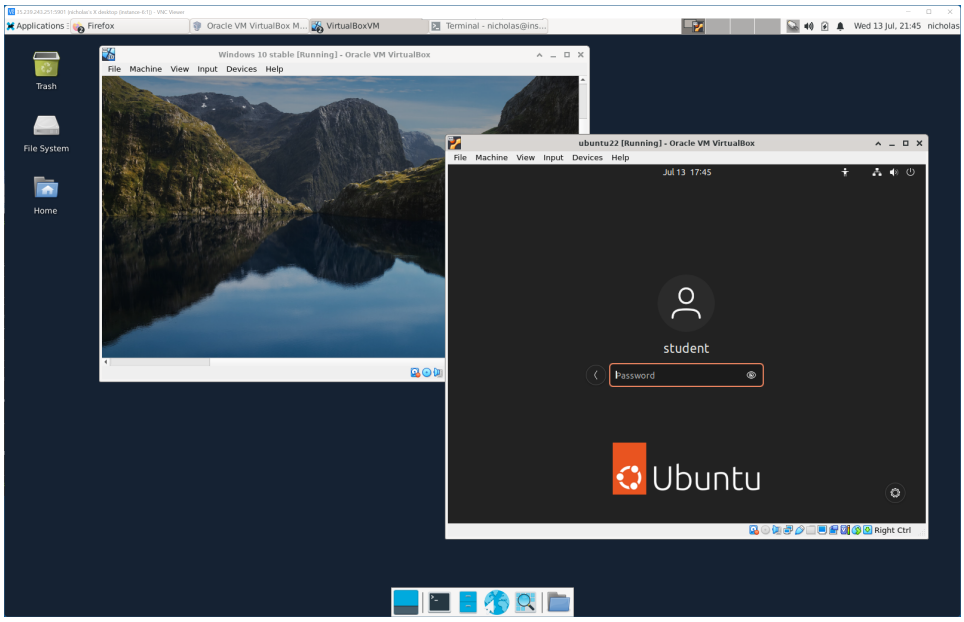

Figure 20: Running Ubuntu and Windows 10 as guests on a Debian GCP instance