eForensics Magazine

MAGAZINE

Cuckoo Sandbox and Malware Analysis

A view into Cuckoo's internals and emulation flow

Extracting Sample Information using Cuckoo API with Python

The First Responder CSIRT on a SECURE Drive

VOL.08 NO.03
ISSUE 03/2019, (88) MARCH
ISSN 2300 6986

TEAM

Editor-in-Chief
Joanna Kretowicz
joanna.kretowicz@eforensicsmag.com

Managing Editor:

Dominika Zdrodowska
dominika.zdrodowska@eforensicsmag.com

Editors:

Marta Sienicka
sienicka.marta@haking.com

Marta Strzelec
marta.strzelec@eforensicsmag.com

Bartek Adach
bartek.adach@pentestmag.com

Senior Consultant/Publisher:
Paweł Marciniak

CEO:
Joanna Kretowicz
joanna.kretowicz@eforensicsmag.com

Marketing Director:
Joanna Kretowicz
joanna.kretowicz@eforensicsmag.com

DTP
Dominika Zdrodowska
dominika.zdrodowska@eforensicsmag.com

Cover Design
Hiep Nguyen Duc

Publisher
Haking Media Sp. z o.o.
02-676 Warszawa
ul. Postępu 17D
Phone: 1 917 338 3631

www.eforensicsmag.com

word from the team

Dear Readers,

This month our main focus is on the leading open source automated malware analysis system - Cuckoo Sandbox.

In this issue, entitled “Cuckoo Sandbox and Malware Analysis”, you will find articles on automated malware analysis with Cuckoo, extracting sample information using Cuckoo API with Python, Cuckoo’s internals and emulation flow, and it’s not all that’s inside this system!

But to give you an overview of what else to expect in this issue, I have to mention that we also have an amazing article on Mobile CSIRT Toolkit, great publication on narrowing down the location of an image, an article, entitled “Real-Time Network Intrusion Detection using SNORT”, and a paper about Buffer Overflow and Integer Overflow.

We’re more than excited! Also, after reading the articles - please share your opinion with us! We added a new feature on our website and now users are able to leave a review of the issue. Don’t hesitate to do it!

As always – we want to thank all authors, betatesters, and proofreaders for participating in this project.

Have a nice read and fun with Cuckoo!

Regards,

Dominika Zdrodowska

and the eForensics Magazine Editorial Team

eForensics Magazine

Contents

Cuckoo Sandbox - what it is, how to install it, submitting suspicious file into sandbox and the analysis report

by Sébastien RAMELLA

Cuckoo Sandbox is an indispensable tool adapted to today's computer world to answer the malware threat. The public who might have an interest to use such a system of analysis are the researchers of malware, the CERTs, ... But for my part, I think we could imagine generalizing a tool like this in many companies, so that end users can post to their IT department (internally) the elements considered suspect (email, pdf, exe, apk, ...).

What is Cuckoo Sandbox?

Cuckoo sandbox allows the automatic analysis of malicious files (Office documents, pdf, executable, ...) for Linux, Windows, MacOS and Android platforms.

The modular design allows, for example:

- to follow the general behavior of a suspicious executable file by tracing the calls of the APIs of this one,
- network traffic analysis,
- static analysis,
- ...

It is an indispensable tool adapted to today's computer world to answer the malware threat. The public who might have an interest to use such a system of analysis are the researchers of malware, the

CERTs, ... But for my part, I think we could imagine generalizing a tool like this in many companies, so that end users can post to their IT department (internally) the elements considered suspect (email, pdf, exe, apk, ...).

Its web interface is very user-friendly and the quality of the reports provides a great help for system and network administrators during an analysis after being infected as part of a survey to understand how the malware works to:

- discover patterns of infection (and look for same patterns on park machines),
- identify the control center of a botnet and block it,
- ...

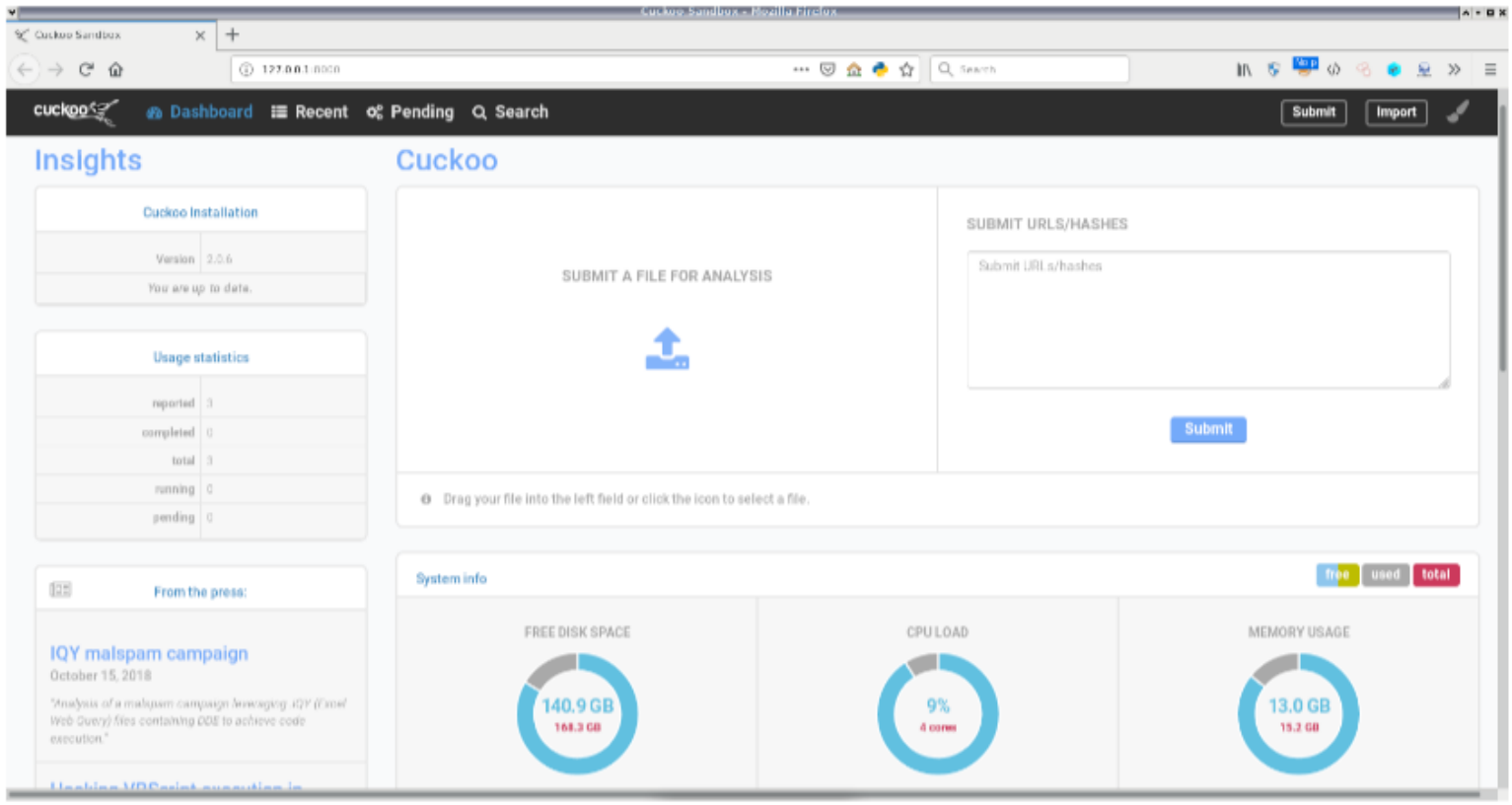


Figure 1: Cuckoo Sandbox Dashboard.

As part of this article, I chose to install Cuckoo Sandbox for nomad investigation use. However, for the installation, you will need to do a little research and have a good knowledge of the system on which it will be installed.

How to install Cuckoo Sandbox

On Github, it is explicitly stated that:

The Cuckoo host components are completely written in Python, therefore it is required to have an appropriate version of Python installed. At this point we only fully support Python 2.7.

So, to write this article, I install Cuckoo Sandbox on my laptop equipped with an Intel i-5 processor and 16 Gb of RAM (and a consequent disk for the storage of guests VMs). My operating system is Arch Linux (with Black Arch repositories installed). For the virtualization environment, I use VirtualBox and I describe the installation of a machine under Microsoft Windows 7. Don't forget, when a command starts with '$', it is executed as user and if the command start with '#', it is executed as root. Basically, we need to make sure that we have the necessary requirements for installation. We check that the system is updated and we install Python 2.7, Oracle VM VirtualBox and Volatility:

```
$ yaourt -Sya --aur && yaourt -Su --aur &&
VISUAL=nano yaourt -Sua --aur
$ yaourt --needed -Sy python2 python2-pip python2-
setuptools python2-virtualenv
$ yaourt --needed -Sy libvirt virtualbox virtualbox-
guest-iso virtualbox-host-dkms
$ yaourt --needed -Sy ssdeep swig tcpdump volatility
mongodb
```

Alternatively, on a more popular operating system like Ubuntu Linux, we can install dependencies this way:

```
$ sudo apt-get update && sudo apt-get upgrade -y &&
sudo apt-get dist-upgrade -y
$ sudo apt-get install python python-pip python-dev
libffi-dev libssl-dev
$ sudo apt-get install python-virtualenv python-
setuptools
$ sudo apt-get install  virtualbox virtualbox-guest-
additions-iso virtualbox-dkms
$ sudo apt-get install libjpeg-dev zlib1g-dev swig
ssdeep tcpdump mongodb volatility

// Trick to keep compatibility with the installation
guide that follows.
$ alias pip2.7=pip2
```

Tcpdump requires root privileges, but we do not want Cuckoo Sandbox to run as root, so we need to set specific permissions on the binary:

```
# setcap cap_net_raw,cap_net_admin=eip /usr/sbin/
tcpdump
```

I chose to use my current user account to start Cuckoo Sandbox. The idea is to be able to launch an analysis on demand since my user account session.

So we can proceed to the installation in the following way:

```
// add your account to ‘ vboxusers’ group.
# usermod -a -G vboxusers USER_ACCOUNT

// Installing Cuckoo Sandbox.
# mkdir /usr/local/share/cuckoo
# chown USER_ACCOUNT:root /usr/local/share/cuckoo
# chmod 750 /usr/local/share/cuckoo
$ cd /usr/local/share/
$ virtualenv cuckoo
$ . cuckoo/bin/activate

$ pip2.7 install -U pip setuptools
$ pip2.7 install -U weasyprint==0.42.2
$ pip2.7 install -U cuckoo

// We need to create the configuration environment.
# mkdir /etc/cuckoo
# chown USER_ACCOUNT:root /etc/cuckoo
# chmod 750 /etc/cuckoo
$ cuckoo --cwd /etc/cuckoo

// We need to add the line below in your .bashrc
file:
export CUCKOO=/etc/cuckoo
```

At this point, our Cuckoo Sandbox is installed on the system but to continue the configuration, we have to create our first guest VM. I chose a guest VM under Microsoft Windows 7 64-bits, I will describe the installation of the subsistence minimum. I will let the readers care to customize their VM by installing software such as Adobe Reader, Adobe Flash Player, Java, Microsoft Office, etc. But don’t forget to

start installed software to get rid of prompts that may interfere with scans (Internet Explorer, Acrobat, ...). We must also disable automatic updates and the firewall. The goal is to have a client virtual machine that is extremely vulnerable to security breaches. Beware, under Microsoft Windows 7, we must think about disabling UAC. So we can create the guest virtual machine with the following commands:

```
$ vboxmanage createvm --name "cuckoo_7" --ostype
Windows7_64 --register
$ vboxmanage modifyvm "cuckoo_7" --memory 1024 --acpi
on --boot1 dvd --nic1 nat

// Be careful you must modify the path of the VDI
file so that it is correctly stored in the directory
of the guest VM.
$ vboxmanage createhd --filename "/path/to/VirtualBox
\ VMs/cuckoo_7/cuckoo_7-hdd1.vdi" --size 18000

$ vboxmanage storagectl "cuckoo_7" --name "SATA
Controller" --add sata --controller IntelAHCI
$ vboxmanage storageattach "cuckoo_7" --storagectl
"SATA Controller" --port 0 --device 0 --type hdd --
medium "/path/to/VirtualBox\ VMs/cuckoo_7/cuckoo_7-
hdd1.vdi"

// We are creating a CD-ROM drive that will be used
for the installation of Microsoft Windows 7.
$ vboxmanage storageattach "test_7" --storagectl
"SATA Controller" --port 1 --device 0 --type dvddrive
--medium /home/mekhalleh/Utils/images.iso/
sw_dvd5_win_pro_n_7w_sp1_64bit_english_-2_mlf_x17-596
41.iso

// Creating the Network Interface (By default, the
guest VM is on an isolated LAN that does not have
access to the Internet)
$ vboxmanage hostonlyif create
$ vboxmanage modifyvm "cuckoo_7" --nic1 hostonly
$ vboxmanage modifyvm "cuckoo_7" --hostonlyadapter1
vboxnet0

// Creating a share between the host and our guest
VM. Inside we deposit our Cuckoo Sandbox agent for
Microsoft Windows.
$ vboxmanage sharedfolder add "cuckoo_7" --name "7"
--hostpath /path/to/VirtualBox\ VMs/shared/7 --
automount
$ cp /etc/cuckoo/agent/agent.py /path/to/VirtualBox\
Vms/shared/7
```

The following are essential to the operation of the agent. We need to install Python 2.7.13 and the PIL1.1.7-win32-py2.7 module on the guest machine. For this, we can download the elements below and install them on the guest VM.

- https://www.python.org/downloads/release/python-2713/
- http://effbot.org/downloads/PIL-1.1.7.win32-py2.7.exe

In VirtualBox we need to get the following configuration (you must disable DHCP server support):

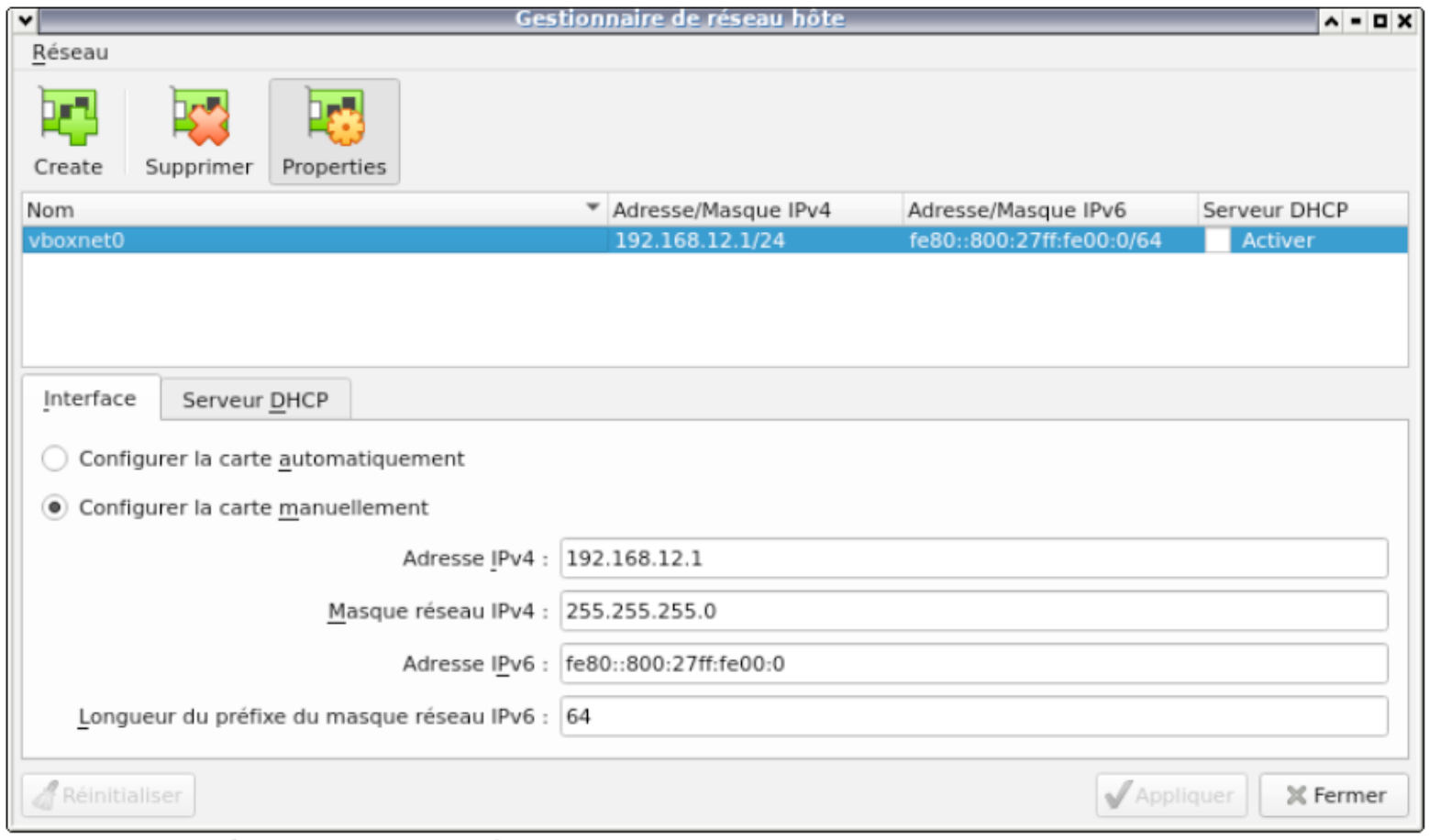


Figure 2: VirtualBox Host Network Manager.

In our guest VM we configure the network interface this way.

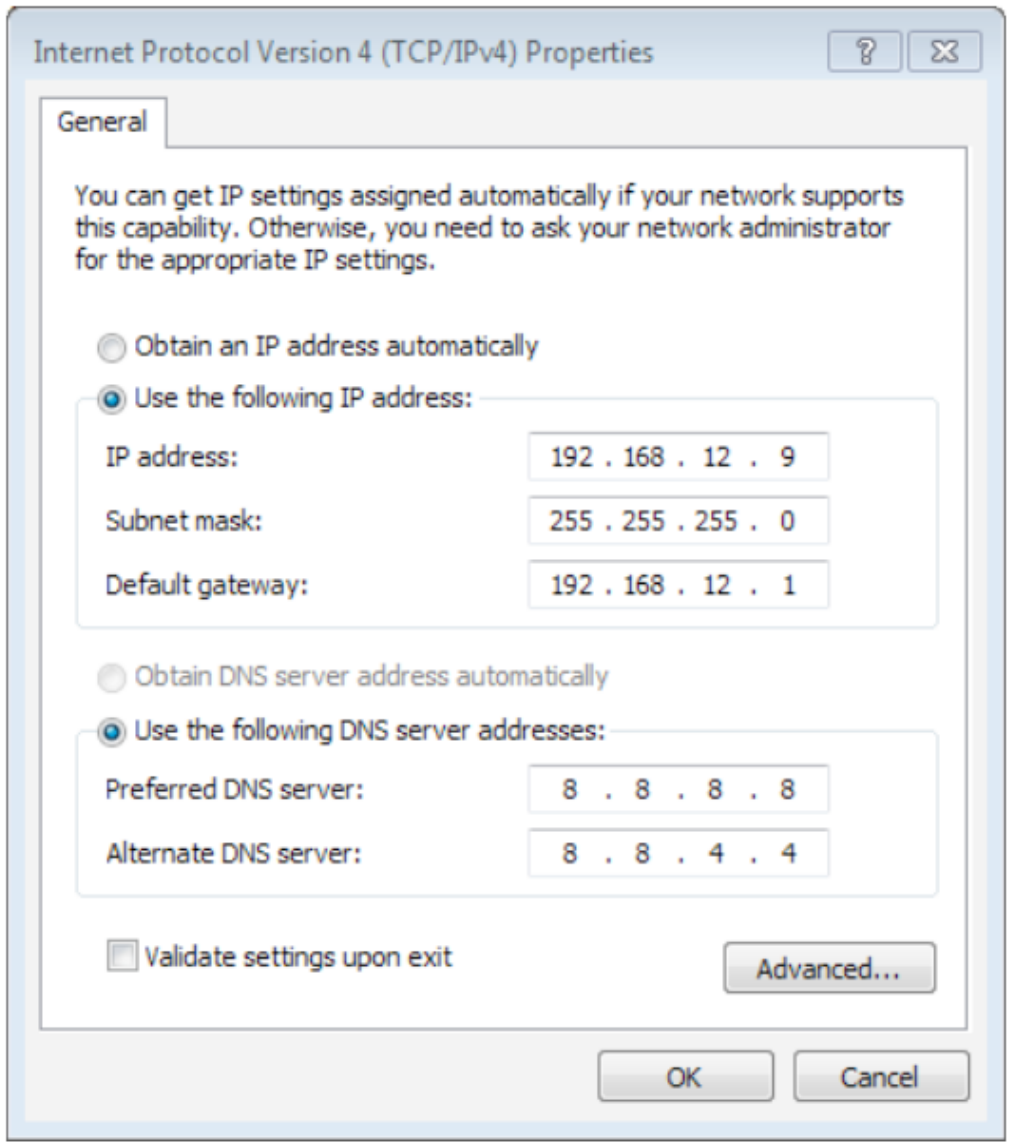


Figure 3: IP Configuration on the Microsoft Windows 7 guest VM.

Then we will make sure that our guest VM accesses the Internet, with IPTables we will configure IP Forwarding.

```
# iptables -A FORWARD -o enp8s0 -i vboxnet0 -s
192.168.12.0/24 -m conntrack --ctstate NEW -j ACCEPT
# iptables -A FORWARD -m conntrack --ctstate
ESTABLISHED,RELATED -j ACCEPT
# iptables -A POSTROUTING -t nat -j MASQUERADE
# sysctl -w net.ipv4.ip_forward=1
```

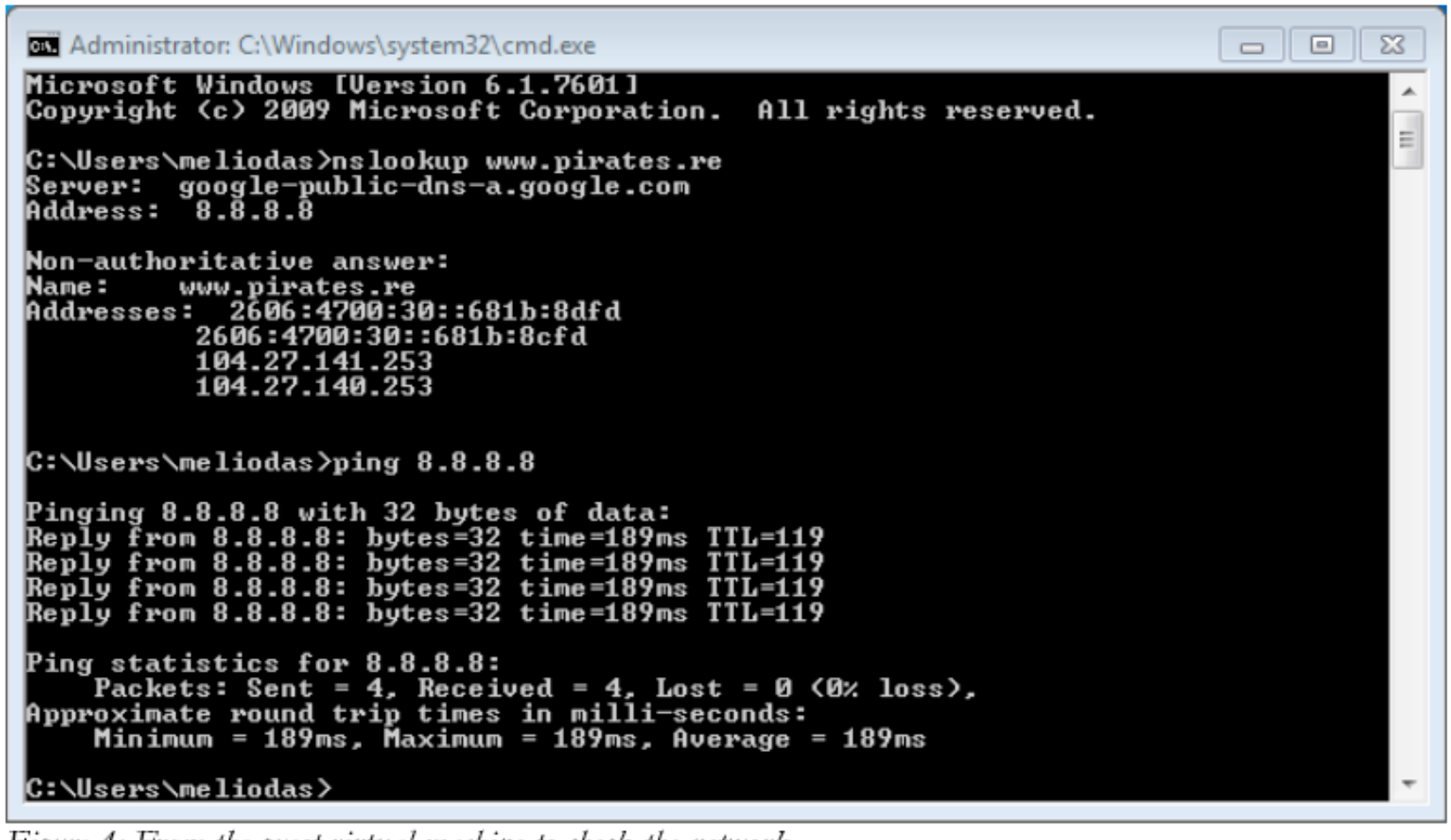


Figure 4: From the guest virtual machine to check the network.

Let's make sure to copy the agent to the "%APPDATA%\Roaming\Microsoft\Windows\Start Menu \Programs\Startup\agent.py" directory. To access the network share between the host and the guest VM, you must first install the guest addition tools of VirtualBox. Once the file is copied, we can reboot our guest VM and we ensure the proper functioning of the agent at startup.

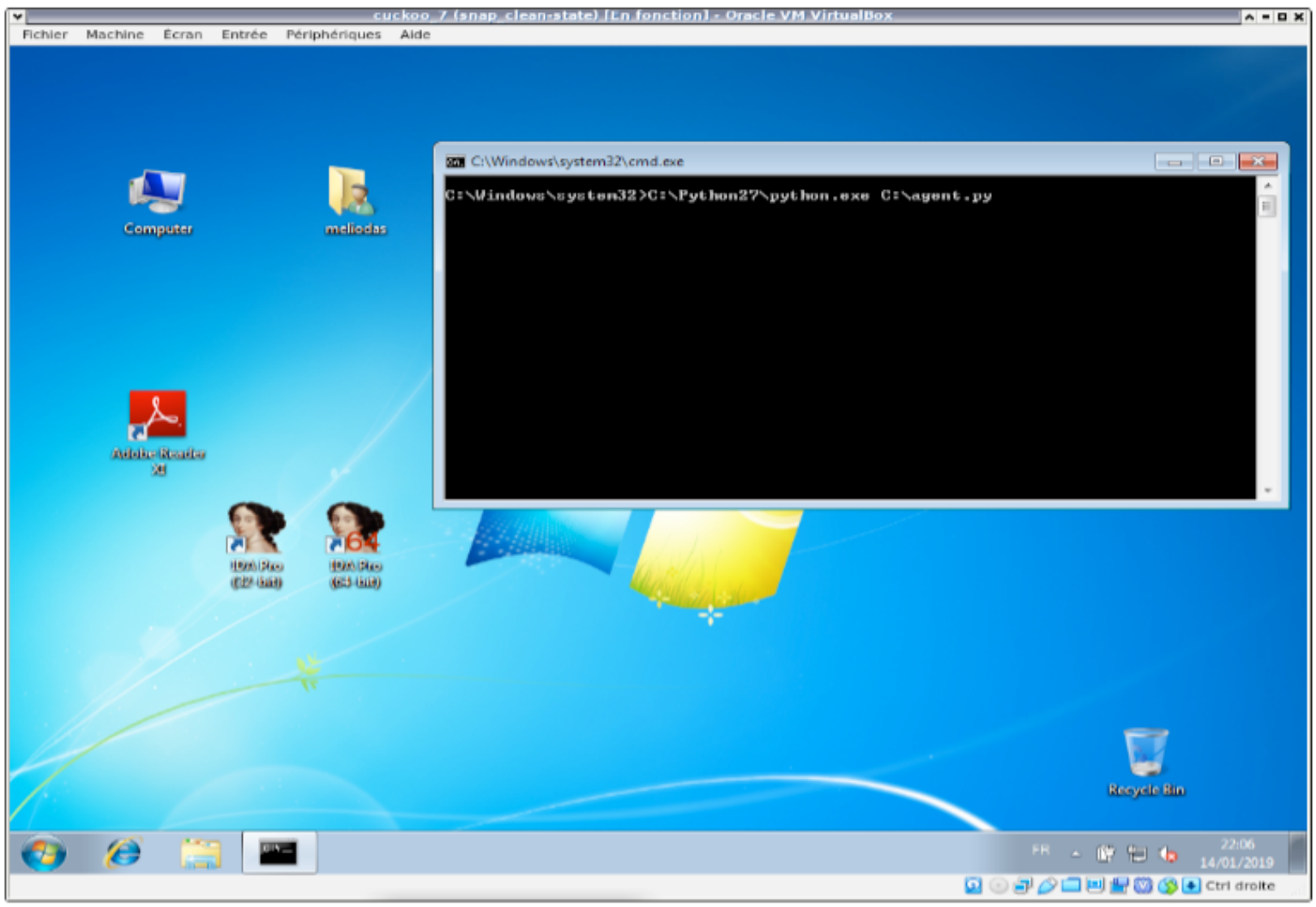


Figure 5: From the guest virtual machine to check the Cuckoo Sandbox agent.

Then, you can execute the following commands to freeze your guest VM:

```
$ vboxmanage snapshot "cuckoo_7" take "snap_clean-
state" --pause
$ vboxmanage controlvm "cuckoo_7" poweroff
$ vboxmanage snapshot "cuckoo_7" restorecurrent
```

Personally, I chose to install the guest VM in the directory/var/cuckoo:

```
# mkdir /var/cuckoo
# chmod  USER_ACCOUNT:root /var/cuckoo
# chmod 750 /var/cuckoo
```

And use VirtualBox to move it properly ...

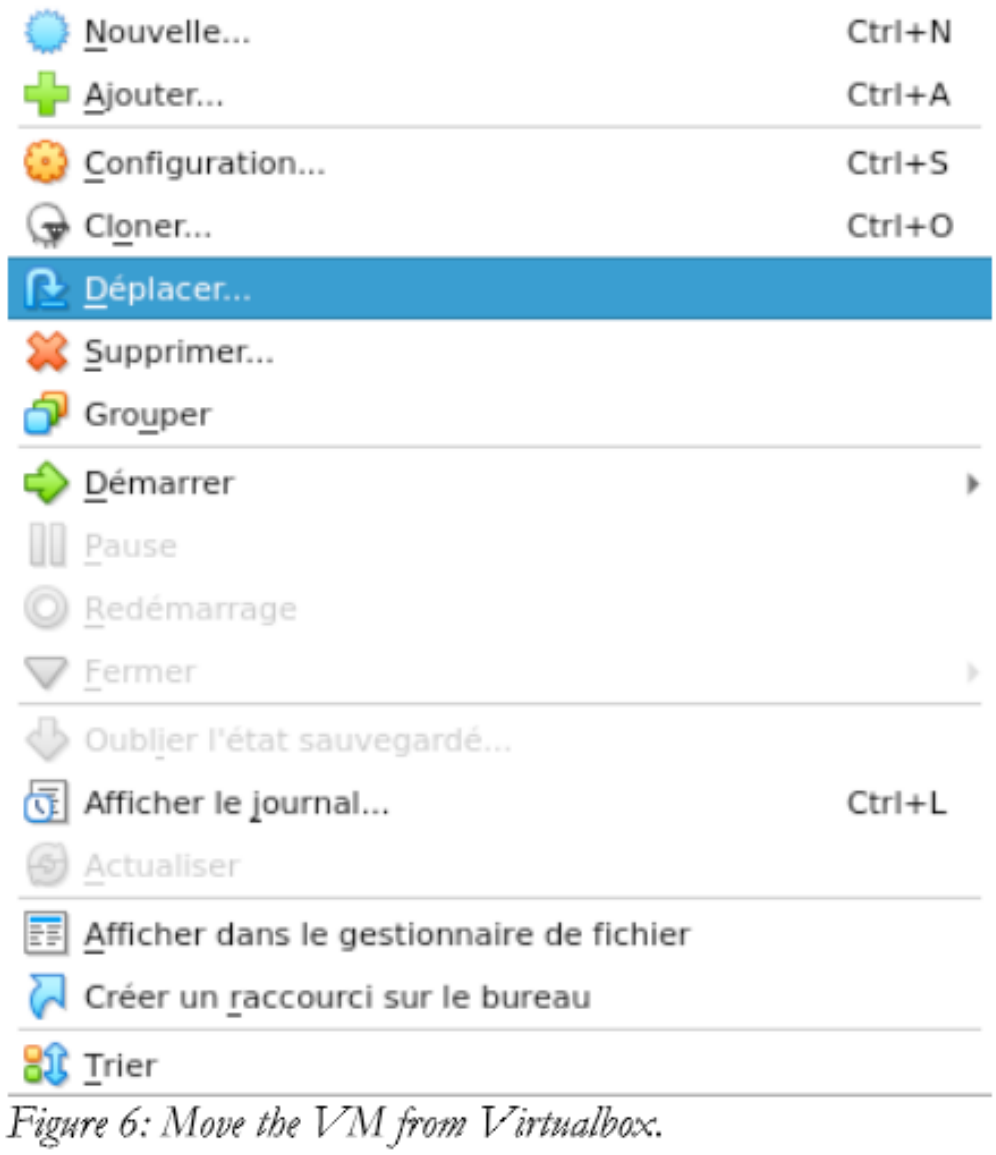


Figure 6: Move the VM from Virtualbox.

From now on, we must proceed to a minimum of configuration for the functioning of Cuckoo Sandbox:

In the configuration file "cuckoo.conf", we can configure the general options of Cuckoo Sandbox (used virtualization system, IP address, storage options, remote control in the Web interface with the Guacamole service, ...).

<cuckoo.conf>

```
[cuckoo]
machinery = virtualbox
memory_dump = yes

[resultserver]
ip = 192.168.12.1
```

In the "memory.conf" configuration file, we need to check the profile used by Volatility to work properly. We can get the list of Volatility profiles with the command '`volatility --info | grep Profiles -A 48`'

<auxiliary.conf>

```
[sniffer]
enabled = yes
tcpdump = /usr/sbin/tcpdump
```

We must declare that our guest virtual machine is used by Cuckoo Sandbox, as we have set up an Oracle VirtualBox environment, we will modify the 'virtualbox.conf' file.

<memory.conf>

```
[basic]
guest_profile = Win7SP1x64
```

However, if the reader chooses to work on another environment, he will have to adapt the configuration by modifying the appropriate file.

<virtualbox.conf>

```
[virtualbox]
mode = headless
path = /usr/bin/VBoxManage
interface = vboxnet0
machines = cuckoo_7
controlports = 5000-5050

[cuckoo_7]
label = cuckoo_7
platform = windows
ip = 192.168.12.9
snapshot = snap_clean-state
interface = vboxnet0
resultserver_ip =
resultserver_port =
tags =
options =
osprofile = Win7SP1x64
```

However, if the reader chooses to work on another environment, he will have to adapt the configuration by modifying the appropriate file.

avd.conf	Specify whether we're running the Android emulator.
esx.conf	Used for VMware ESX environment.
kvm.conf	Used for KVM environment.
physical.conf	Used in case of remote physical sandboxing.
qemu.conf	Used for Qemu environment.
virtualbox.conf	Used for Oracle VirtualBox environment.
vmware.conf	Used for VMware environment.
vsphere.conf	Used for VMware Vsphere environment.
xenserver.conf	Used for Xen environment.

<processing.conf>

```
[virustotal]
enabled = yes
scan = yes
key = YOUR_API_KEY
```

In the file "reporting.conf" we will realize the minimum configuration to access the Web interface of Cuckoo Sandbox. We activate HTML reports (they will be stored in the '/etc/cuckoo/storage/analyses/' directory).

<reporting.conf>

```
[singlefile]
enabled = yes
html = yes

[mongodb]
enabled = yes
```

In the configuration file "routing.conf", we can modify the various outputs on the Internet (VPN, ToR, ...). This step requires a little extra configuration depending on your needs. Before we can start an analysis with our sandbox environment, we need to update the signatures, modules, ...

For this we need to use the following commands (remember that the mongodb service must be started):

```
$ . /usr/local/share/cuckoo/bin/activate
$ cuckoo community
```

We can now boot the Cuckoo Sandbox system (we can use the '-d' switch to start as daemon too):

```
$ . /usr/local/share/cuckoo/bin/activate
$ cuckoo
```

In another terminal we start the Cuckoo web service:

```
$ . /usr/local/share/cuckoo/bin/activate
$ cuckoo web
```

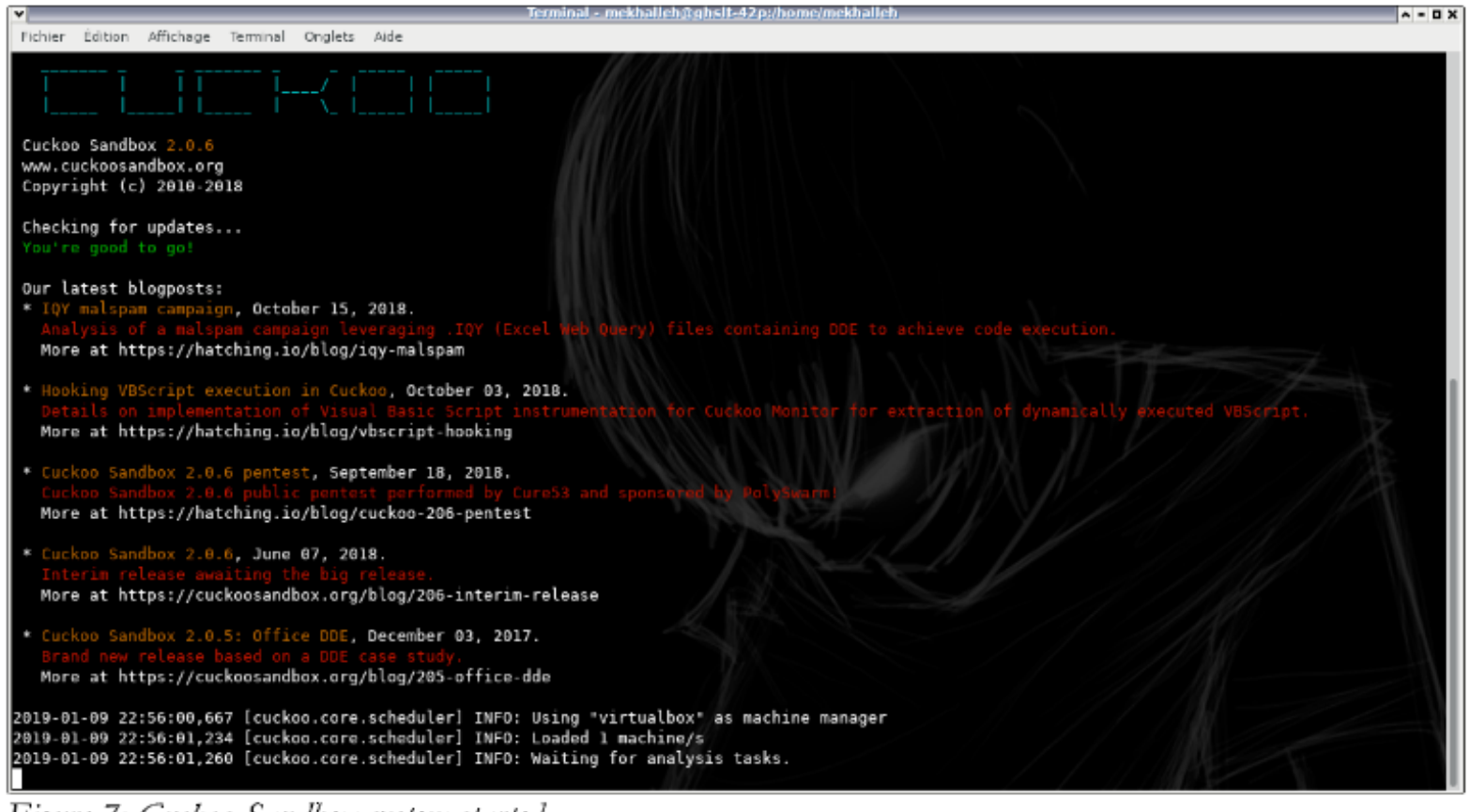


Figure 7: Cuckoo Sandbox system started.

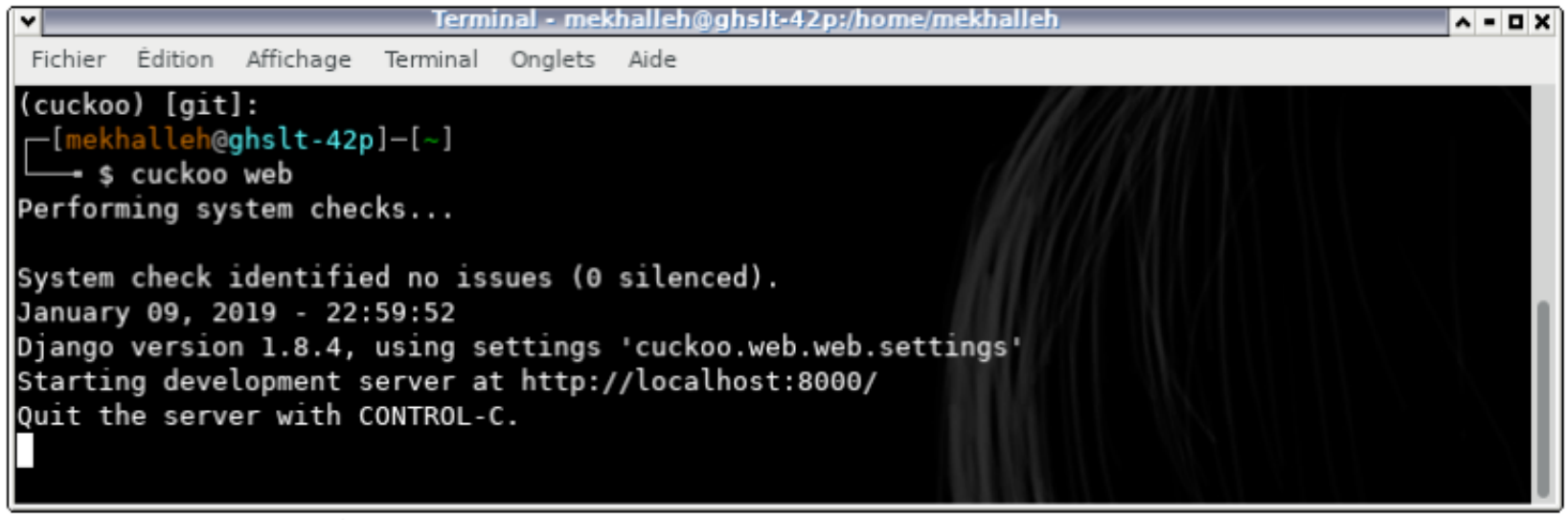


Figure 8: Cuckoo Sandbox Web Interface started.

Do not forget that after rebooting the system, we lose part of the configuration (IPTables, VirtualBox interface, ...). So after a reboot we have to restore this configuration, with the following commands:

```
$ vboxmanage hostonlyif ipconfig vboxnet0 --ip
192.168.12.1
# iptables -A FORWARD -o enp8s0 -i vboxnet0 -s
192.168.12.0/24 -m conntrack --ctstate NEW -j ACCEPT
# iptables -A FORWARD -m conntrack --ctstate
ESTABLISHED,RELATED -j ACCEPT
# iptables -A POSTROUTING -t nat -j MASQUERADE
# sysctl -w net.ipv4.ip_forward=1

# systemctl start mongodb
```

Submit suspicious file, hash or URL for analysis

After installation, access to the web interface of our Cuckoo Sandbox from the URL http://127.0.0.1:8000/. We arrive on the dashboard, which is the main page, from this page, we notice some information about the server (disk space, memory, CPU, Cuckoo Sandbox version, ...). From this page, the analyst can follow the links in the top menu to access the recent analysis, view the pending analysis or simply access a search engine. We can use the form on this page to submit suspicious items to our Sandbox.

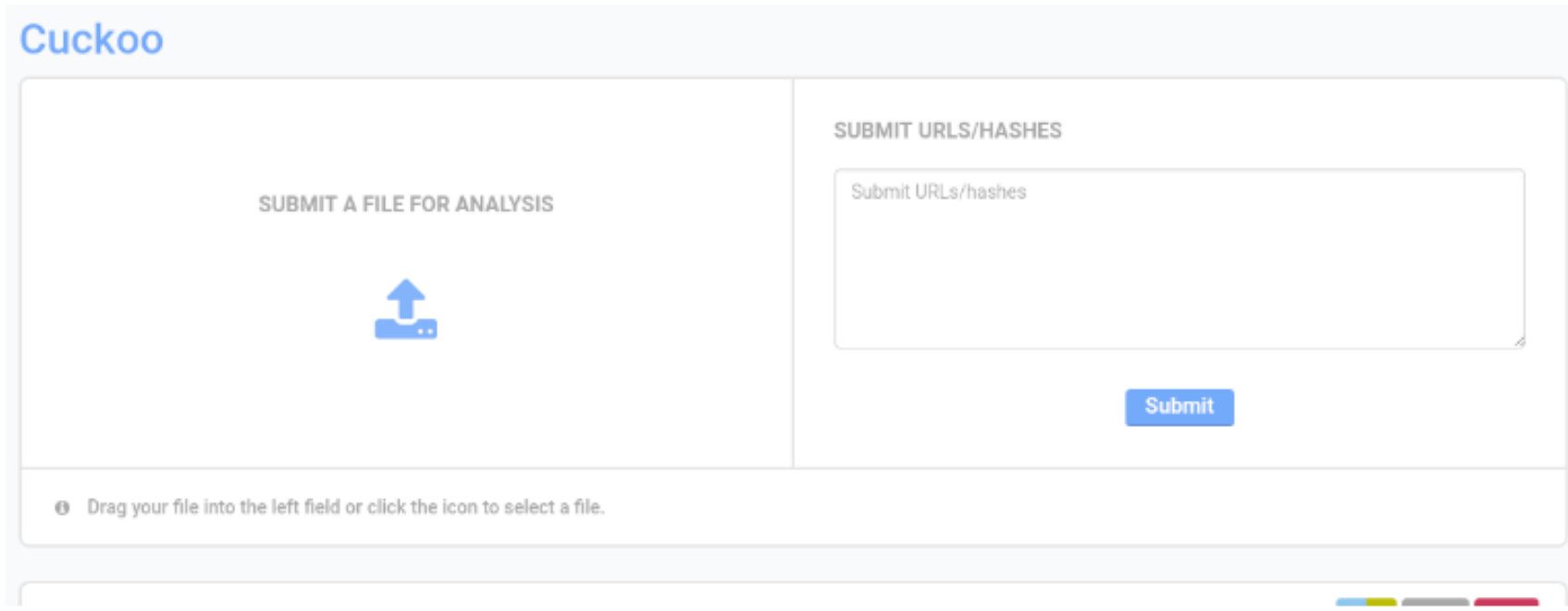


Figure 9: Submission form from the Cuckoo Sandbox dashboard.

For test purposes, I have downloaded several samples from the Hybrid-Analysis website to obtain various analysis reports ...

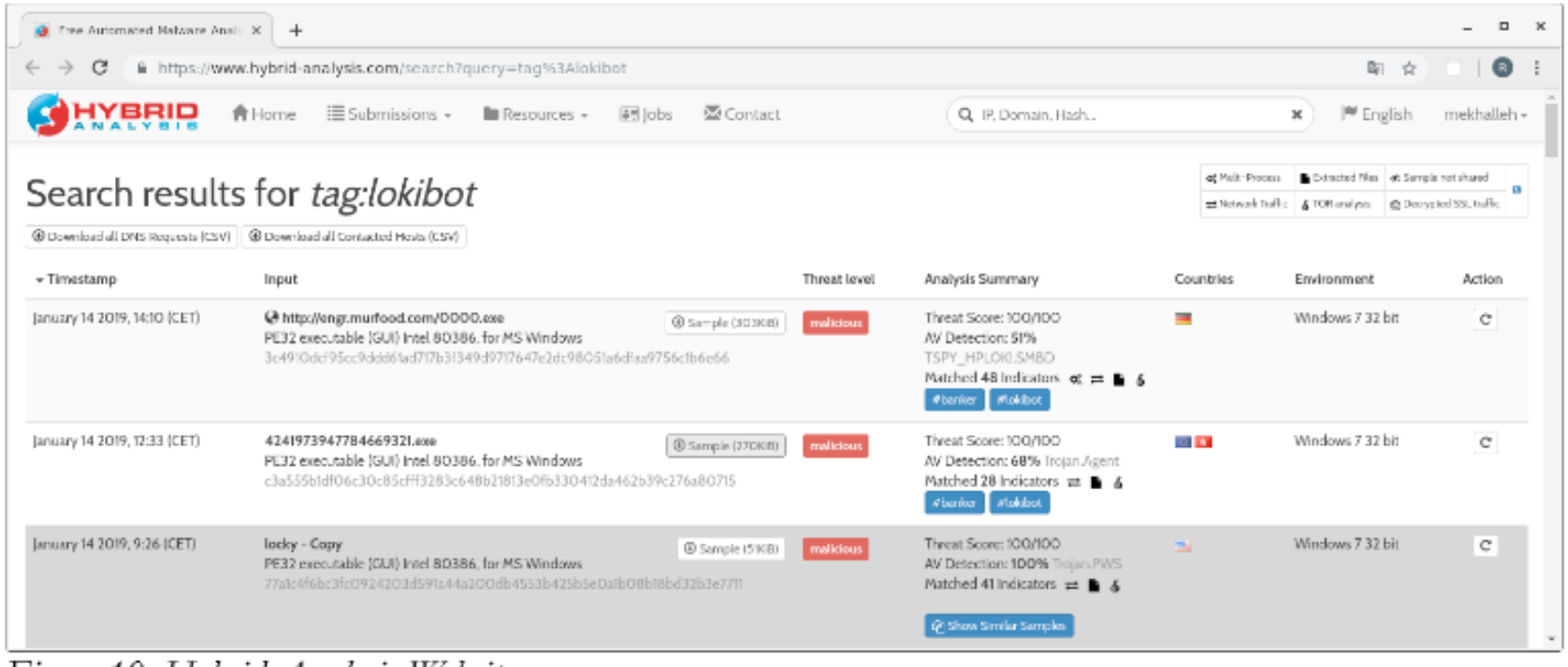


Figure 10: Hybrid Analysis Website.

Then we can post our suspicious files, incidentally, modify the options and start the analysis. Analysis options are available in the navigation bar on the left.

During a submission, we have access to options to refine our analysis by:

- Defining your preferred Internet output (VPN, ToR, network simulator, ...), we must first configure the routing via the configuration file "routing.conf";
- Describing how the Cuckoo analysis component should perform the task using the Package option we can specify/force the analysis of a specific file (exe, xls, dox, ...). Noted that it is possible to define the priority of this task and ... the timeout;
- Choosing to activate the remote control (if we previously realized the configuration of the Guacamole service) it is possible for the analyst to interact with the Sandbox (on the other hand, in this case, we must not forget to disable the simulation of human interaction not to be embarrassed by it);

- Enabling/Disabling on-demand injection, process dump in memory or full dump of guest VM memory for use with Volatility;
- Defining new options we can, for example, add arguments to the file we want to analyze if it needs this or the password corresponding to a zip file to submit;
- Allowing us to choose the guest VM we want to use when analyzing,
- ...

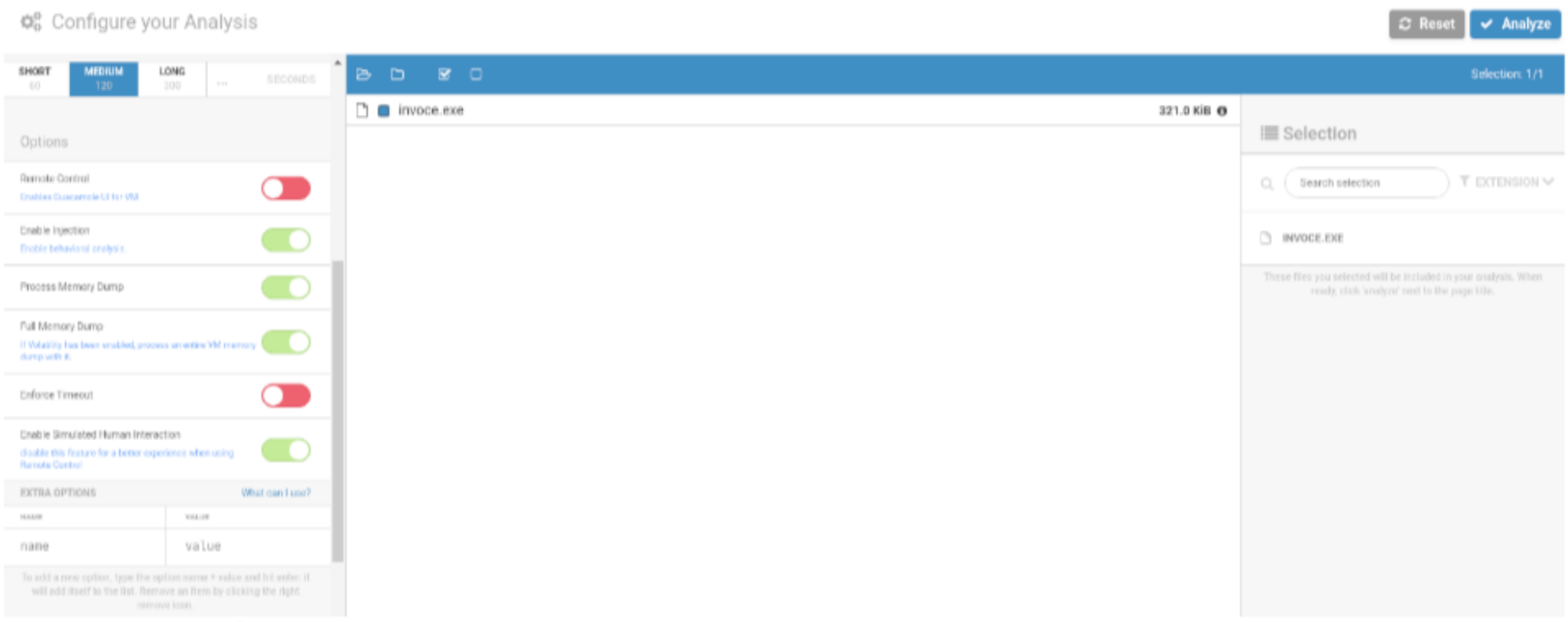


Figure 11: Analysis Configuration page.

I then submitted several malware to generate different analysis reports, botnet known as ZeuS, Ice9 but also Cryakl, Dridex, Pony and others ... and also malicious URLs that I generated with Metasploit Framework.

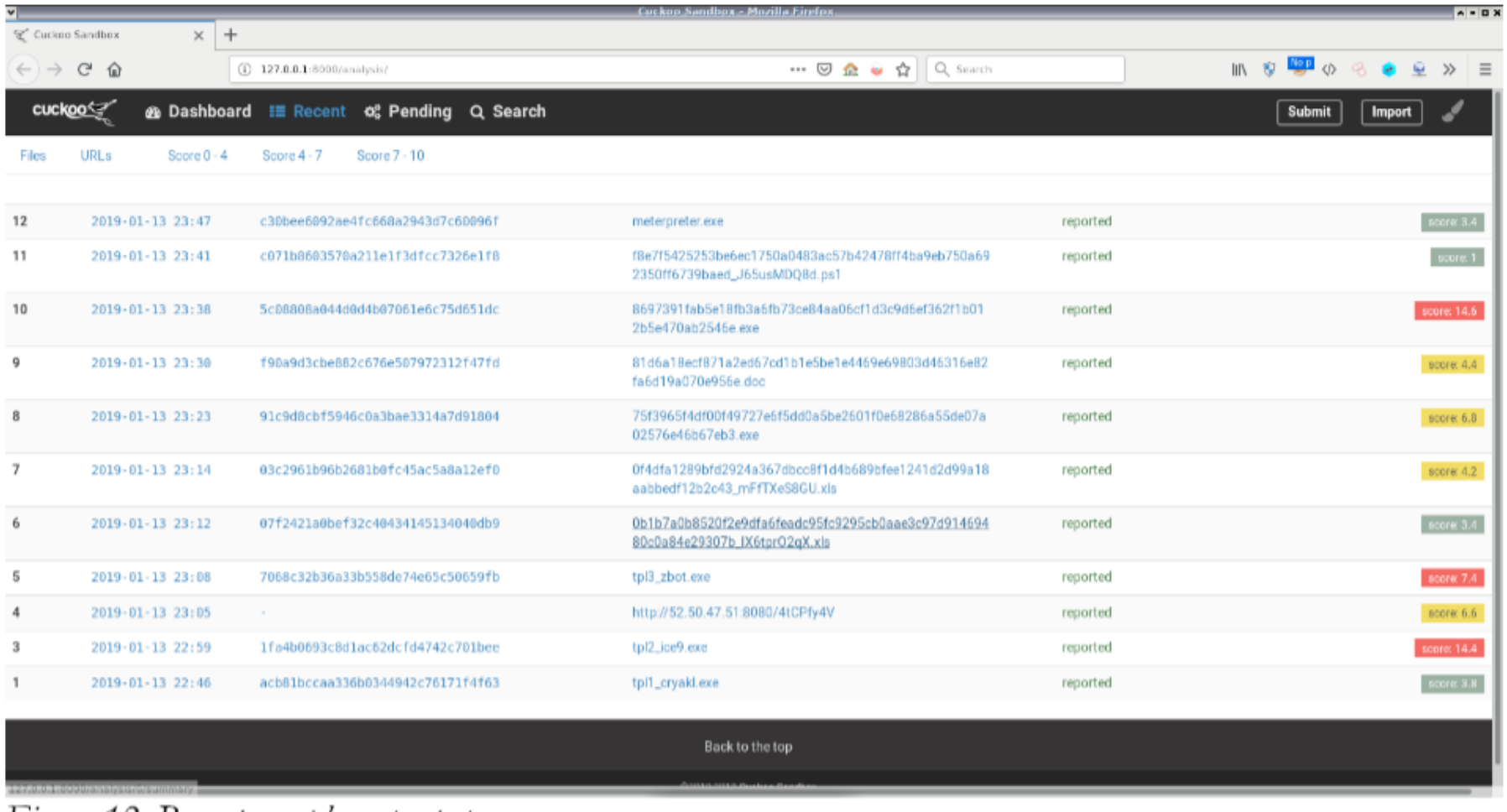


Figure 12: Recent samples reports page.

There are other methods to submit, we can do it on the command line:

```
cuckoo submit --machine cuckoo_7 /path/to/file
cuckoo submit --machine cuckoo_7 --url http://
www.example.com
cuckoo submit --package exe --options arguments=--
dosomething /path/to/file
```

It should also be noted that Cuckoo Sandbox can easily integrate with other security solutions, for this reason it is strongly recommended to use the REST API interface. However, nothing prevents us from submitting an element via a Python script.

There is no excuse for not integrating this tool into an internal files check workflow.

Analysis Report

When the analyst selects a report, it is returned to the summary page (its default page). We can observe the menu on the left that allows us to navigate the report.

By viewing the summary, at first glance, the analyst is able to identify if the file is potentially dangerous (here, the summary of information about zbot).

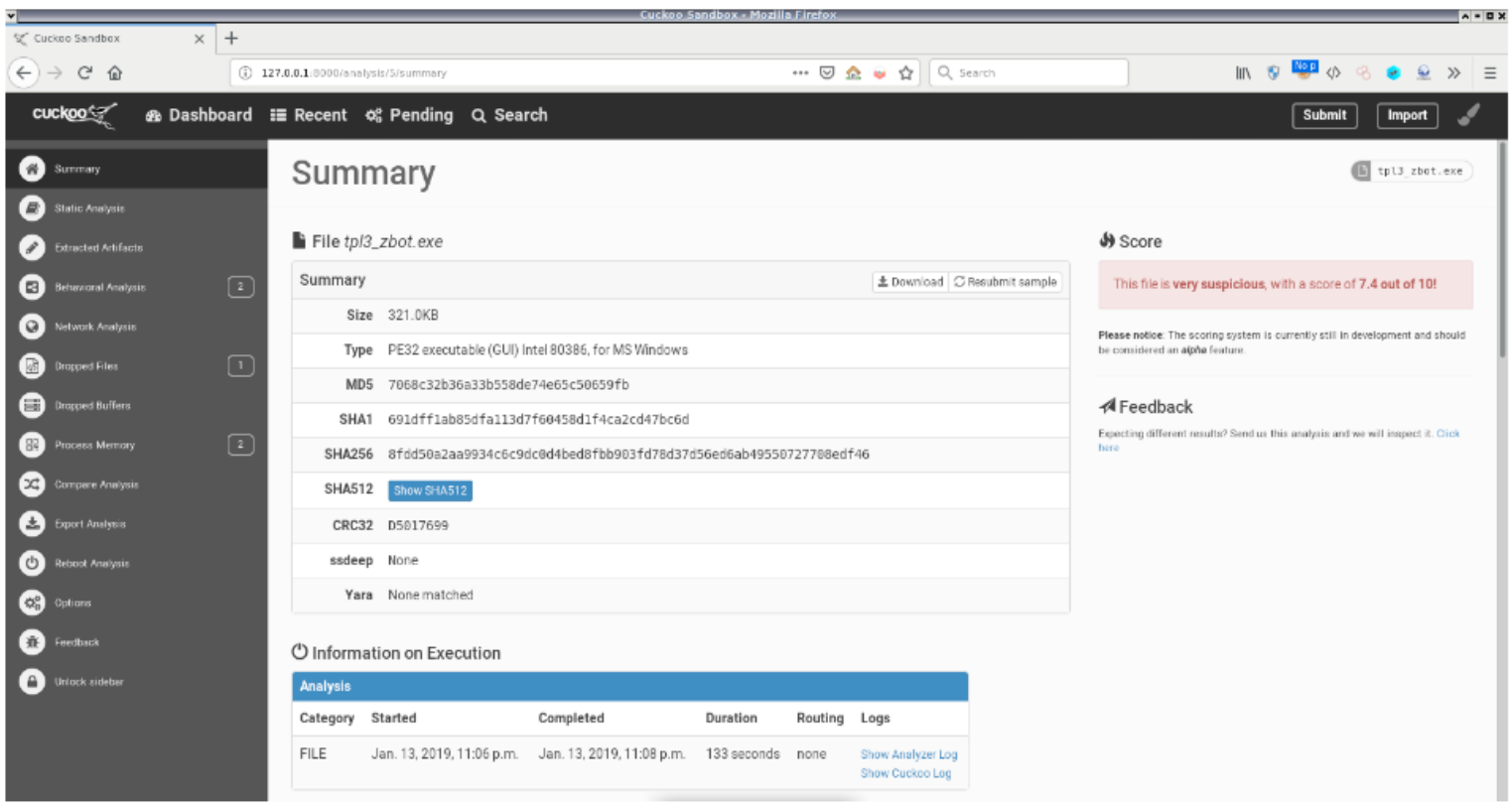


Figure 13: Summary analysis report, here an zbot malware.

The scoring is a good indicator to determine if the item is potentially malicious, but do not fall into the trap, the malware developers adapt, they make sure to distort the analysis using anti-sandboxing techniques. In this same page, the signatures detected during the analysis can be very useful to determine the threats.

Figure 14: Analysis summary of another stealer tool.

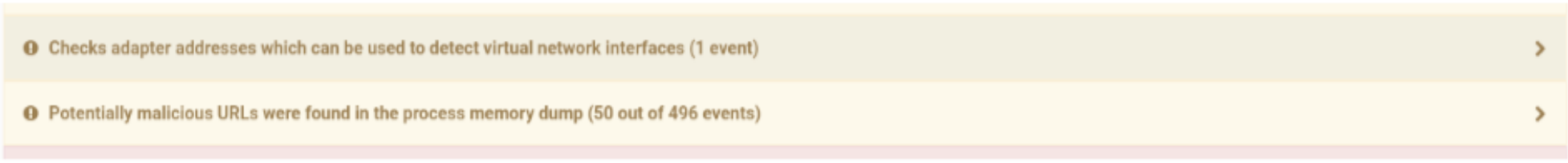


Figure 15: Analysis summary of another tool.

Analysts will be happy to quickly get the information to investigate, for example, the address of a C&C or a relay (or sometimes both).

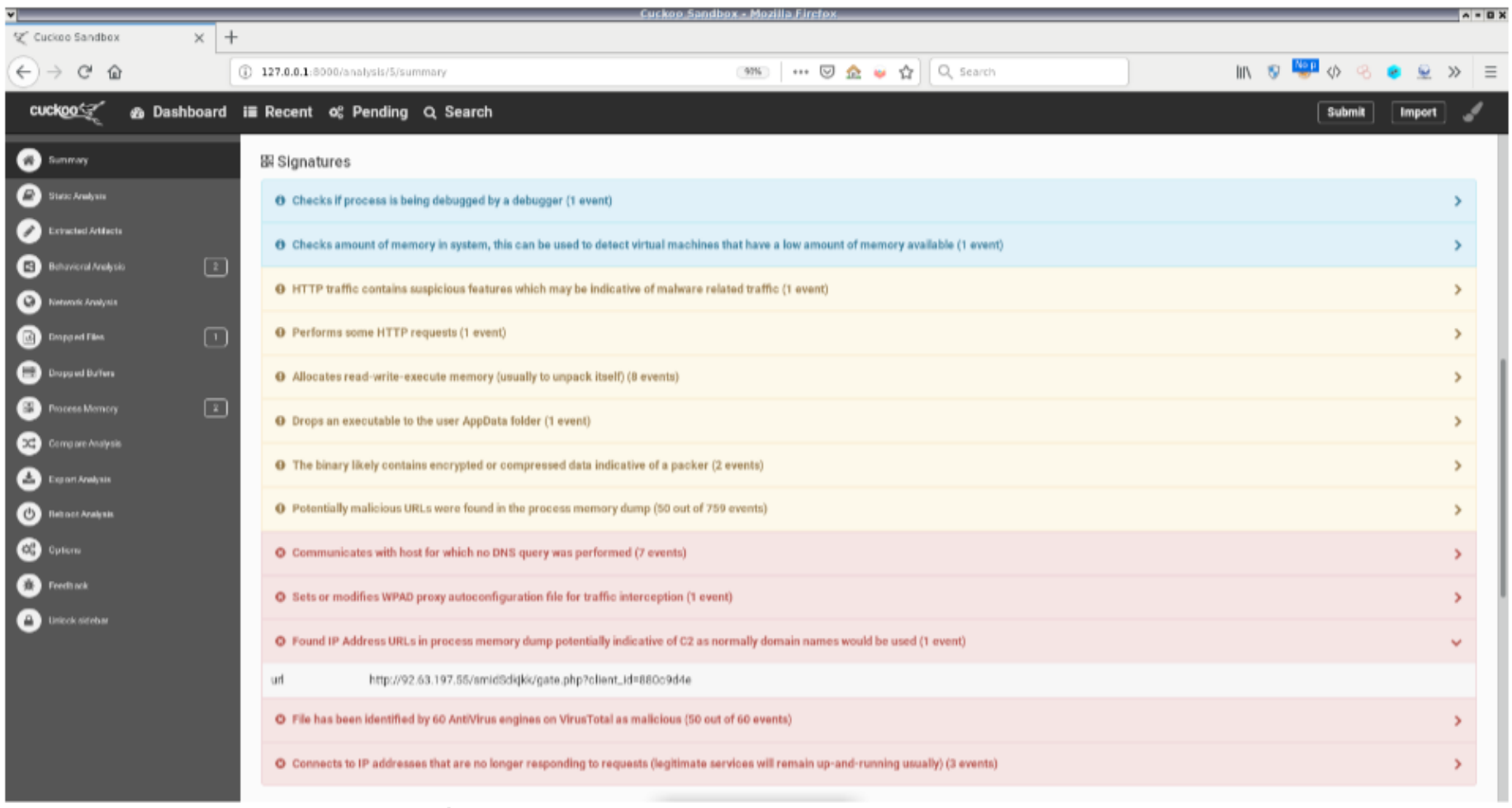


Figure 16: Locate the zbot relay/control center.

If we navigate Network Analysis, we see what appears to be a connection to a control center.

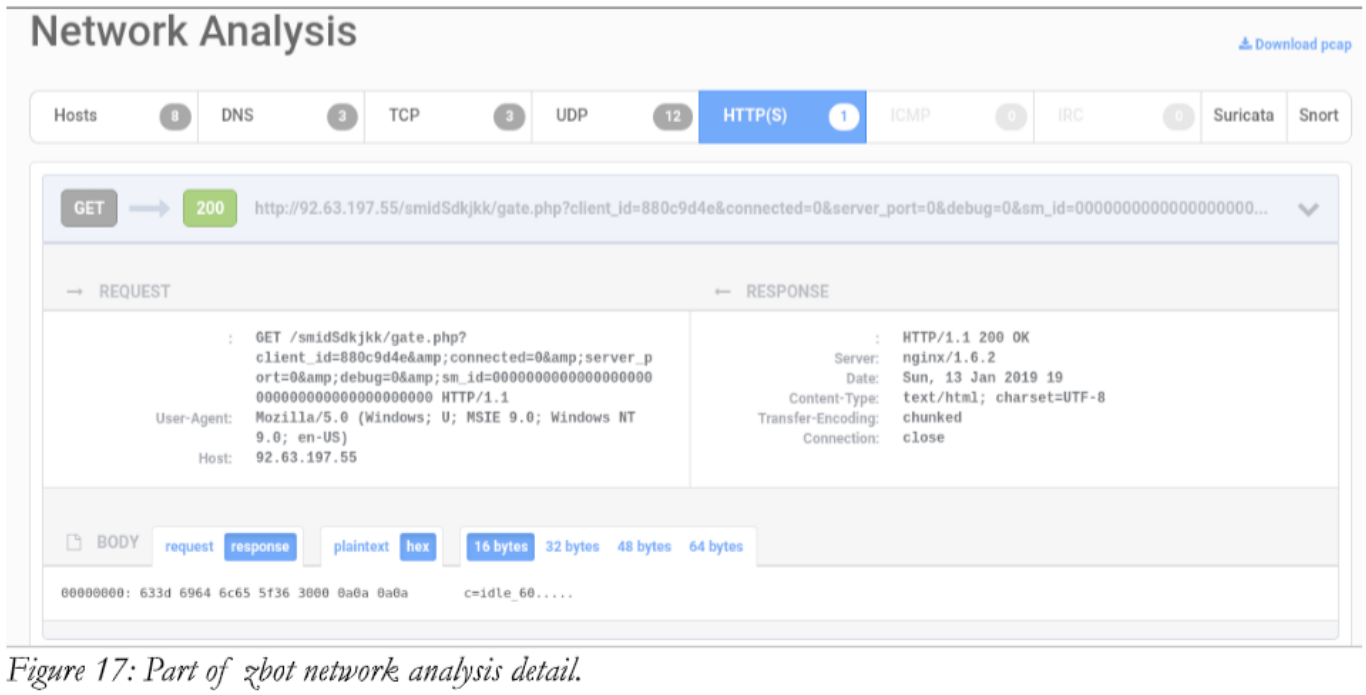


Figure 17: Part of zbot network analysis detail.

If we take a closer look, the first IP address is located in Russia, the others are located in Germany. We can see what looks like the default page of an Apache service on a CentOS server.

Figure 18: Visit the page (gate.php) of zbot.

Figure 19: Another visit to the same page (a later time) highlights a different IP address.

Figure 20: Follow the IP address found in the gate.php page (when she answers).

Note that if we look at the network analysis of a meterpreter payload (without stream encryption) we will observe the download of stage 2 (on Windows system, the reflective DLL) on our Sandbox Windows 7 Guest VM ...

The analysis reports are extraordinarily useful, as we can observe in the different tabs of the menu, for the most interesting ones:

- In static analysis, we will have access to the ASCII character string, to the virustotal results (the sample can be submitted automatically);
- In extracted artefact, we will find the artifacts related to our analysis (scripts, commands, ...).

Figure 21: Extracted artifacts page from analyse report, here a maldoc who execute a Powershell command to download a Payload locally.

- In behavioral analysis, we will observe the behavior of processes (network, registry, interaction with other processes, …);
- In Network Analysis we have access to the network information, we will observe the requests that are emitted from the sample (DNS, TCP, UDP, HTTP, HTTPS);
- The dropped file or buffer includes information on the files dropped by the sample and dumped by sandboxing process;
- In the memory processing page, we will have the information from Volatility;

Figure 22: Process memory page extracted from the analysis report, here the detection of a potential executable file (URL found in memory).

The analyst can also compare the execution graph of two threats.

Apart from that, the proofs (snapshot, memory dump, reports, ...) are available in the directory '/etc/cuckoo/storage/analyzes' to allow us a further manual study.

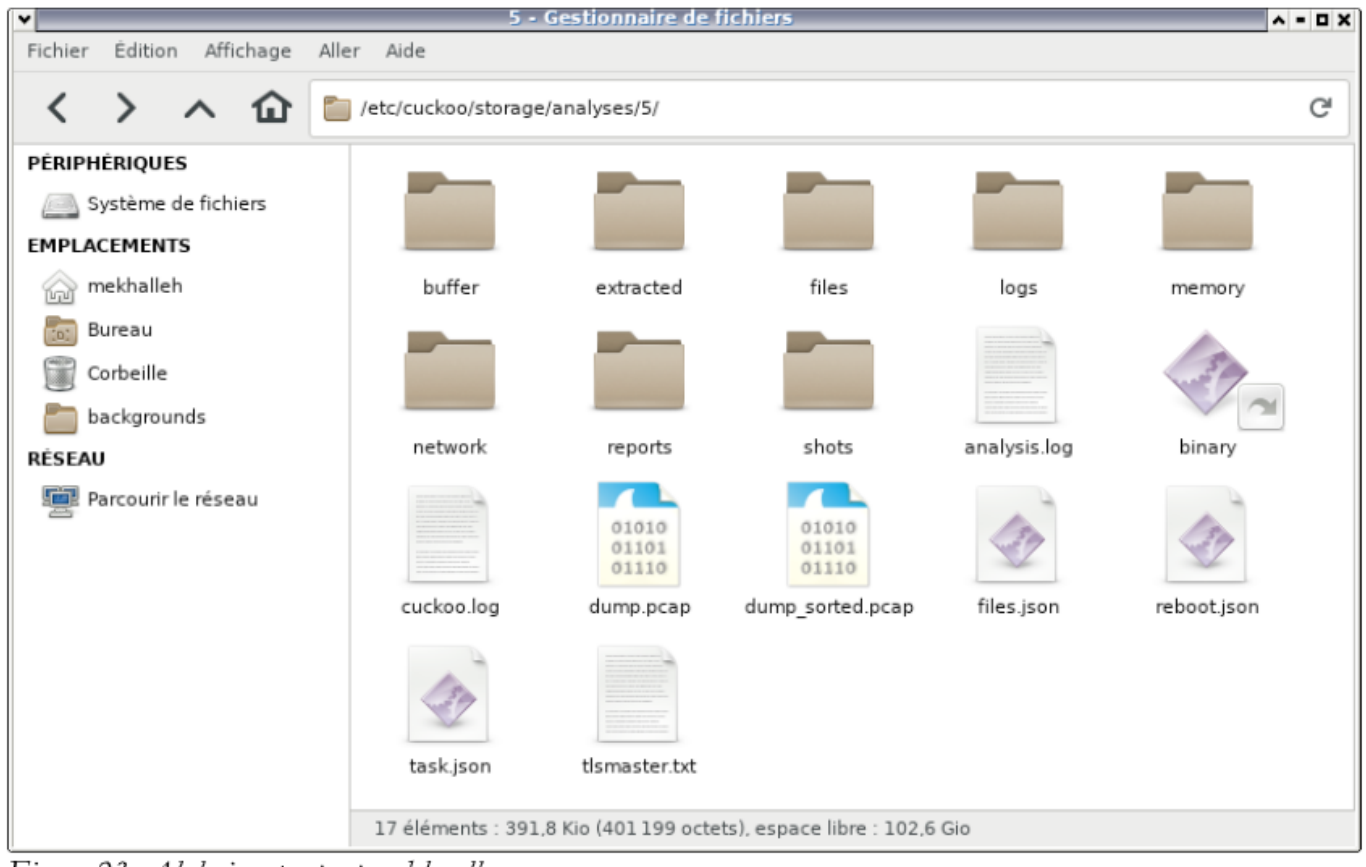


Figure 23: Alalysis reports stored locally.

We can clean the analysis with the comma.

```
$ cuckoo clean
```

Conclusions

Cuckoo sandbox allows us to do a dynamic analysis, a rather serious job, allowing the analyst to have a quick overview of the behavior of a program. It is not uncommon for a system administrator to be solicited to "analyze" a file, sometimes, it is the end user who has a doubt about a file or more often, during an incident response. Cuckoo Sandbox allows the analysis to respond faster than if it had to do the job manually.

However, fully automated tools typically do not provide as much information as a manual scan, leaving the choice to the analyst to continue his investigation:

- Statics property analysis,
- Interactive behavior,
- Manual code reversing.

The malware designer continually works on sandbox bypass, for example, by basically detecting whether the process is running in a virtual environment. Thus, nothing will replace human analysis.

To learn more about this topic, please refer to the Cuckoo Sandbox online documentation:

- https://cuckoosandbox.org/
- https://cuckoo.readthedocs.io/en/latest/

Malware hunters can gain privileged access to obtain samples:

- https://avcaesar.malware.lu/
- https://www.hybrid-analysis.com/
- ...

It is better to install your own sandboxing system, however, it is possible to test it online:

- https://malwr.com/ (broken for a while but says he'll be back soon)
- https://cuckoo.cert.ee/dashboard/

About the Author

I had my first computer very young, like most enthusiasts, I was immediately a fan of technologies that play an important role in our daily lives (computers, Internet, phones, ...). I have a degree in computer science. Infrastructure and network project manager, integrator, pentester ... today based at Reunion Island. I have always been sensitive to security. I write from time to time on the blog Pirates.RE (https://www.pirates.re/). Considering that knowledge of the different attacks used by cybercriminals are fundamental concepts to protect our information systems. I have been doing research and watching the subject of IT security for at least 15 years. Even today, I am interested in several fields (Reverse Engineering, Programming, Electronics, SDR, ...).

Inside Cuckoo Sandbox: A view into Cuckoo's internals and emulation flow

by Mark Lechtik

To better understand how one can adapt Cuckoo and maximize the profit from it, it is essential to know its nuts and bolts, exactly what we intend to do in this article. We will discuss the general architecture, which should give a clear idea of the main components that comprise Cuckoo and how they interact with one other, all with a "guided tour" through all the phases that happen from the moment a sample is submitted for analysis until results are reported back to the user. We will also try to see what kind of shortcomings the current architecture and implementations present and what can be done to overcome them. So without further ado, let's dig in.

Malware analysis is a complicated task. Given a potential malware sample, an analyst would have to go through quite a few phases to understand what it does – starting from overall observation on whether it is packed or obfuscated, through actual extraction of intrinsic payload to final triage or analysis of the results. Doing all of this manually is a lot of work that proves to be tedious and time consuming.

In particular, some of the tasks required to understand a behavior of a file in question would often involve execution of the sample. For instance, understanding how a malicious PE invokes its final payload can be done through monitored execution, i.e. the analyst would have to spin up a virtual machine and run the sample with an instrumentation of some sort (e.g. using Process\API Monitor in the background), after which he\she would have to go through the resulting logs and infer what

happened – did the sample drop another binary and execute it? Did it inject payload to another process? and so on.

Fortunately, all of the steps required to get such an insight into a behavioral flow can be automated through sandboxing solutions like Cuckoo. These would eventually run a submitted file, intercept system calls that its underlying processes invoke and provide a summary report outlining various behavioral traits like the execution's process tree, dropped artifacts, network communication and particular API calls conducted by each process.

In this sense, Cuckoo is not the only available solution but it certainly is one of the popular ones. There are a few reasons for that - for one, it is highly maintained by its developer community and the level of its documentation allows fast deployment. Moreover, the fact it is open source provides good extensibility and allows users to adapt it for their particular needs. Finally, it provides quite nice results, even in comparison with some of its commercial counterparts, and it does so free of charge.

To better understand how one can adapt Cuckoo and maximize the profit from it, it is essential to know its nuts and bolts, exactly what we intend to do in this article. We will discuss the general architecture, which should give a clear idea of the main components that comprise Cuckoo and how they interact with one other, all with a "guided tour" through all the phases that happen from the moment a sample is submitted for analysis until results are reported back to the user. We will also try to see what kind of shortcomings the current architecture and implementations present and what can be done to overcome them. So without further ado, let's dig in.

A Fly over the Cuckoo's Nest

First and foremost, like any discussion on system internals, let's take a look at a diagram that captures most of its components and structure from a fifty-thousand foot view:

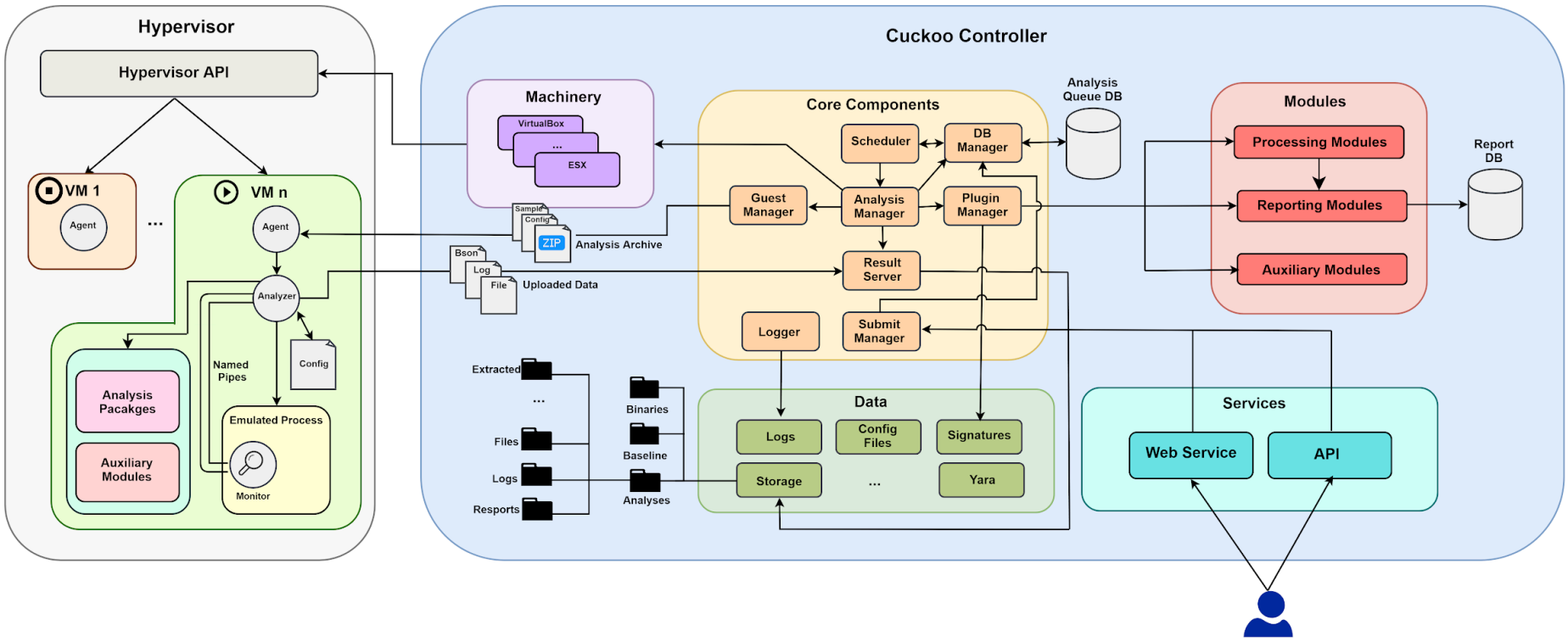


Figure 1: Outline of Cuckoo's architecture

As evident from the above figure, the system can be essentially divided into two parts – the controller (or host) and the guests. The former accepts files as emulation tasks while the latter is a group of machines on which the actual execution of the files takes place. Just before we dive into each part and how things work in it, a few notes:

- The majority of Cuckoo's code is written in Python (while particular components, like the monitor, are coded in C). It's important to keep this in mind as it underlines the system's strengths as well as weaknesses.

- The guests are assumed to be virtual machines, which would be the typical scenario for most Cuckoo deployments. However, it is possible to have them as physical machines, which would obviously require a few adaptations. This doesn't affect the overall architecture.

- The controller is presented here as an 'external' machine. In most cases, it is actually common to have it as yet another VM under the same hypervisor. The prerequisites in either case are that the controller can communicate with the hypervisor via an API (which should be supported by Cuckoo), and that it is able to reach the guests through the network.

- Cuckoo can be adapted to run on Windows, Linux and OSX guests, but Windows is the most common option for a guest of the three, and we will assume it to be the guest platform throughout this article.

And now for some details on the controller part (which can also be referred to as 'the nest').

Every emulation naturally starts with a user submitting a file to the system. In Cuckoo this can be done in one of two ways – through a **web interface** or **API**. In either case, the analysis task would be given an id and the request would be formatted as a JSON by the corresponding server side code. This will be in turn forwarded to the **Submit Manager** (core/submit.py), which is the first component to handle the request in Cuckoo's core and is in charge of reading, interpreting and converting the JSON into an object that can be then inserted into an SQL database.

The database referred to here is the entity that represents the system's **analysis queue**, and will be populated with information about the tasks that go through it. Such information is comprised of metadata that regards the task itself (e.g. status, when did it start\end, which guest machine handled it, etc.) as well as fields related to the submitted sample (e.g. size, file type, hashes, etc.). Of course, this is not the only information kept there, but for lack of time we won't go through all of it. Instead, you can find all classes of kept data and their fields in core/database.py.

The same piece of code (core/database.py) also contains the **Database Manager**, a class that handles the creation of a user for the aforementioned DB and provides functions for interaction with it. This particular component will be used by the **Submit Manager** mentioned earlier to insert the user's task into the queue. The supported database systems in this case are MySQL, PostgreSQL or SQLite. One of these should be installed on the host and back the queue's data.

Once a task gets inside the queue, it is ready to be fetched and taken care of. For this purpose, Cuckoo leverages a **Scheduler** component (core/scheduler.py) which would poll the queue indefinitely for new tasks and handle them as they come along. Since it would have to interact with the database, it would do so through the **Database Manager** that we just described. When it actually retrieves a task, it would first do some checks to verify that the basic conditions for emulation are met (e.g. there are available

guest machines, there is enough space for storing reports on the host and the system hasn't reached a threshold of concurrent analyses) and then pass it on to the next component.

At this point, control gets into the hands of to the **Analysis Manager** (also part of core/scheduler.py). This component is in charge of the bulk amount of management work in the system. It does mainly bootstrapping by acquiring a target guest machine, preparing an analysis configuration file, notifying the **Result Server** on the initiated task (more on this later) and updating the **analysis queue DB** on the task's status. During the rest of the emulation it will communicate with various components to see the analysis process through.

Before conducting the emulation, it is necessary to start the guest machine from an existing snapshot. In order to do so, the **Analysis Manager** would be required to invoke a call through dedicated **machinery**. This term relates to a set of modules that are in charge of interacting with the virtualization software via API, where only one of them can be used at any given Cuckoo setup. This module is capable of doing things like starting, stopping and reverting the guest machine's state, all of which are necessary in the course of emulation. The list of supported machineries can be found under the machinery directory, and can be extended by the user upon usage of a non-supported platform.

So far, the system did mainly preparations for the analysis, but now it is ready to actually start it. For this purpose, the **Analysis Manager** will hand over the control to the **Guest Manager** (resides in core/guest.py), a class that handles communication with a guest component called the **agent** (data/agent/agent.py). The latter is a Python web server that should run on the guest machine's snapshot and wait for a contact from the former.

The **Guest Manager** would in turn prepare a zip file that will contain several analysis resources (**analyzer**, **monitor**) and forward them to the **agent** along with a configuration file and the sample to execute. In the next section, we will see how these are consumed to conduct monitored execution on the guest's side.

Digging Deeper into the Guest

As mentioned above, we will now switch to the guest and take a look on emulation from its perspective. Once again, the guest must have a memory snapshot of a 'clean state' where a Python

process of the **agent** component is executed in the background. This will act as a server that receives data and commands and invokes callbacks corresponding to each one. The underlying protocol for sending data\commands is HTTP (the old version of the **agent** used XMLRPC for this and is still supported).

Recall that the last thing the **Guest Manager** did on the controller's side was to pass an analysis archive, a configuration file and the sample to execute. All of them are going to be used throughout the analysis, but let's first take a closer look at the archive and understand its components – the **analyzer** and **monitor**.

The **analyzer** (/data/analyzer) is a class that handles initialization and execution of the analysis procedure. It will be invoked by the **agent** and in turn carry out a sequence of actions – first it would do some preparations like granting its own process elevated privileges (*SeDebugPrivilege* and *SeLoadDriverPrivilege*), which would be needed for DLL injection in subsequent stages. It would also create two pipe servers, which will use underlying named pipes (with random names generated at run-time) to communicate with the monitored process and to receive logs from it accordingly (the latter is called the log pipe server).

Next, the **analyzer** has to determine what is the proper analysis package required for the execution of the file. An **analysis package** is a class that describes how the **analyzer** should execute the emulated file, based on its type (for instance executing a PE or an Office document would require different actions, as the former can be executed by spawning a new process while the latter would require execution of a relevant Office application). All supported packages can be found under data/analyzer/Windows/modules/packages.

Once the correct package has been determined, and before letting it actually execute the file, there is another set of modules that will be invoked by the **analyzer**. These are called the **Auxiliary modules**, and they define procedures that ought to be executed in the guest in parallel to the analysis process. An example for such a module is human.py that would mimic human interaction in the machine by moving the mouse randomly on the screen and clicking buttons. All such modules can be found under data/analyzer/Windows/modules/auxiliary.

At this point, the **analysis package** can invoke execution, but in order to trace the created processes behavior it would have to instrument it. For this purpose, execution will start from a binary component called **execsc.exe** (can be found at data/analyzer/Windows/bin) that will execute the required process for the underlying file in a suspended state, inject a DLL component into it called the **monitor** using a technique called APC injection and then resume the process.

When code is injected into a suspended process using this technique and then resumed, the injected code will be invoked as a callback prior to the execution of the first scheduled thread in that process, which is exactly what will happen here, and allow the **monitor** to do some bootstrapping actions (see a diagram with general flow of APC injection below). These will include (and are not restricted to) initialization of a global data structure with system info using the *GetSystemInfo* API, initialization of pipe clients, initialization of the Capstone disassembly engine and usage of it for hook installation.

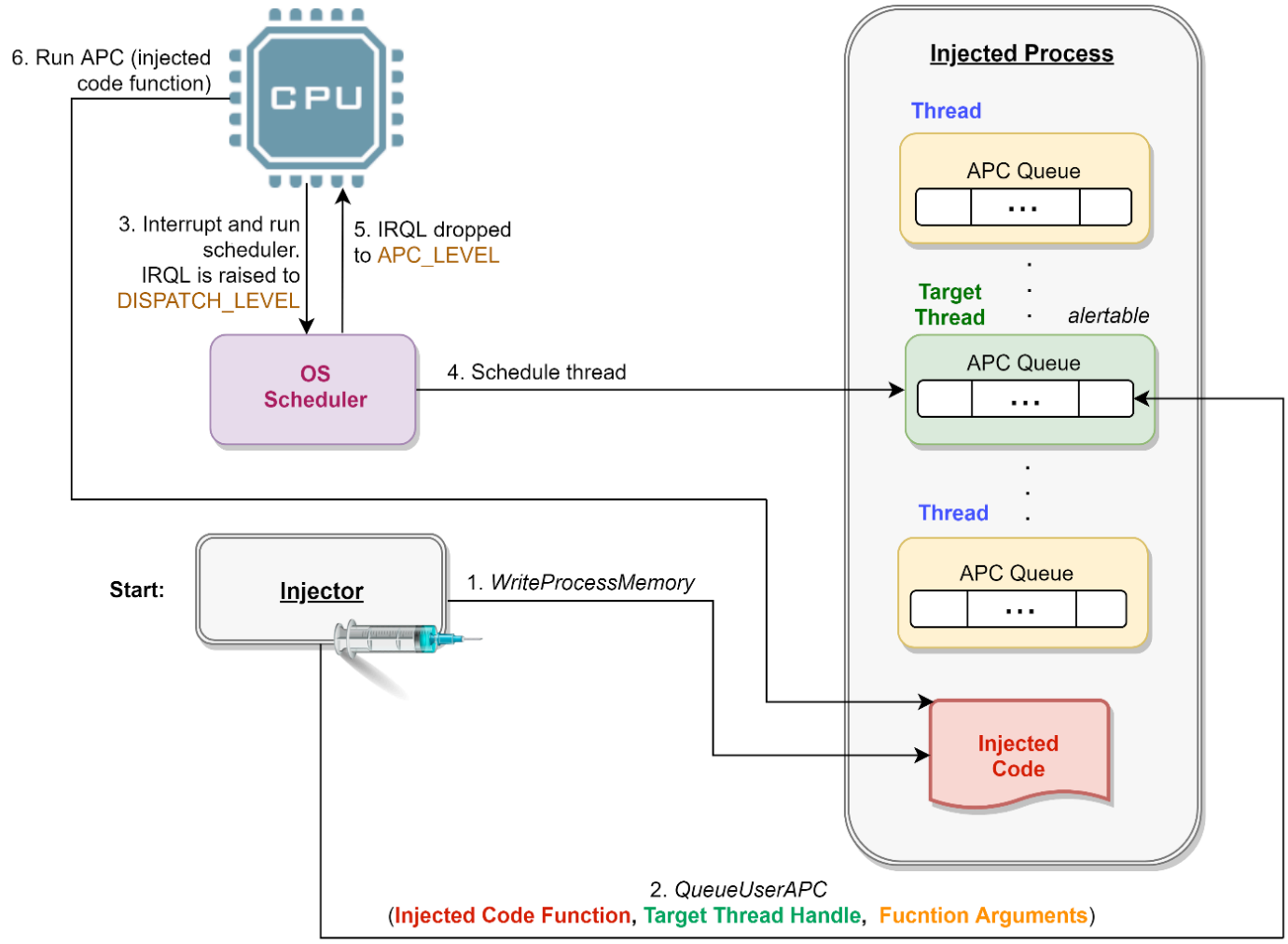


Figure 2: Outline of the APC injection technique flow, used for injecting the monitor into the analyzed process

As far as hooks are concerned, the **monitor** will install these (using an inline hooking method outlined in a figure below) based on signature files. The signatures for the hooks have a predefined format that describes which function should be hooked and its particular hook's logic. Upon compilation of the **monitor**, code will be emitted to implement the requested hooks. This component comes with compiled signatures out of the box, but in case one wants to incorporate others not supported, it is possible and fairly easy to do by just following the guidelines outlined here.

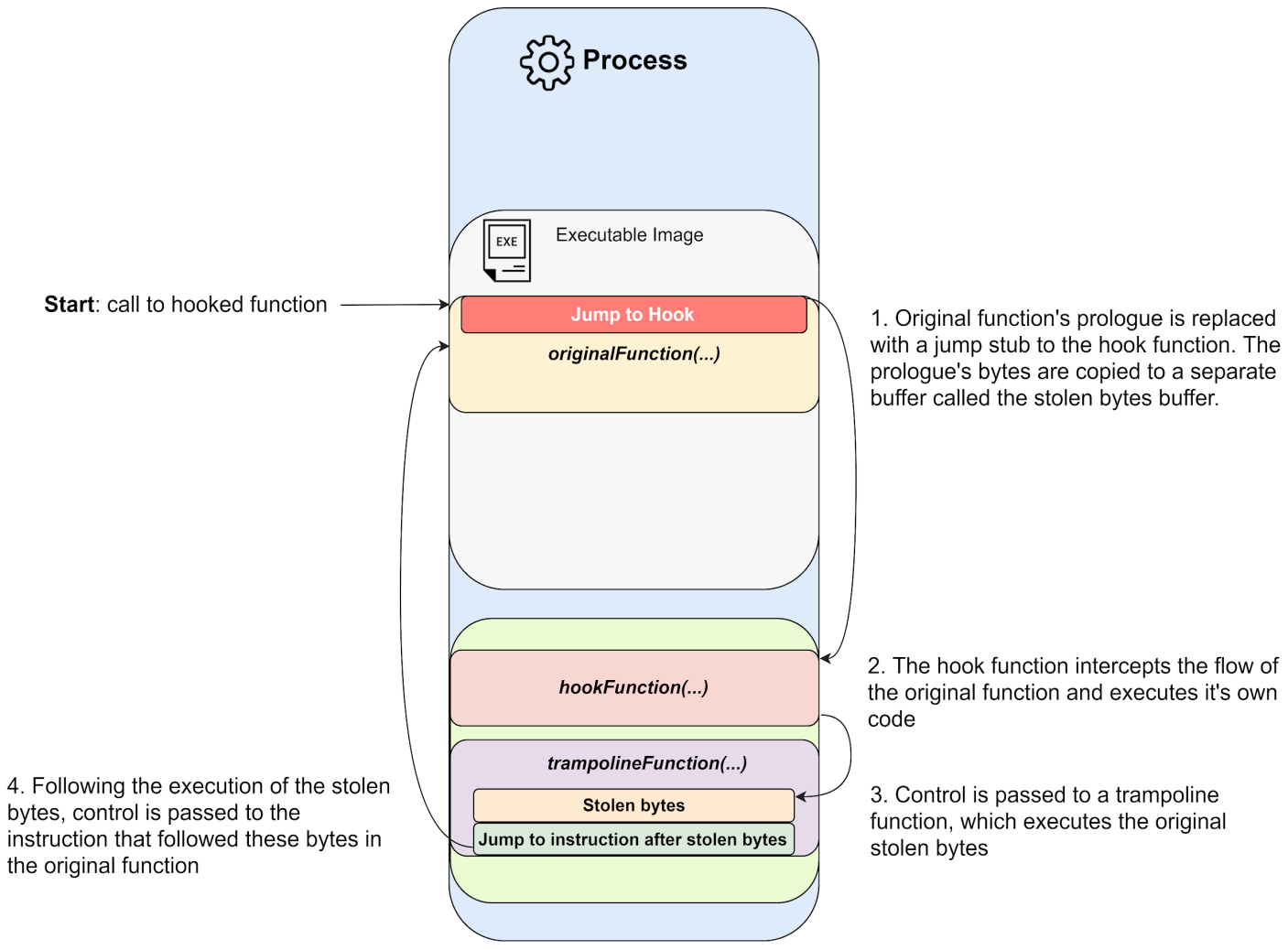


Figure 3: Inline hooking technique, used to intercept API calls by the monitor

During emulation, data from the hook intercepted functions will be collected by the monitor and gathered into a BSON file. This (along with other artifacts, like dropped buffers, files, network traffic captures, etc.) will be in turn passed back to the analyzer via a named pipe, and then forwarded on to the controller machine to a component called the **Result Server**. In the upcoming section, we will see how this data is processed back at the controller to create the final Cuckoo report designated for the user.

Back to the Controller - Finalizing Analysis

At this stage, the controller has gathered quite a bit of data about the emulated sample, all of which was passed through the network to the **Result Server**. As the name suggests, this is a multi-threaded TCP server component that simply awaits incoming analysis data from the guest.

The analyzer and the result server negotiate any upload through a predefined protocol (outlined in /data/analyzer/Windows/lib/common/results.py) which allows three types of upload – a BSON, a log or a file. Upon each upload, the server would create a new thread to handle it through a result handler which would create a protocol parser that corresponds the underlying uploaded data type.

Recall that during the initial stage of analysis, the **Analysis Manager** registered the server with a task, so once incoming data is received, the thread that handles it is associated with this task. For each uploaded artifact, the **Result Server** would eventually write it into the analysis directory on the host (under /data/storage), which would have the id number of the analysis as the directory name.

Now the data is hosted on the controller side but it's still raw and not ready to be consumed by the user, at which point the **Analysis Manager** makes use of the **Plugin Manager**, a class that allows applying various modules on the data to process it. These modules can be divided into two main types (other than **Auxiliary Modules**, which we already discussed) – **Processing** and **Reporting modules**.

The **Processing modules** are the first to be invoked on the gathered raw data and each one of them defines a way to enrich it. For instance, the network.py is designed to take PCAPs recorded during emulation and extract information like DNS traffic, contacted domains, IPs, HTTP requests, etc. The user may add any custom module of his choice to the processing directory and it will be invoked at this stage, given that it follows the conventions specified here and that it's enlisted in the processing.conf configuration file.

The results of these modules will all be gathered into a global container - one giant Python dictionary object, which is referred to by the Cuckoo developers as the "fat dict". The **Analysis Manager** would further enrich these results by applying signatures on them (both standard Cuckoo signatures that look for user defined patterns in analysis results as well as Yara rules that are applied on binary files). Once again, both kinds of signatures can be extended by the user.

Finally, the **Analysis Manager** would pass control to the **Reporting Modules**, which are able to convert the data in the global container into various formats that can be consumed by the user. Two prominent examples that come out of the box with Cuckoo are the jsondump.py module, which would save the analysis results as a JSON report to the host storage directory corresponding to the analysis task, as well as mongodb.py that is in charge of storing the report data in MongoDB database (should run on the host machine).

The latter is the **report database** shown in Figure 1, and is actually used to lay out the analysis details in a web front-end (part of the web service mentioned at the beginning of this article), which is one of the common ways to consume Cuckoo results.

Conclusion - Is Cuckoo the Ultimate Solution?

Now that we went through some of the deeper aspects of Cuckoo, it is proper to ask whether it is indeed the ultimate sandboxing solution. In short – no, and here are just a few reasons based on what we discussed so far:

- Cuckoo provides an infrastructure for automating dynamic analysis, but it does not facilitate the guest's capability in coping with anti-VM\anti-sandbox techniques. The virtual machine on which the malware runs is as prone to evasions by malware as any other VM, and it's fully up to the user to fine-tune it to resolve such issues. Fortunately, several tools are available online to assist with that. A notable one called InviZzzible was created just for Cuckoo by Check Point researchers Aliaksandr Chailytko and Stanislav Skuratovich, based on a study of Cuckoo evasion techniques that they conducted in 2016.

- On the same note, the fact that Cuckoo's monitor instruments a user space process makes it easier to get detected and bypassed by malware. This can be done by locating the inline hooks in the malware's process, loading copies of the underlying memory modules (e.g. kernel32.dll, ntdll.dll) that won't get hooked by the monitor and running all calls through them. To cope with that, Nicolas Correia and Adrien Chevalier came up with zer0mon, a driver (for Windows XP and 7) which serves as a kernel mode alternative for the standard monitor. It has been incorporated as another option for analysis in the latest Cuckoo release.

- Another problem might be scale. Unfortunately, Cuckoo is not well adapted for dealing with hundreds or even dozens of concurrent analysis tasks properly. An example for this shortcoming is the fact that the processing and reporting phase operates on one analysis task at a time, creating a bottleneck in the scenario when multiple emulations provide results concurrently (you can see the code snippet below to understand this problem better).

```
def process_results(self):
    """Process the analysis results and generate the enabled reports."""
    logger(
        "Starting task reporting",
        action="task.report", status="pending"
    )

    # TODO Refactor this function as currently "cuckoo process" has a 1:1
    # copy of its code. TODO Also remove "archive" files.
    results = RunProcessing(task=self.task).run()
    RunSignatures(results=results).run()
    RunReporting(task=self.task, results=results).run()
```

Figure 4: Processing raw analysis results, one analysis at a time

This can be handled by rewriting this mechanism (for instance by making it multi-threaded), or by disabling processing in 'cuckoo.conf' and then running multiple Cuckoo process instances (thus, multiple processes look for task data to be processed or reported and take care of that). Otherwise, you can always check out Hatching, which aims to provide automated malware analysis at scale for organizations, courtesy of Cuckoo's core developers.

- Finally, it takes a lot of resources to maintain a virtualized environment with multiple machines on premise. Organizations nowadays often turn to cloud solutions to be able to use computing resources on demand, which brings up the next problem – Cuckoo does not have machinery modules to support such environments. This has also been tackled by Check Point's research not long ago and solved for a particular setup of Cuckoo on Amazon. You can read more details here.

To conclude this discussion, we can say that while Cuckoo does present some challenges, a lot of them can be handled by various community contributed solutions. In this sense, it is important to remember that Cuckoo is not meant to be a one shot solution, instead its power lies in the fact that it is open source and highly extensible. After reading the above outline of its internals, you should be able to start investigating Cuckoo's source code, and the ability of enhancing it for your particular needs should be only few steps away. Good luck!

Thanks to Jurriaan Bremer and Ricardo van Zuthpen for their assistance on clarifying various details in this publication, as well as utmost work in maintaining the Cuckoo project.

References:

- The official site:
  https://cuckoosandbox.org
- Haow do I sandbox?!?! – a great outline by Jurriaan Bremer on Cuckoo internals from REcon 2013:
  https://recon.cx/2013/slides/recon2013-Jurriaan%20Bremer-Haow%20do%20I%20sandbox.pdf
- Cuckoo Sandbox Architecture – yet another great resource for understanding how the system is built:
  https://hatching.io/blog/cuckoo-sandbox-architecture
- Cuckoo's Github repository
- https://github.com/cuckoosandbox/cuckoo
- Official documentation:
  https://cuckoo.readthedocs.io/en/latest/
- Monitor's Github repository:
  https://github.com/cuckoosandbox/monitor/
- Monitor's documentation:
  https://github.com/cuckoosandbox/monitor/
- x86 API Hooking Demystified – a great resource by Jurriaan Bremer on the inline hooking implementation used by Cuckoo's monitor http://jbremer.org/x86-api-hooking-demystified/

- Injecting a DLL without a remote thread – an outline by Pavel Yosifovich on various injection techniques, including APC injection, which is used for injecting the monitor to the analyzed process http://blogs.microsoft.co.il/pavely/2017/03/14/injecting-a-dll-without-a-remote-thread/

About the Author

Mark Lechtik (@_marklech_, Check Point Software Technologies LTD.) is the malware research team leader in Check Point, and has been working there in several research positions for the past five years. He was born in Russia, but lives most of his life in Israel, where he graduated the Ben-Gurion University with a B.Sc. in communication system engineering. His day to day routine deals with reverse engineering and malware analysis both as an occupation and hobby. He enjoys deep diving into a variety of malware, digging out their gory technical details and outlining their underlying stories and threat actors. During his work in Check Point, he was engaged in designing and implementing Cuckoo based pipelines for various projects.

Automated Malware Analysis with Cuckoo Sandbox

by Januar Sugeng

In **automated analysis**, malware is submitted to a dedicated system that will perform automatic initial analysis. This way usually gives similar results to the static analysis and dynamic analysis. This article will focus on the automated analysis using Cuckoo Sandbox (https://cuckoosandbox.org/) version 2.0.6.

Introduction

Malware analysis might be the most interesting experience to work at for cybersecurity professionals. It's a process that uses various tools and techniques to determine how malware is working. It can be a time consuming task, especially when you are analyzing advanced malicious code and there isn't a single mechanism on how to analyze such code. Most common methods include static (or code) analysis and dynamic (or behavior) analysis, and automated analysis.

In the **static analysis** method, the code structure is analyzed without executing or running the malicious code. The purpose is to understand the functionality and any characteristic features that could be used to identify and create a unique malware signature.

In **dynamic/behavior analysis**, the malware's code is intentionally run in a controlled isolated environment (sandbox) to observe its behavior. We can observe what changes it makes to the O/S, file system changes, changes in the Windows registry, changes on the process list, system resources usage, and any other anomalies (e.g. disappearing files). At the same time, we can capture all the network traffic during the analysis to find out what hosts the malware was communicating with and search for

any well-known network traffic patterns (e.g. spam sending). It is often possible to identify addresses of the C&C (Command and Control) servers and specific botnet to which a malware belongs.

In **automated analysis**, malware is submitted to a dedicated system that will perform automatic initial analysis. This way usually gives similar results to the static analysis and dynamic analysis. This article will focus on the automated analysis using Cuckoo Sandbox (https://cuckoosandbox.org/) version 2.0.6.

Started as Google Summer Code Project in 2010 and part of the Honeynet Project, Cuckoo Sandbox is an advanced, extremely modular, and 100% open source automated malware analysis system with infinite application opportunities. By default, it is able to analyze many different malicious files (executables, office documents, pdf files, emails, etc.) as well as malicious websites under Windows, Linux, Mac OS X, and Android virtualized environments.[1] Designed to be used as a standalone system, and can be scaled up into a larger framework, Cuckoo has a web-based Django interface with MongoDB database support. From its website at https://cuckoosandbox.org/ below is the list (not exhaustive) of what it can do:

- Trace API calls and general behavior of the file and distill this into high level information and signatures comprehensible by anyone.
- Dump and analyze network traffic, even when encrypted with SSL/TLS. With native network routing support to drop all traffic or route it through InetSIM, a network interface, or a VPN.
- Perform advanced memory analysis of the infected virtualized system through Volatility as well as on a process memory granularity using YARA.[2]

Cuckoo Sandbox can analyze the following types:

- Windows EXEs
- DLL files
- PDF files
- Microsoft Office (macro enabled) files
- URLs and HTML files
- PHP scripts

- CPL files
- Visual Basic (.VB) files
- ZIP files
- Java JAR
- Python files
- *And many more*

Besides using functionalities from the operating systems (Linux/Windows), Cuckoo Sandbox has many functionalities from other technologies:

- Python 2.7

  - The Cuckoo host components is completely written in Python and currently it only fully supports Python 2.7.

- Virtualization (VirtualBox, KVM, VMWare, etc.)

  - Sandboxing functionalities will be provided by virtualization technologies and, for this article, we'll use Oracle's Virtual Box 6.0.

- TCPDUMP

  - In order to dump the network activity performed by the malware during execution, you'll need a network sniffer properly configured to capture the traffic and dump it to a file. By default, Cuckoo adopts tcpdump, the prominent open source solution.

- Volatility

  - Volatility is a tool to do forensic analysis on memory dumps. In combination with Cuckoo, it can automatically provide additional visibility into deep modifications in the operating system as well as detect the presence of rootkit technology that escaped the monitoring domain of Cuckoo's analyzer.[3]

- Yara

  - YARA is a tool aimed at (but not limited to) helping malware researchers identify and classify malware signatures. We'll use the database from Cuckoo's Community Yara database.[4]

- MiTMProxy

  - mitmproxy is the Swiss-army knife for debugging, testing, privacy measurements, and penetration testing. It can be used to intercept, inspect, modify and replay web traffic such as HTTP/1, HTTP/2, WebSockets, or any other SSL/TLS-protected protocols. You can prettify and decode a variety of message types ranging from HTML to Protobuf, intercept specific messages on-the-fly, modify them before they reach their destination, and replay them to a client or server later on.[5]

- Django & MongoDB

  - Django is Python's web application framework for Cuckoo's web-based application and it's making use of MongoDB as the database.

- VirusTotal

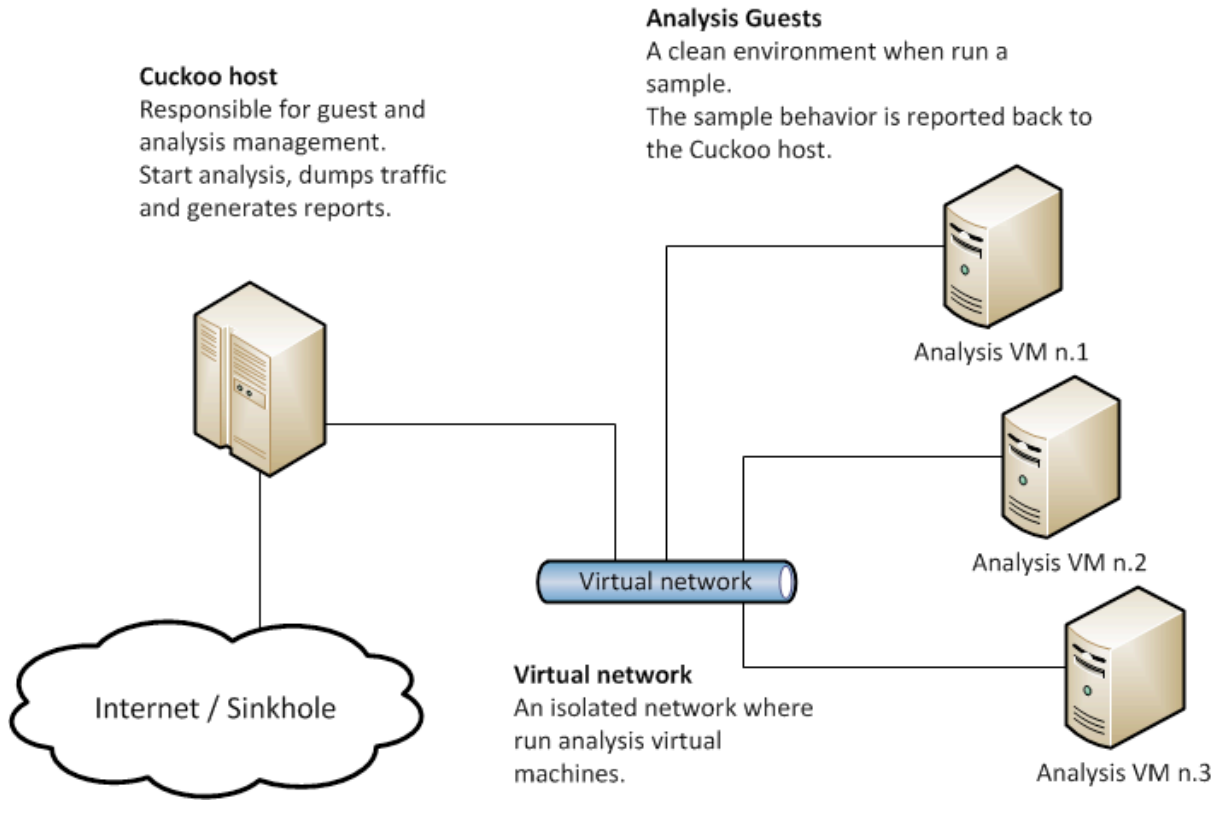
  - VirusTotal (virustotal.com) aggregates many antivirus products and online scan engines to check for viruses that the user's own antivirus may have missed, or to verify against any false positives. Interestingly, VirusTotal for dynamic analysis of malware uses Cuckoo Sandbox. Created by the Spanish security company Hispasec Sistemas in 2004, VirusTotal was acquired by Google, Inc. in September 2012.[6]

Cuckoo Sandbox structure has two main components:

1. **Cuckoo Host**, a central management software that controls the whole analysis process and reporting. This can be an actual physical machine or a virtual machine. We will be using Ubuntu 16.04.5 Virtual Machine for Cuckoo Host.

2. One or more **Guest virtual machines** for malware execution and analysis. We will be using a Windows 7 Pro 32 bit Virtual Machine, running inside the UbuntuVM above (VM running inside another VM).

The structure would be something like the following:



Building the Cuckoo Host

Safety Precaution

The lab environment we'll build must be a safe environment to monitor and analyze a running malicious code, so it must be designed so that malware can't escape to the outside world, which could potentially be harmful to the production environment. Some malware will need a gateway to go online, otherwise it won't function properly, so we also need to have a gateway that we can use to monitor the behavior of the malware while it is connecting to the outside world.

Virtualization technology will allow us to build this special lab and we'll use **Oracle Virtual Box 6.0** throughout this article.

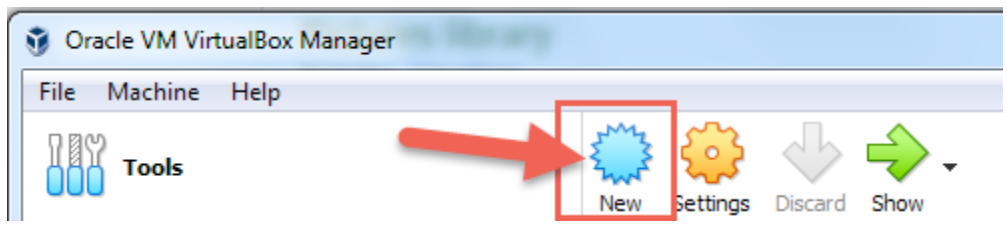
Our **Cuckoo Host** will be installed on a virtual machine running **Ubuntu Desktop 16.04 64bit**. This virtual machine will be running in a **Windows 7 host (physical) PC**. So, you will need to have a working **Windows 7 PC** and **Oracle Virtual Box** installed. I won't go into details on how to install a Virtual Box on a Windows 7. The Internet and Google have lots of examples on how to do this.

Installing Ubuntu Desktop 16.04.5 64bit on VirtualBox as Cuckoo Host

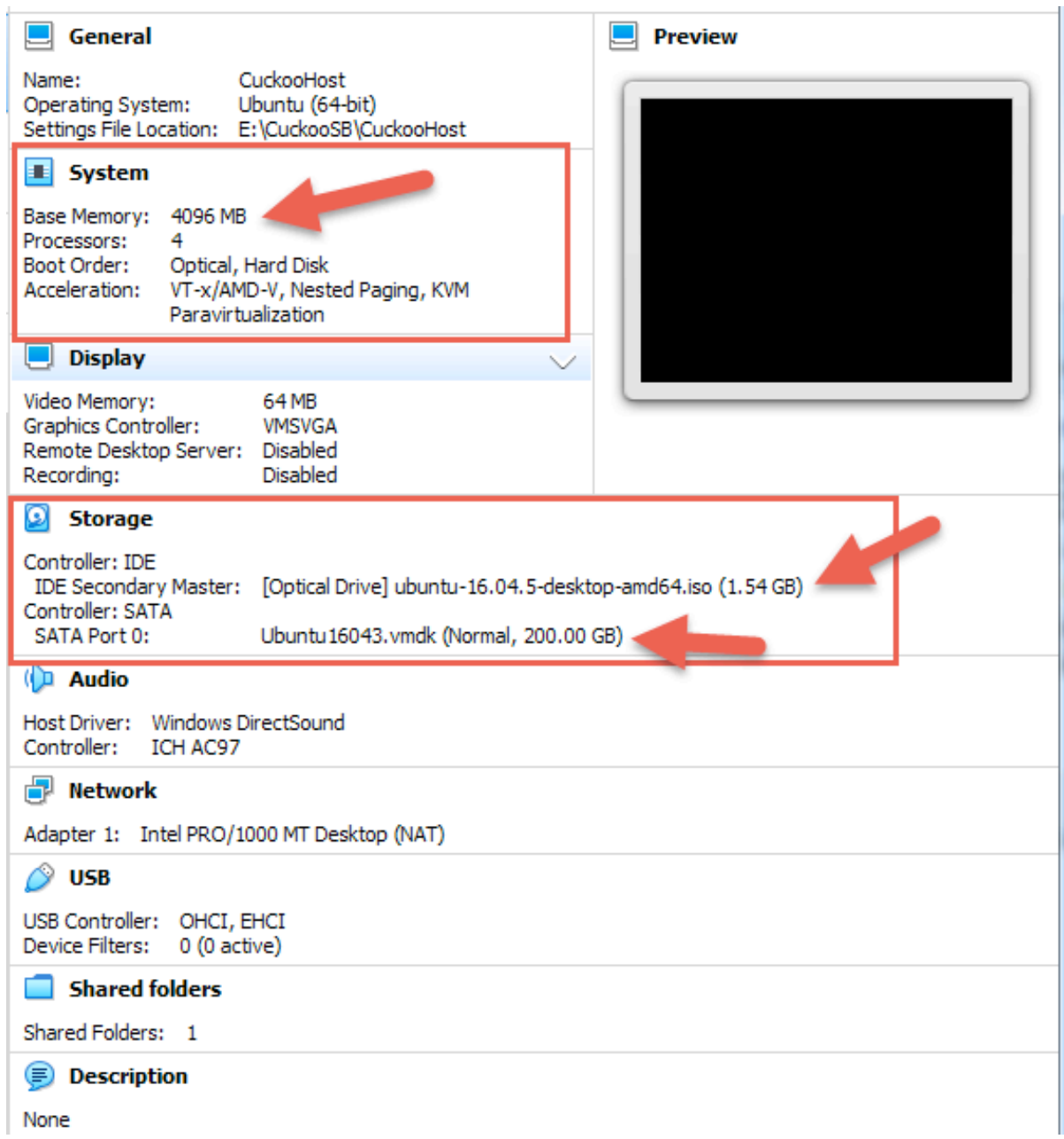
First, download the ISO image here:

http://releases.ubuntu.com/xenial/ubuntu-16.04.5-desktop-amd64.iso

Start VirtualBox and click **New** to create a new virtual machine:



Then use the following as a guide to create your Ubuntu VM:

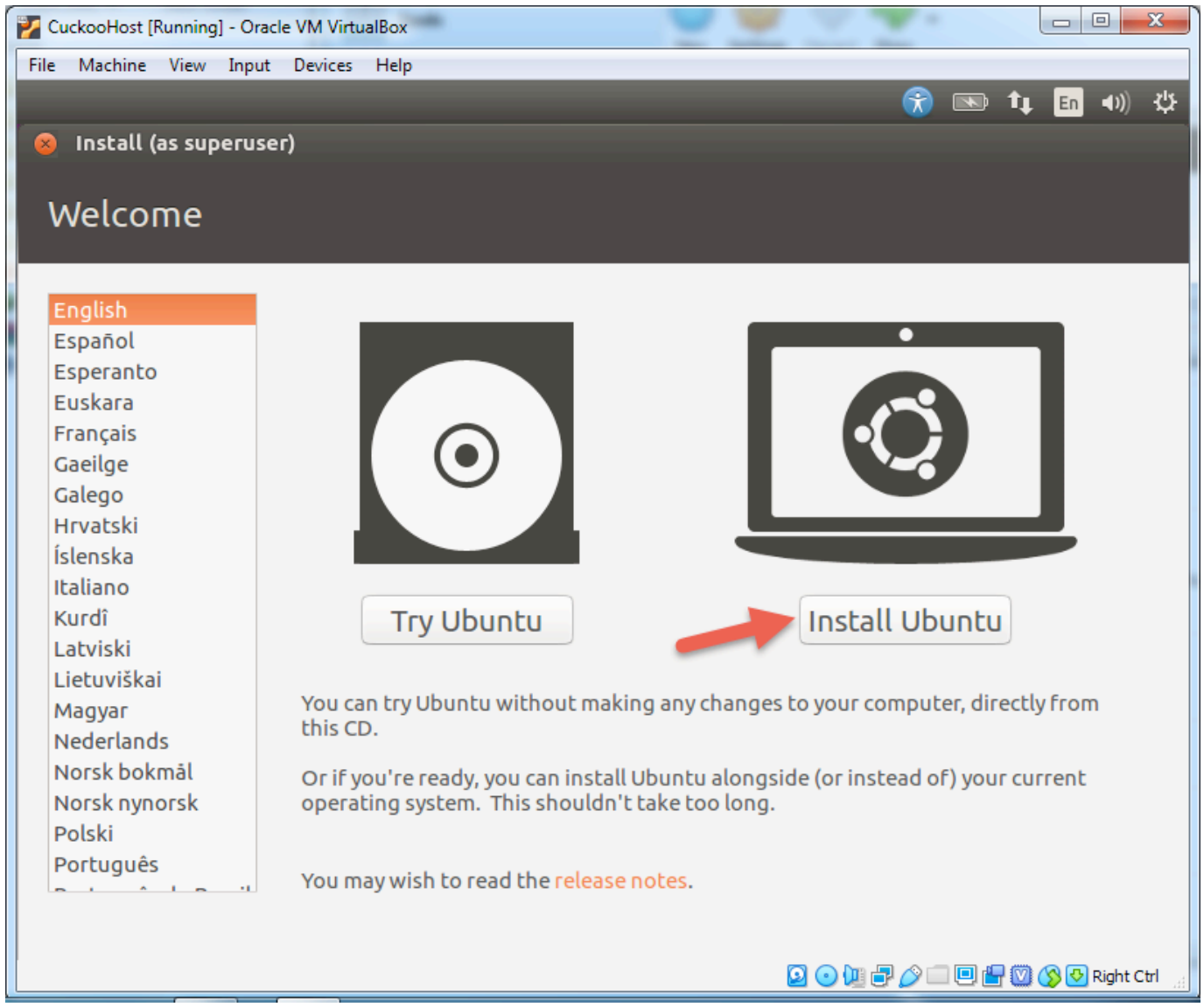


Important:

- Set the RAM to be at least **4GB** (4096 MB)
- Set the virtual hard-disk to be at least **100GB**

Note that the ISO image is mounted on one of the '**Storage**' settings.

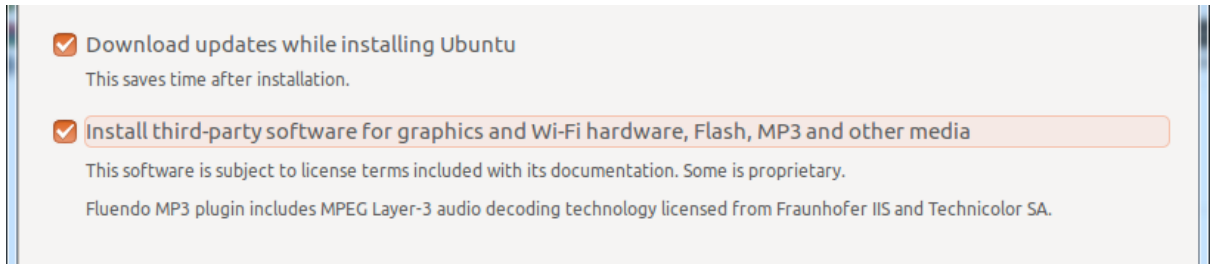
Start the UbuntuVM and it should boot from the ISO image. Let it run and once you see the following screen, select Install Ubuntu:

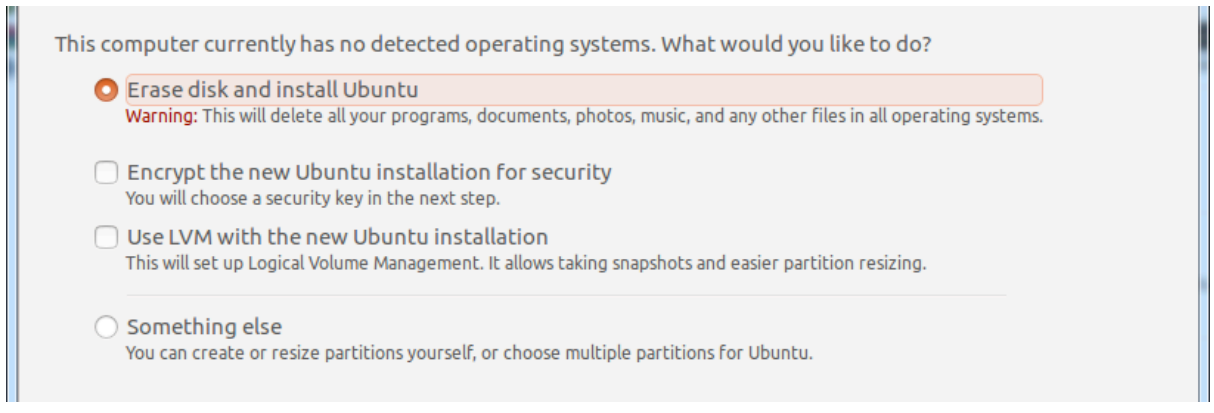


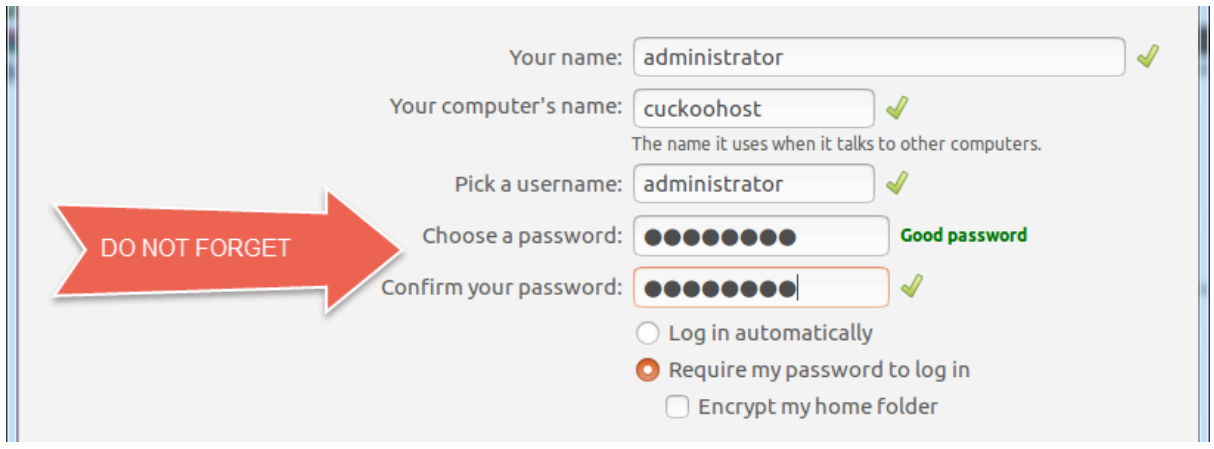
Follow the prompts on the screen, with the following notes:

- You might want to **Download updates and 3rd Party software while installing.**
- Use the entire disk

- Create a User with "**Administrator**" as the user name (and password of your own choosing) and "**cuckoohost**" as the PC name:

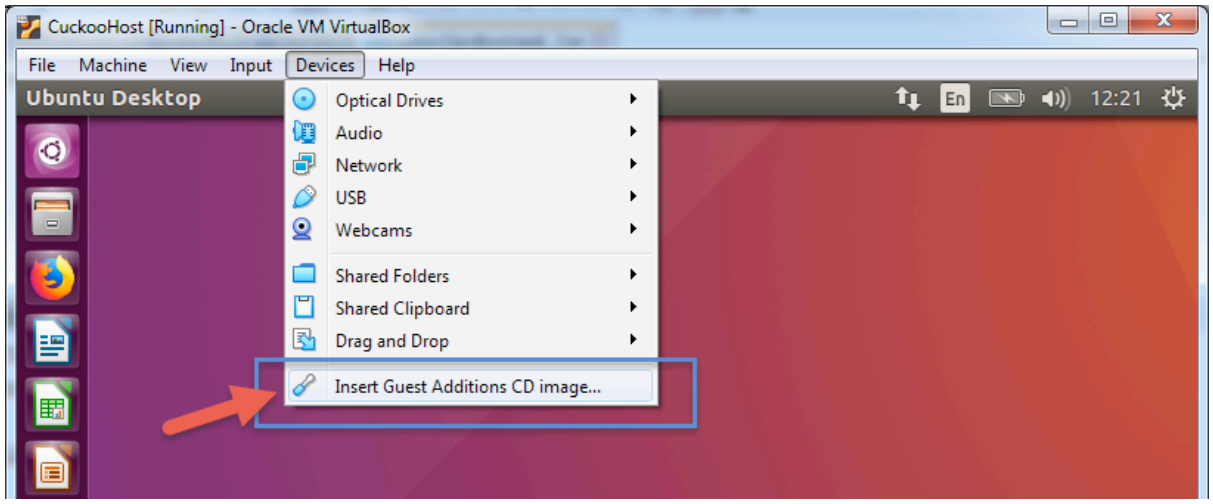




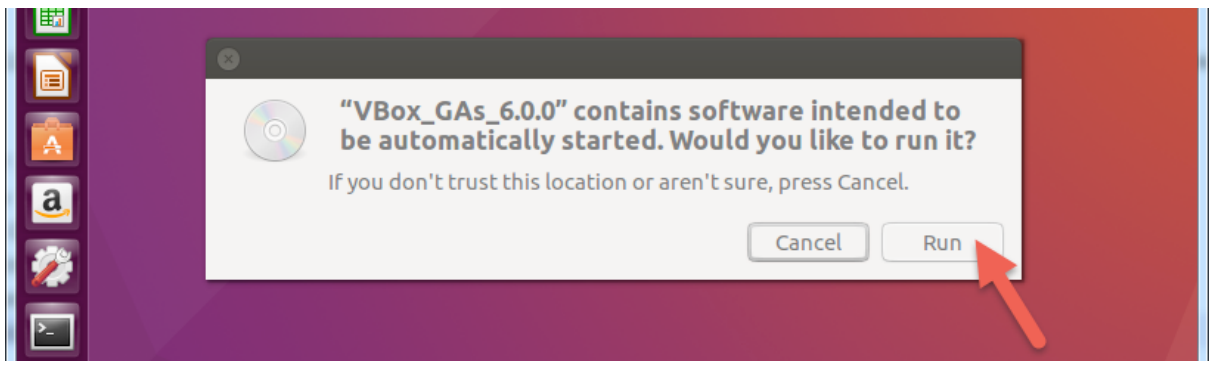


Once completed, the Ubuntu VM is ready to be configured as the Cuckoo Host.

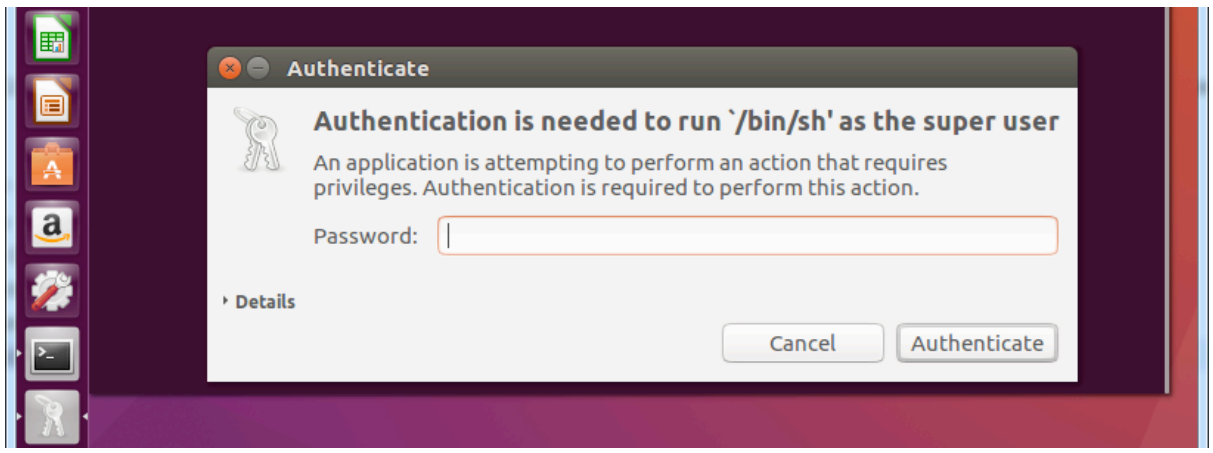
First, we need to install "VirtualBox Guest Additions". Login to the newly created UbuntuVM and open up the Devices menu on the menu bar:



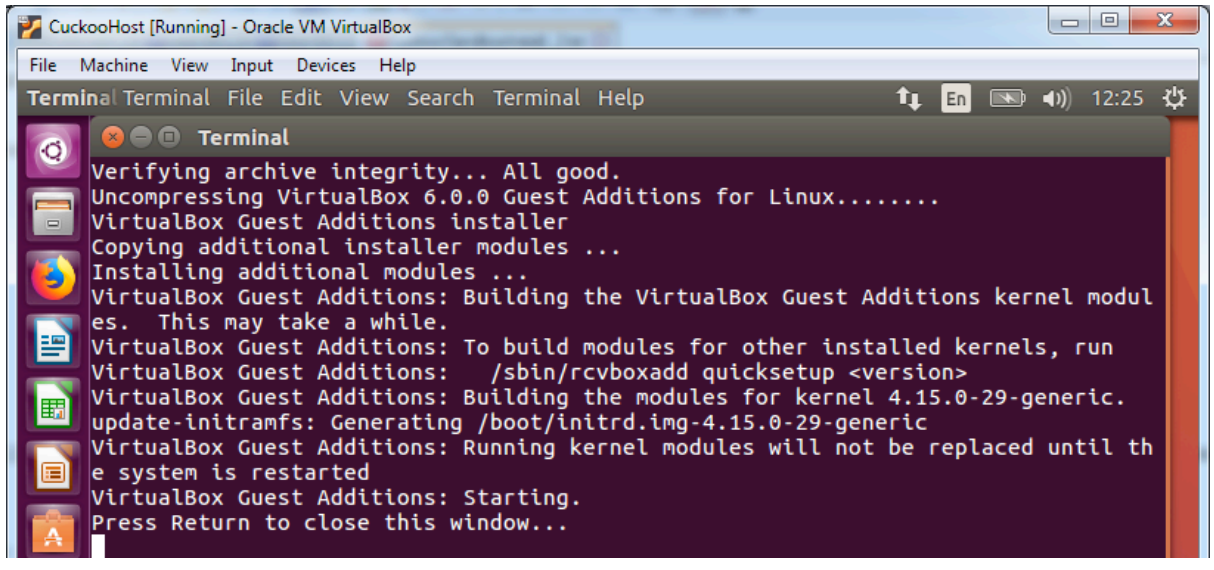
When you get the following prompt, click on the '**Run'** button:



Enter the password and it will start the installation:



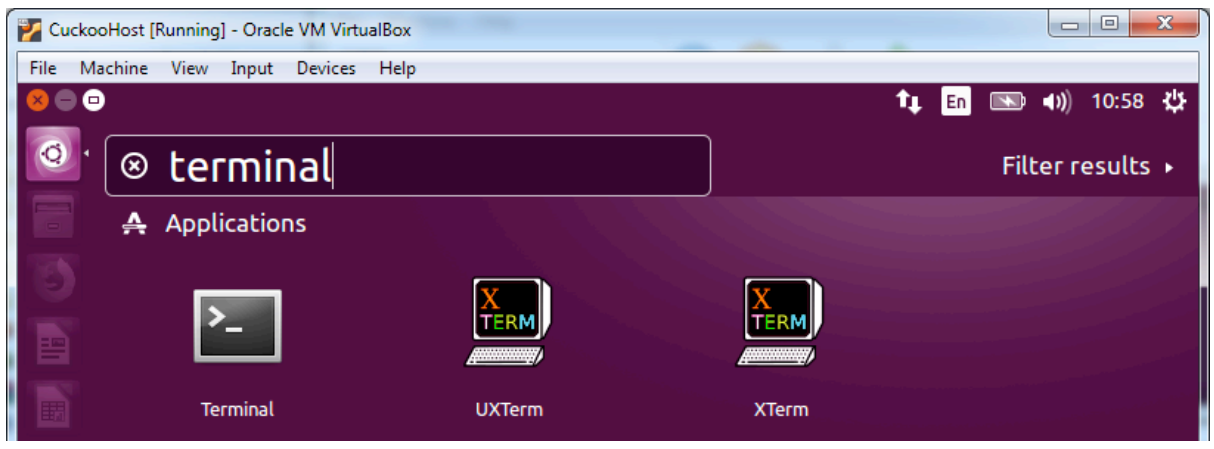
Wait till it completes the Guest Additions installation and restart the UbuntuVM:



Cuckoo Host Installation Steps

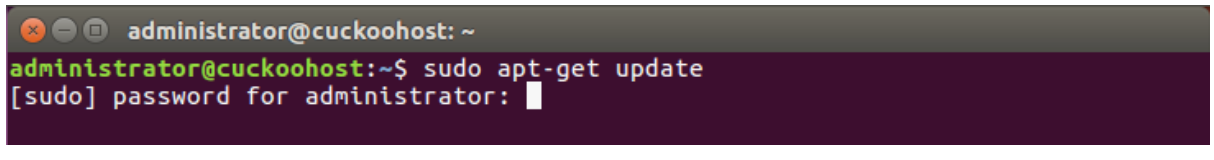
In Ubuntu VM, login as '**administrator'**, or other account you created earlier during installation.

Open a **Terminal** window:



Then type in the following command:

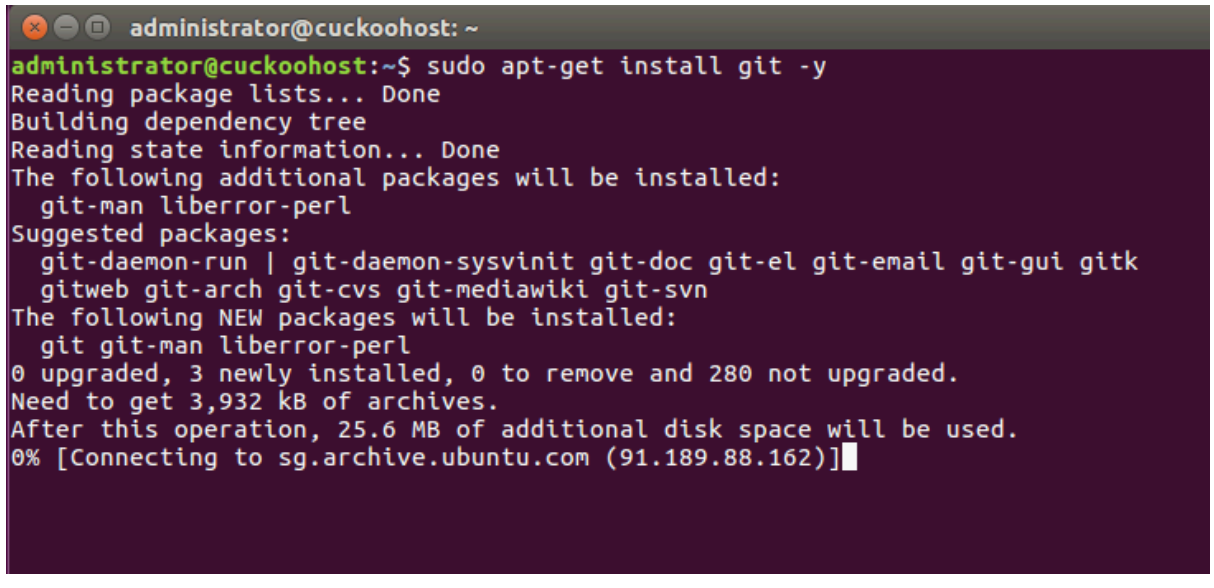
```
$ sudo apt-get update
```



Enter the 'administrator' password if necessary and wait till Ubuntu finishes updating the packages.

Next, enter the following command at the Terminal's prompt to install the 'git' package:

```
$ sudo apt-get install git -y
```

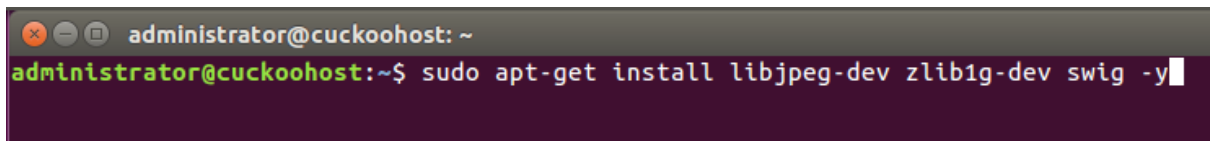


Installing Python 2.7 and other dependencies:

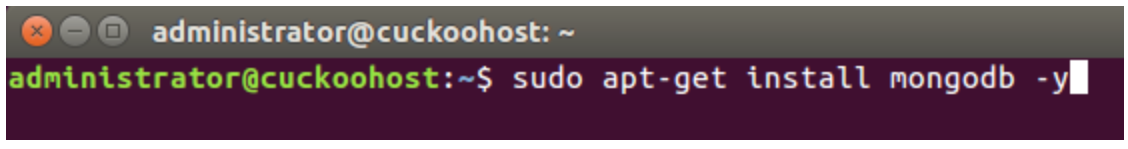
At the terminal's prompt, enter the following command and press **Enter**, one at a time:

```
$ sudo apt-get install python python-pip python-dev libffi-dev libssl-dev –y
```


```
$ sudo apt-get install python-virtualenv python-setuptools –y
```

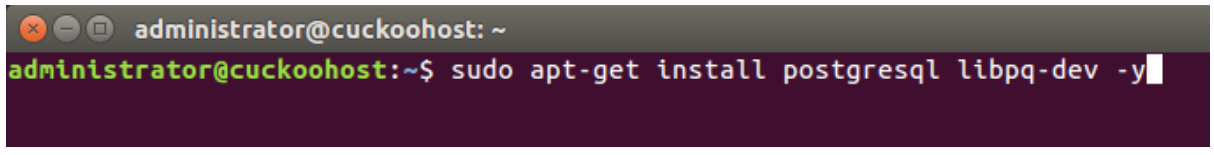



```
$ sudo apt-get install libjpeg-dev zlib1g-dev swig -y
```



We also need to install the MongoDB database as the main database for Cuckoo's Django Web-app:

```
$ sudo apt-get install mongodb –y
```

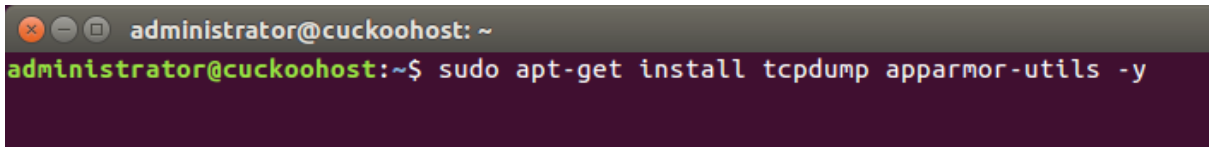


In the future, we may need other databases and while we are at databases, let's install PostgreSQL too.

```
$ sudo apt-get install postgresql libpq-dev -y
```

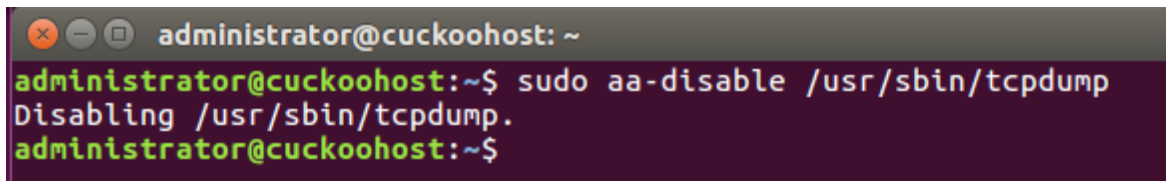
Installing TCPDUMP and AppArmor

```
$ sudo apt-get install tcpdump apparmor-utils –y
```



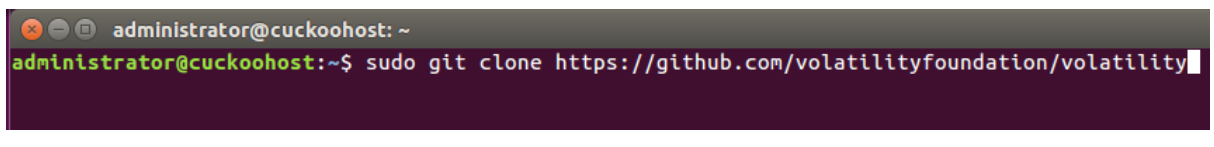
For Cuckoo Sandbox, we need to disable tcpdump profile on AppArmor (see https://cuckoo.sh/docs/installation/host/requirements.html#installing-tcpdump for more information):

```
$ sudo aa-disable /usr/sbin/tcpdump
```



Installing Volatility

```
$ sudo git clone https://github.com/volatilityfoundation/volatility
```

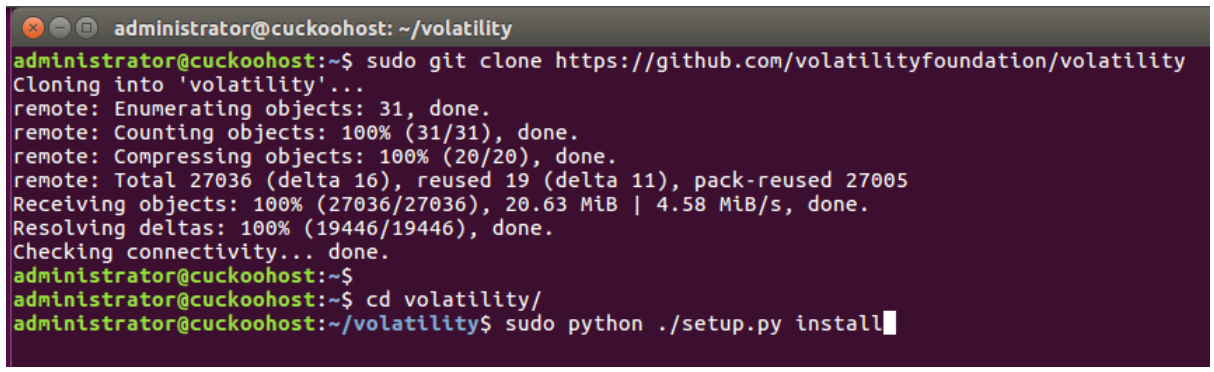


This will download the package into ./volatility directory. Now, navigate to it:

```
$ cd volatility
```

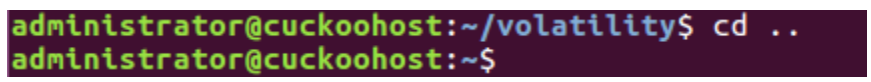
and run its installer:

```
$ sudo python ./setup.py install
```



Once installed, navigate back to the parent folder:

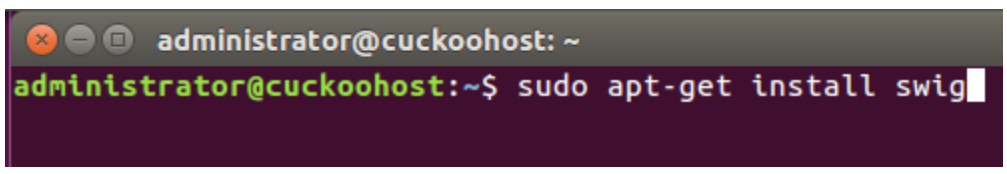
```
$ cd ..
```



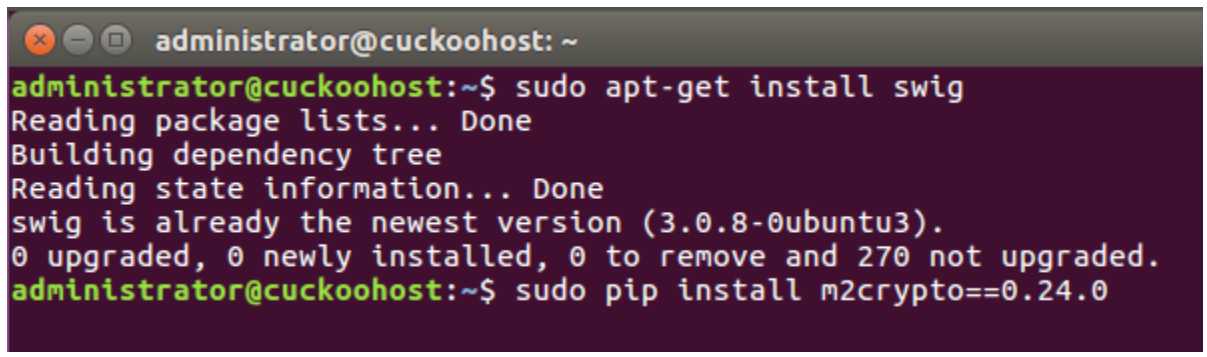
Installing M2Crypto

From its website, M2Crypto is described to be the most complete Python wrapper for OpenSSL featuring RSA, DSA, DH, EC, HMACs, message digests, symmetric ciphers (including AES); SSL functionality to implement clients and servers; HTTPS extensions to Python's httplib, urllib, and xmlrpclib; unforgeable HMAC'ing AuthCookies for web session management; FTP/TLS client and server; S/MIME; M2Crypto can also be used to provide SSL for Twisted. Smartcards are supported through the Engine interface.[7]

```
$ sudo apt-get install swig
```



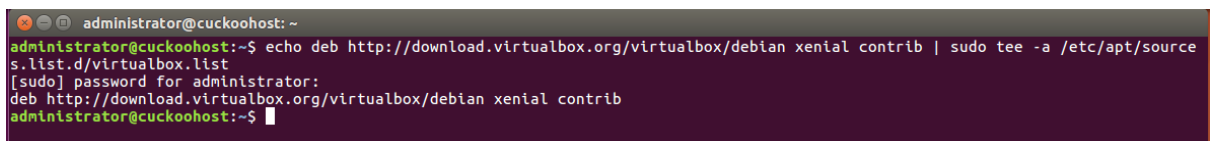

```
$ sudo pip install m2crypto==0.24.0
```



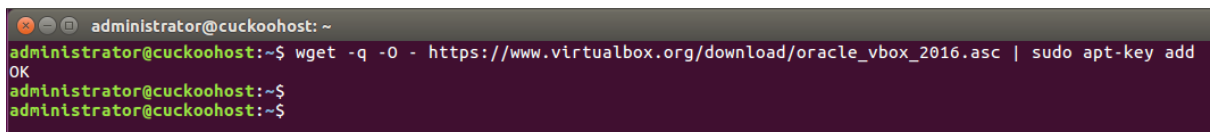
Installing VirtualBox (inside UbuntuVM)

Oracle VirtualBox will be used to create and install our Cuckoo Guest virtual machine(s).

```
$ echo deb http://download.virtualbox.org/virtualbox/debian xenial contrib |
sudo tee -a /etc/apt/sources.list.d/virtualbox.list
```

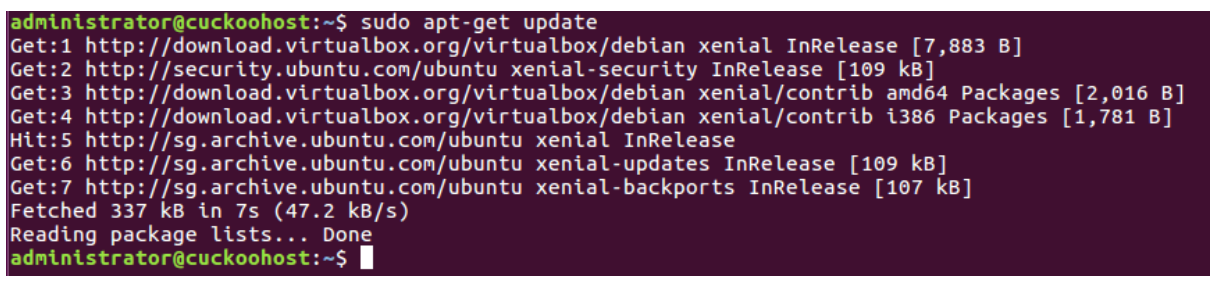



```
$ wget -q -O - https://www.virtualbox.org/download/oracle_vbox_2016.asc |
sudo apt-key add
```



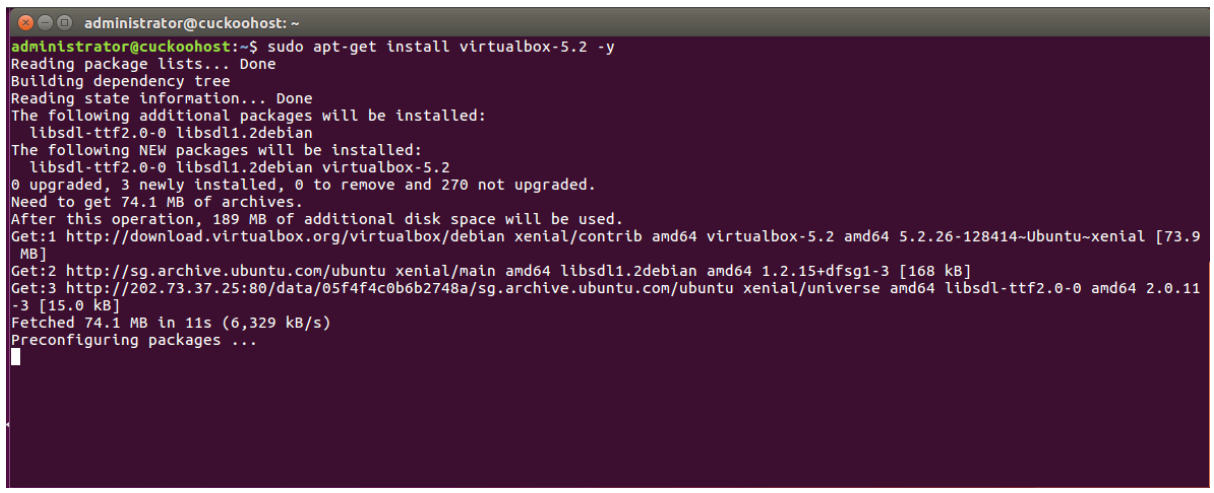
To get the latest update from Ubuntu and VirtualBox, make sure you run the following command:

```
$ sudo apt-get update
```

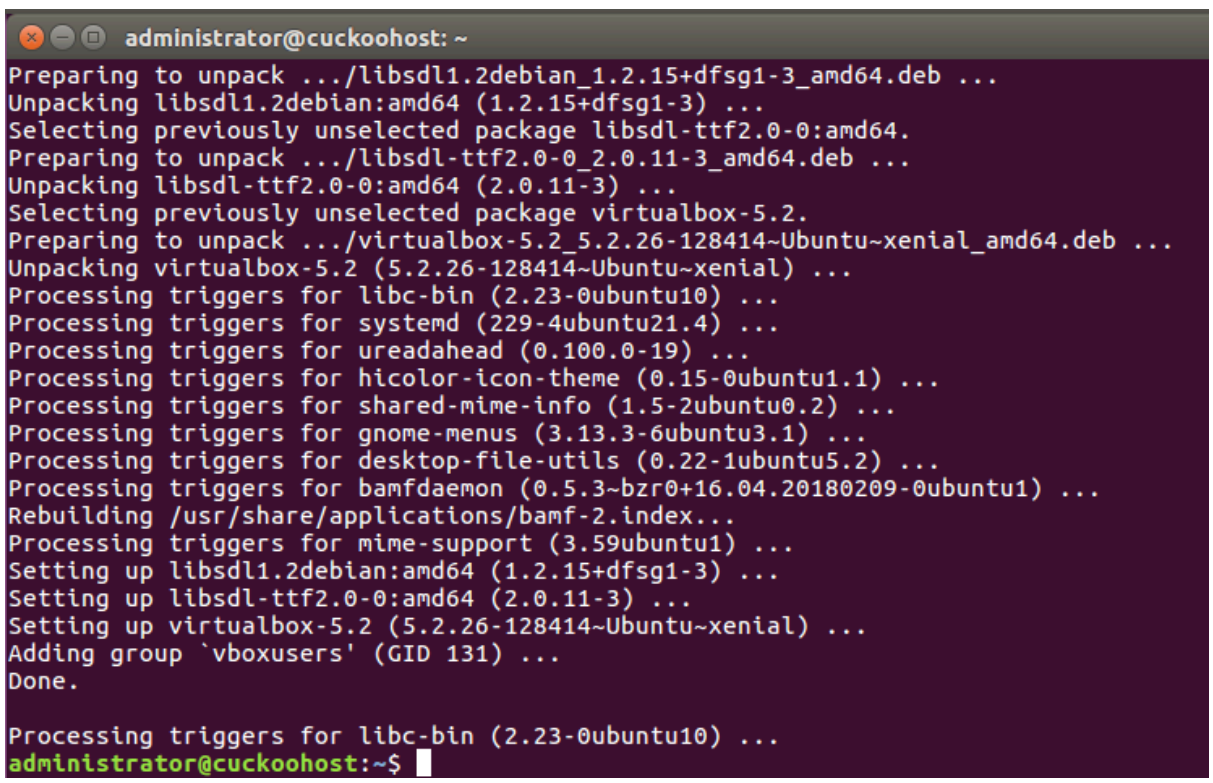


Then install VirtualBox 5.2 inside this UbuntuVM by running the following command:

```
$ sudo apt-get install virtualbox-5.2 -y
```

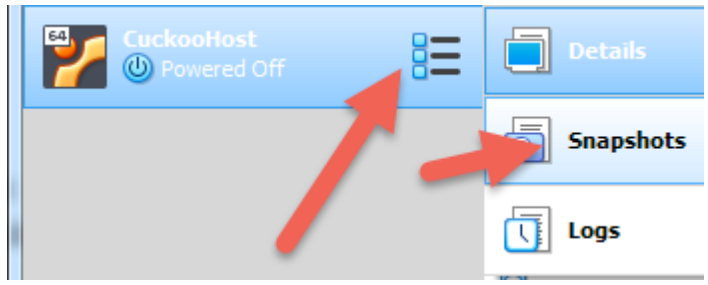


Wait till the installation is completed:

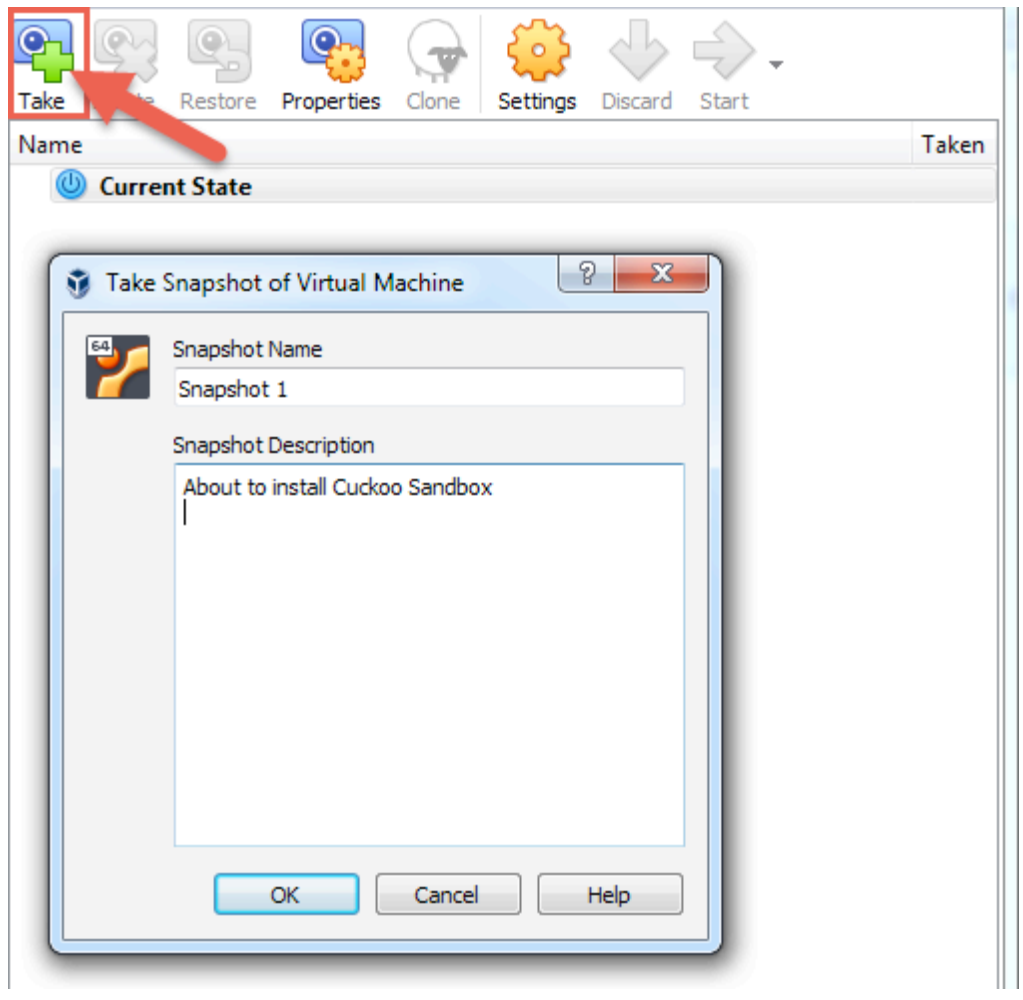


Shut down the UbuntuVM. That's it for the base of Cuckoo Sandbox Host. There are more additional packages to add more functionalities to Cuckoo Sandbox but I'll leave it up to you for later exploration and installation. For now, we have all the required dependencies to install Cuckoo Sandbox itself.

Before we move to the next step, now is a good time to take a snapshot of our UbuntuVM so we can revert back to this state if we made any mistakes in the following steps. To take a snapshot, open the VirtualBox Manager and click on the 3-line menu to open up the drop down menu. Then select **Snapshots** on the dropdown menu.



Click the Take button to take a new snapshot. Leave the name **Snapshot 1** or you can change this if you want. In the Description box, enter "about to install Cuckoo Sandbox"

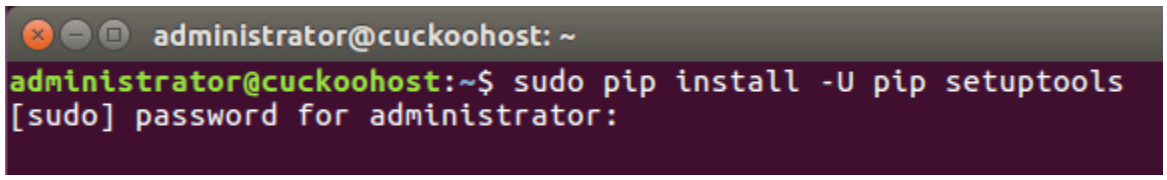


Once completed, start the UbuntuVM, login as administrator and open a Terminal window.

Download & Install Cuckoo

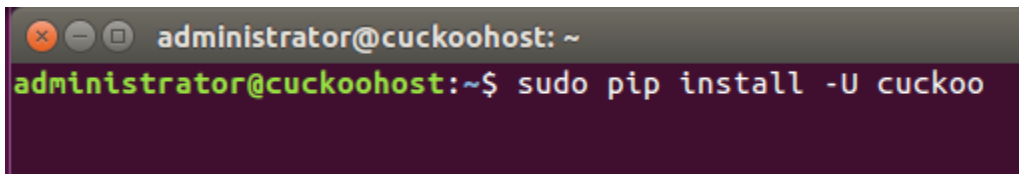
In the Terminal window, type the following command and press Enter:

```
$ sudo pip install -U pip setuptools
```

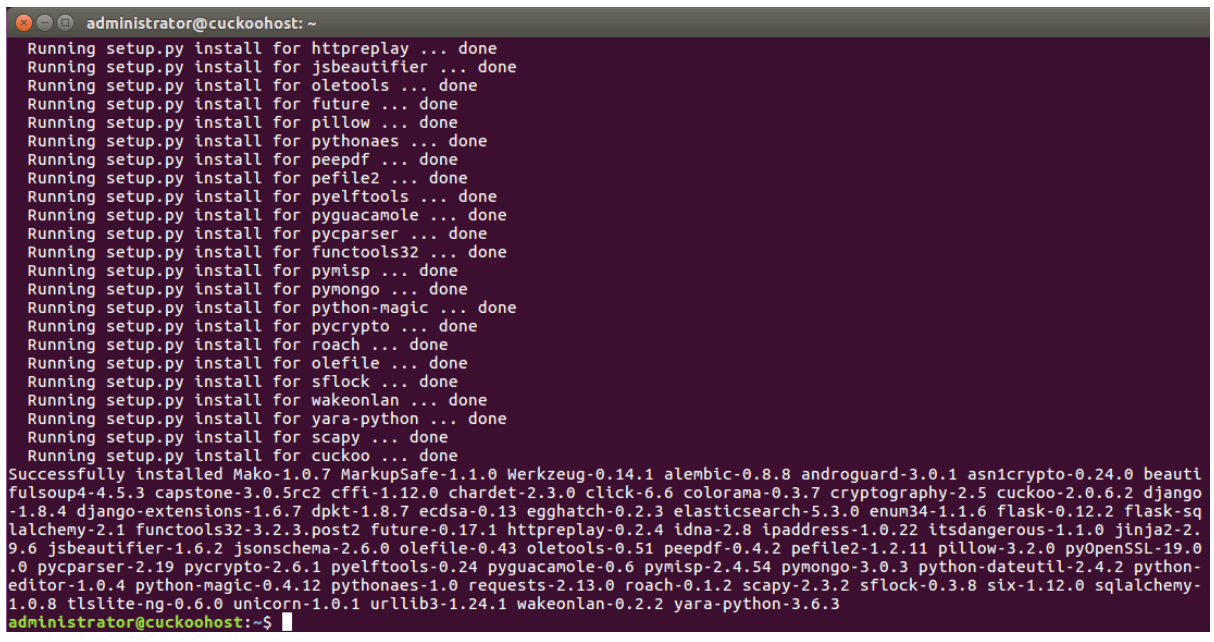


Once completed, we can now install Cuckoo:

```
$ sudo pip install -U cuckoo
```

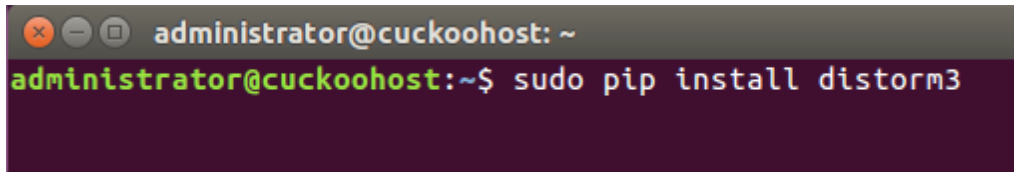


This will install the whole Cuckoo stacks. Wait till it completes:



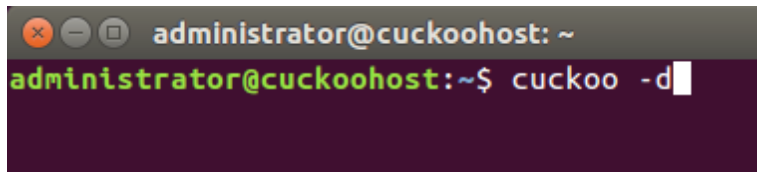
And the last step is to install distorm:

```
$ sudo pip install distorm3
```

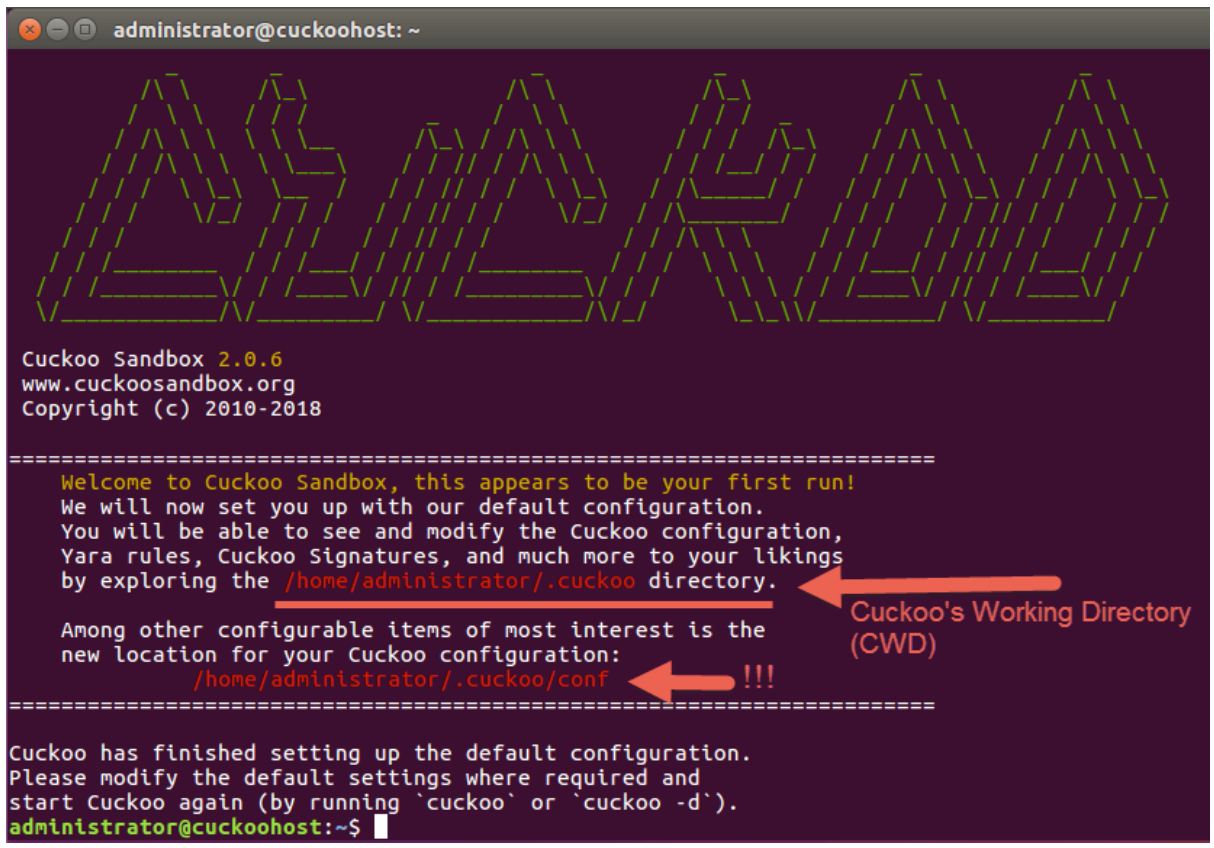


You should now have a working Cuckoo Sandbox Host. To verify that installation is successful, run the following command:

```
$ cuckoo -d
```



If Cuckoo setup is working, you will get the following screen:



This screen also gives you a few pieces of important information, especially the location of Cuckoo's Working Directory (CWD), which in our case is pointing to /home/administrator/.cuckoo.

Note the dot (.) in front of the .cuckoo, this means CWD is a hidden directory. The structure of CWD is as follows:

```
administrator@cuckoohost: ~/.cuckoo
administrator@cuckoohost:~$ cd .cuckoo/
administrator@cuckoohost:~/.cuckoo$ ls -la
total 80
drwxr-sr-x 17 administrator administrator 4096 Feb 16 13:50 .
drwxr-xr-x 18 administrator administrator 4096 Feb 16 13:50 ..
drwxr-sr-x  2 administrator administrator 4096 Feb 16 13:40 agent
drwxr-sr-x  6 administrator administrator 4096 Feb 16 13:40 analyzer
drwxr-sr-x  2 administrator administrator 4096 Feb 16 13:50 conf
-rw-rw-r--  1 administrator administrator   40 Feb 16 13:50 .cwd
drwxr-sr-x  2 administrator administrator 4096 Feb 16 13:40 distributed
drwxr-sr-x  2 administrator administrator 4096 Feb 16 13:40 elasticsearch
-rw-r--r--  1 administrator administrator  163 Jun  6  2018 __init__.py
drwxr-sr-x  2 administrator administrator 4096 Feb 16 13:40 log
drwxr-sr-x  3 administrator administrator 4096 Feb 16 13:40 monitor
drwxr-sr-x  2 administrator administrator 4096 Feb 16 13:40 pidfiles
drwxr-sr-x  9 administrator administrator 4096 Feb 16 13:40 signatures
drwxr-sr-x  5 administrator administrator 4096 Feb 16 13:40 storage
drwxr-sr-x  2 administrator administrator 4096 Feb 16 13:40 stuff
drwxr-sr-x  2 administrator administrator 4096 Feb 16 13:40 supervisord
-rw-rw-r--  1 administrator administrator  957 Feb 16 13:50 supervisord.conf
drwxr-sr-x  2 administrator administrator 4096 Feb 16 13:40 web
drwxr-sr-x  2 administrator administrator 4096 Feb 16 13:40 whitelist
drwxr-sr-x  9 administrator administrator 4096 Feb 16 13:40 yara
administrator@cuckoohost:~/.cuckoo$
```

Spend some time to get yourself familiar with the content of CWD.

Next, navigate to the conf directory and see the contents by using these commands:

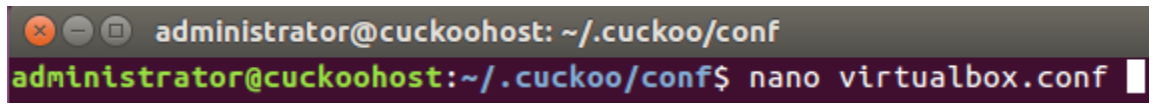
```
$ cd ~/.cuckoo/conf
$ ls -la
```

```
administrator@cuckoohost: ~/.cuckoo/conf
administrator@cuckoohost:~$ cd ~/.cuckoo/conf
administrator@cuckoohost:~/.cuckoo/conf$ ls -la
total 84
drwxr-sr-x  2 administrator administrator 4096 Feb 12 12:57 .
drwxr-sr-x 17 administrator administrator 4096 Feb 16 21:39 ..
-rw-rw-r--  1 administrator administrator 2269 Jan 29 16:32 auxiliary.conf
-rw-rw-r--  1 administrator administrator 2530 Jan 29 16:32 avd.conf
-rw-rw-r--  1 administrator administrator 5892 Jan 29 16:32 cuckoo.conf
-rw-rw-r--  1 administrator administrator 2654 Jan 29 16:32 esx.conf
-rw-r--r--  1 administrator administrator    0 Jun  6  2018 .gitignore
-rw-rw-r--  1 administrator administrator 2740 Jan 29 16:32 kvm.conf
-rw-rw-r--  1 administrator administrator 3454 Jan 30 15:01 memory.conf
-rw-rw-r--  1 administrator administrator 1789 Jan 29 16:32 physical.conf
-rw-rw-r--  1 administrator administrator 5182 Jan 29 16:32 processing.conf
-rw-rw-r--  1 administrator administrator 7292 Jan 29 16:32 qemu.conf
-rw-rw-r--  1 administrator administrator 3662 Jan 30 15:02 reporting.conf
-rw-rw-r--  1 administrator administrator 3922 Jan 29 16:32 routing.conf
-rw-rw-r--  1 administrator administrator 4485 Feb 12 12:57 virtualbox.conf
-rw-rw-r--  1 administrator administrator 2759 Jan 29 16:32 vmware.conf
-rw-rw-r--  1 administrator administrator 3746 Jan 29 16:32 vsphere.conf
-rw-rw-r--  1 administrator administrator 3161 Jan 29 16:32 xenserver.conf
administrator@cuckoohost:~/.cuckoo/conf$
```

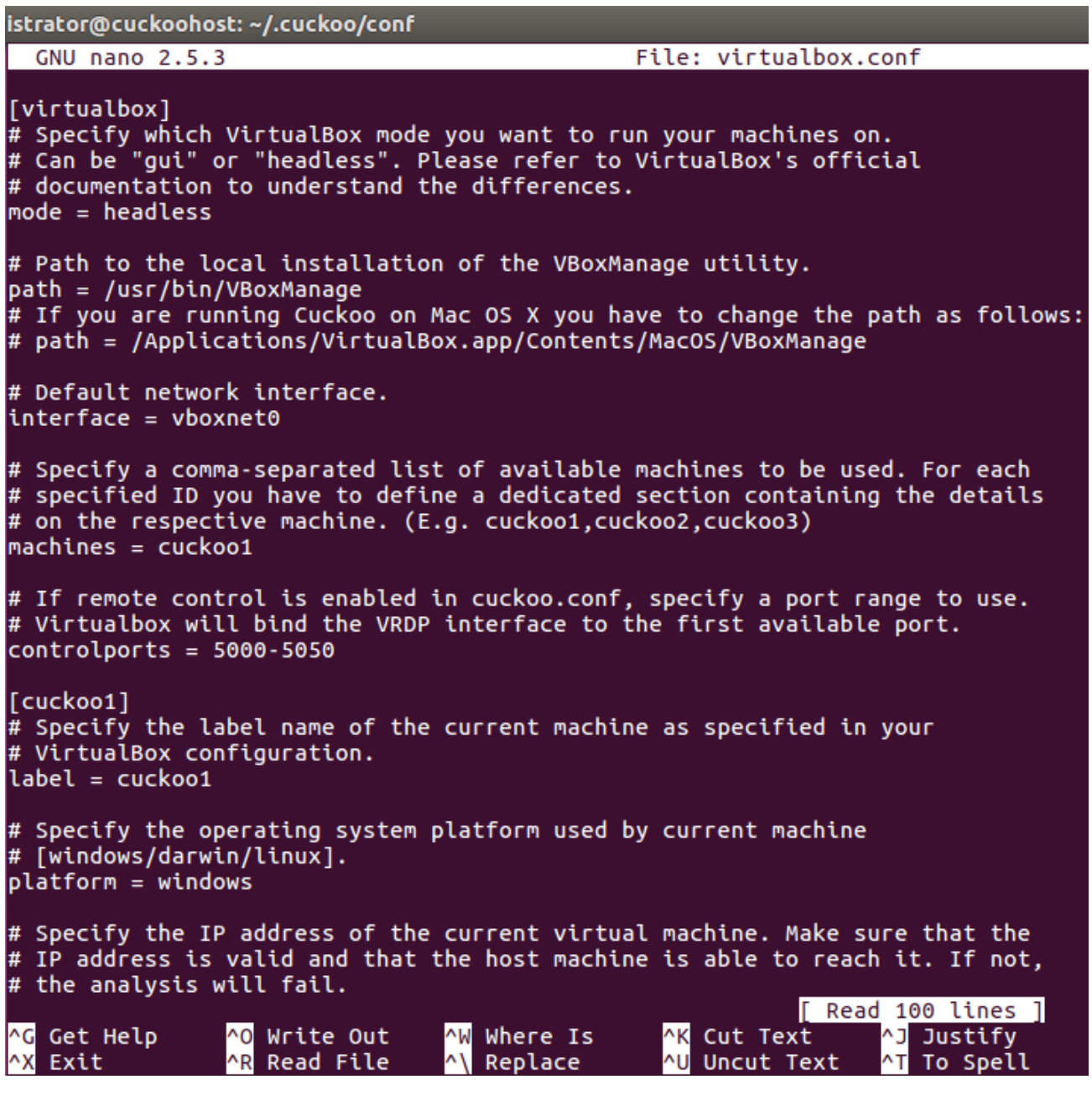
For now, we are just interested in the settings in the virtualbox.conf configuration file. Leave the rest as default.

Open the virtualbox.conf configuration file using nano text editor:

```
$ nano virtualbox.conf
```



The content is something similar to the following:



Find the following entries in the file and note their values. Change those settings according to the following:

```
interface = vboxnet0

machines = cuckoo1

label = cuckoo1

ip = 192.168.56.101

snapshot = Snapshot
```

Save the file by pressing Ctrl+X and answer 'Y' to the prompt to save and overwrite the original file.

Other Configuration Files

Open the memory.conf file:

```
$ sudo nano memory.conf
```

Change the guest_profile to the version of Windows you are using. I am using Windows 7 Service pack 1 32bit so mine will be 'guest_profile = Win7SP1x86'. Then press CTRL + X and 'Y' to save and exit nano.

Open the reporting.conf file:

```
$ sudo nano reporting.conf
```

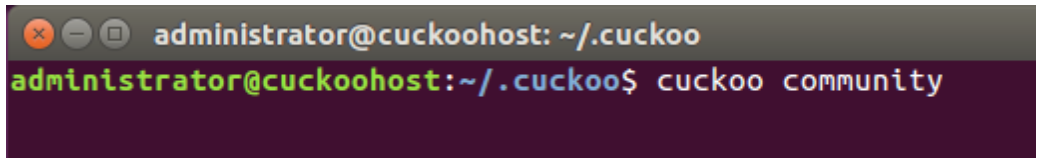
Locate the line with [mongodb]. Ensure 'enabled = yes'. Then press CTRL + X and 'Y' save and exit nano.

Downloading and Using Yara's Community Rules

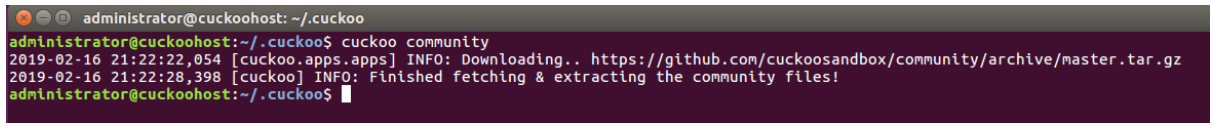
Before we start analyzing any file for any malicious codes, we need to download and use Yara rules provided by the community.

In a Terminal window, navigate to the ~/.cuckoo working directory and enter the following command:

```
$ cuckoo community
```

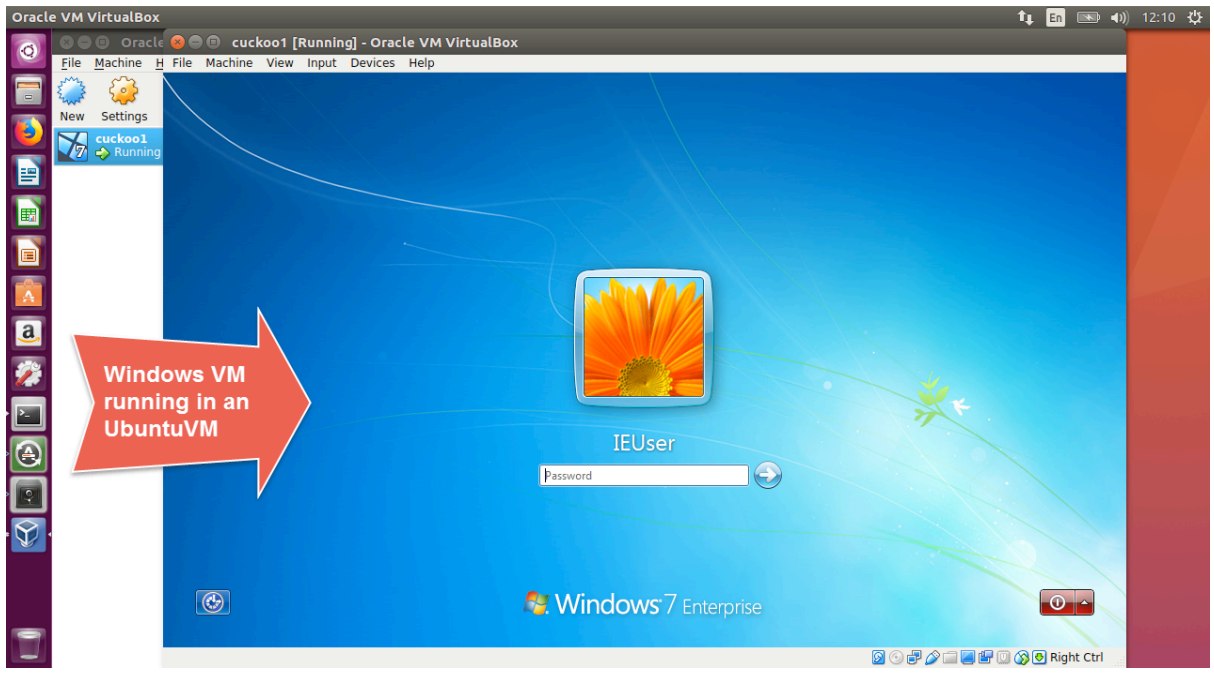


Wait till cuckoo finishes fetching & extracting the community files:



Setting up the Guest Virtual Machine.

You now have a working Cuckoo Host, so it is time to build a Cuckoo GuestVM for our testing environment. This GuestVM will be a Windows virtual-machine running inside the Host UbuntuVM. For this purpose, I chose to use Windows 7 32-Bit (Enterprise or Professional version will do). This is what it should look like once you have it installed:



There are two ways you can do this:

1. Install Windows from scratch from an ISO image. You will need an official ISO image and a valid Product Key for this. You can download the ISO image from this URL:

https://www.microsoft.com/en-us/software-download/windows7

2. For testing purposes, Microsoft provides pre-installed Windows 7 virtual machines in different formats (VirtualBox, VMWare, Hyper-V, etc.), which you can download here:

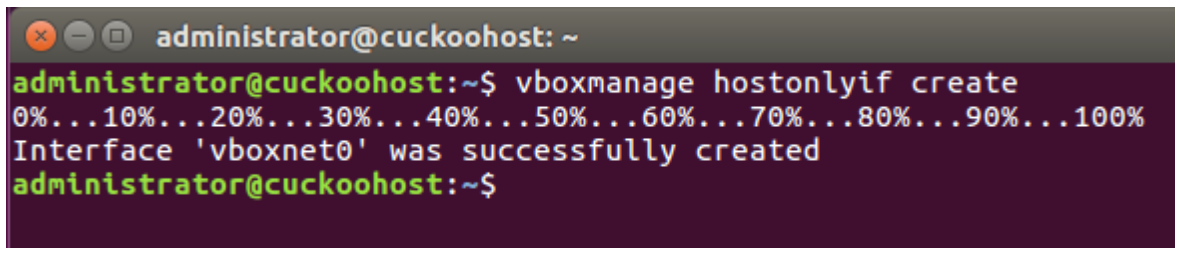
https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/

Download the **IE10-Win7 for VirtualBox** version. The file will be in a ZIP format that you need to unzip. Once unzipped, you'll get one big file in OVA format. Copy and paste this OVA file into the UbuntuVM if it's not already there. We'll use this file later. Note that since Windows was already installed in the OVA file, the username is **IEUser** and the password is **passw0rd**.

However, before we install Windows 7 as our GuestVM, we need to create a virtual switch on the UbuntuVM so both VMs (UbuntuVM and WindowsVM) can communicate during analysis. Since the two VMs will be isolated from the outside world, we need to connect using a **Host-only** network setting in VirtualBox.

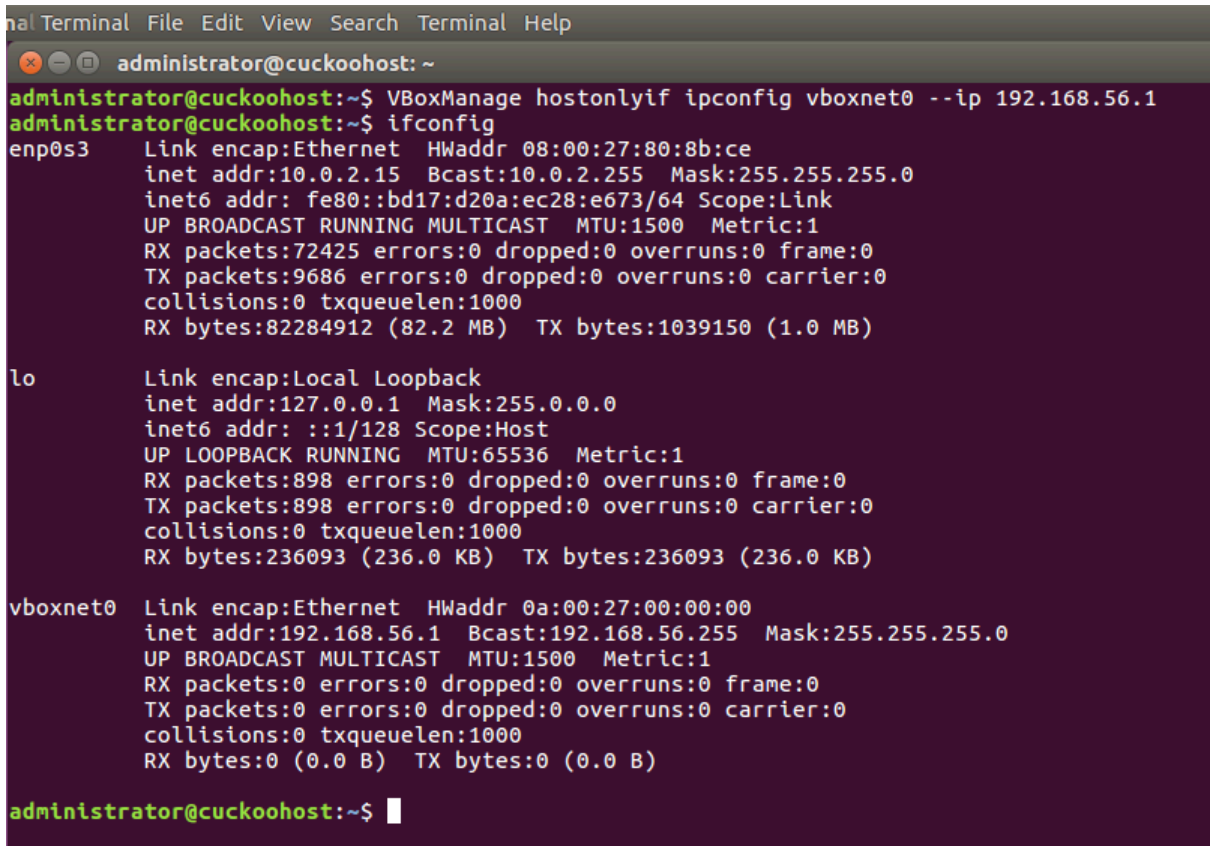
Open a Terminal window and type

```
$ vboxmanage hostonlyif create
```



Assign an IP address **192.168.56.1** to the new **Host-only** interface:

```
$ VBoxManage hostonlyif ipconfig vboxnet0 --ip 192.168.56.1
```

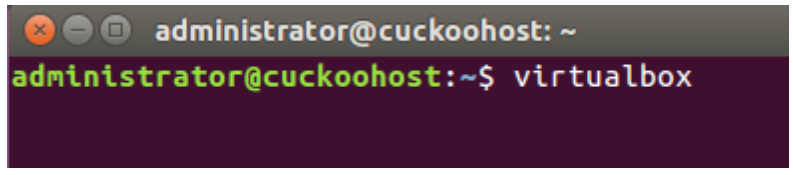


Once completed, use ifconfig command to ensure the network adapter shows up (above picture). It should be called 'vboxnet0' as above. This is also to align the interface name with the settings in the virtualbox.conf file:

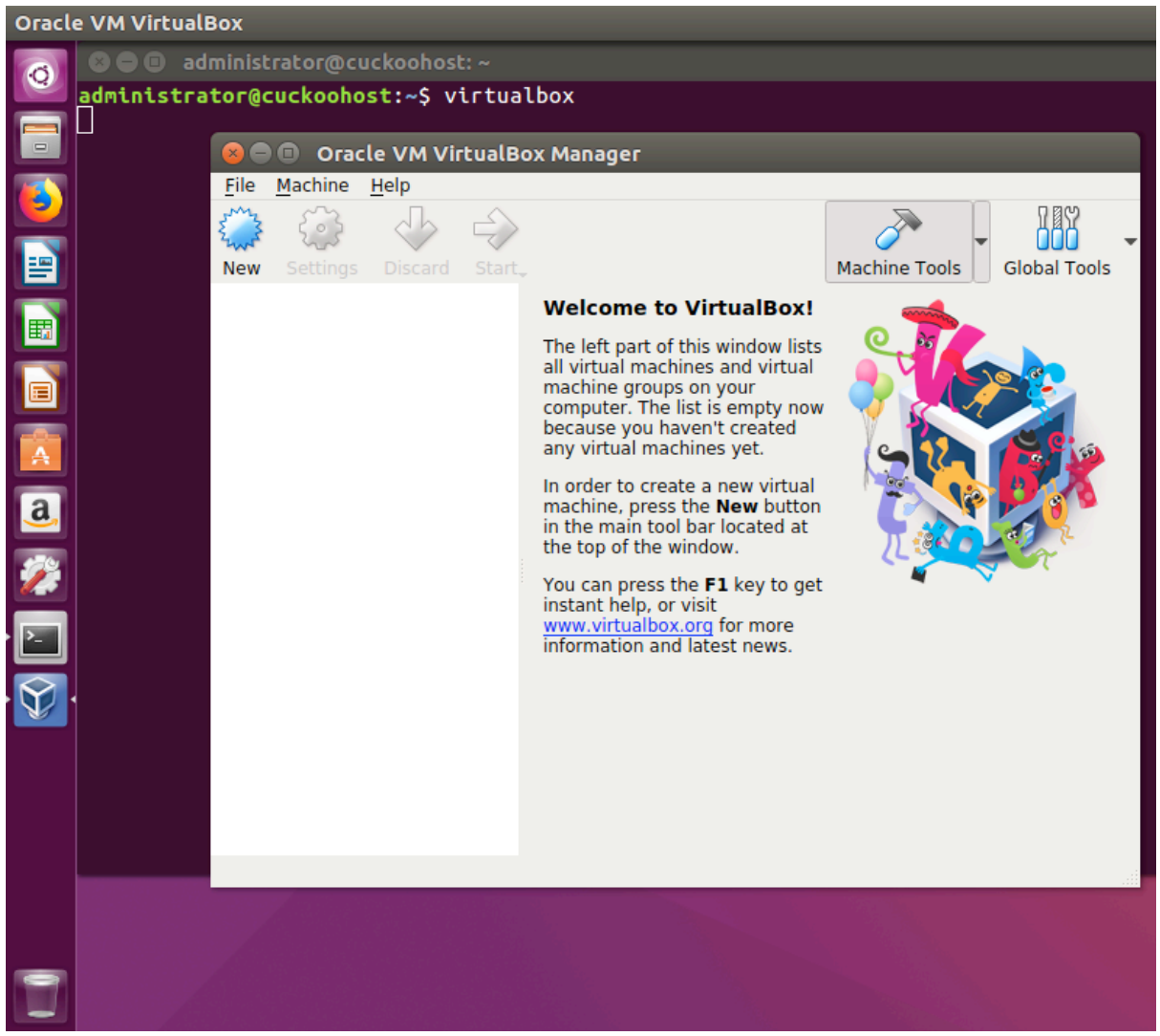
```
# Default network interface.
interface = vboxnet0
```

Now we can start **virtualbox** by typing the following command and press Enter:

```
$ virtualbox
```



And it should give you this familiar interface:



If you are using the ISO image, you need to install Windows 7 from scratch. I assume everyone knows how to install a virtual machine within VirtualBox so I won't repeat the steps again (please refer to the steps on installing the Host UbuntuVM as they are the similar). Install Windows 7 32 bit with the following setting:

Name: cuckoo1 **(This name is important!)**

RAM: 1.5GB (1036MB)

HDD: 32GB

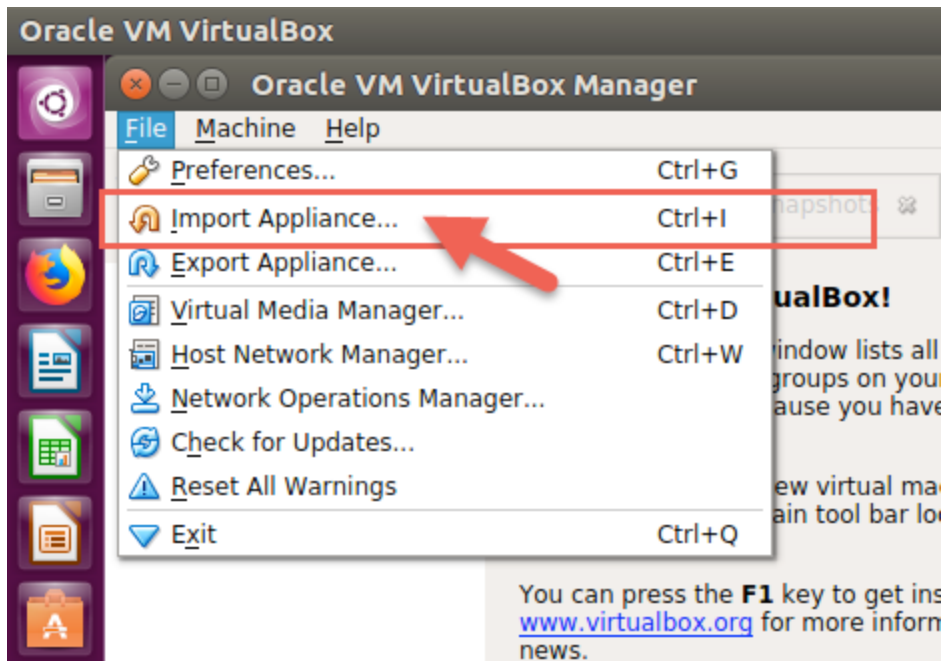
Note: the 'name' must be **cuckoo1**, the same name and label in the virtualbox.conf file:

```
[cuckoo1]
# Specify the label name of the current machine as specified in your
# VirtualBox configuration.
label = cuckoo1
```

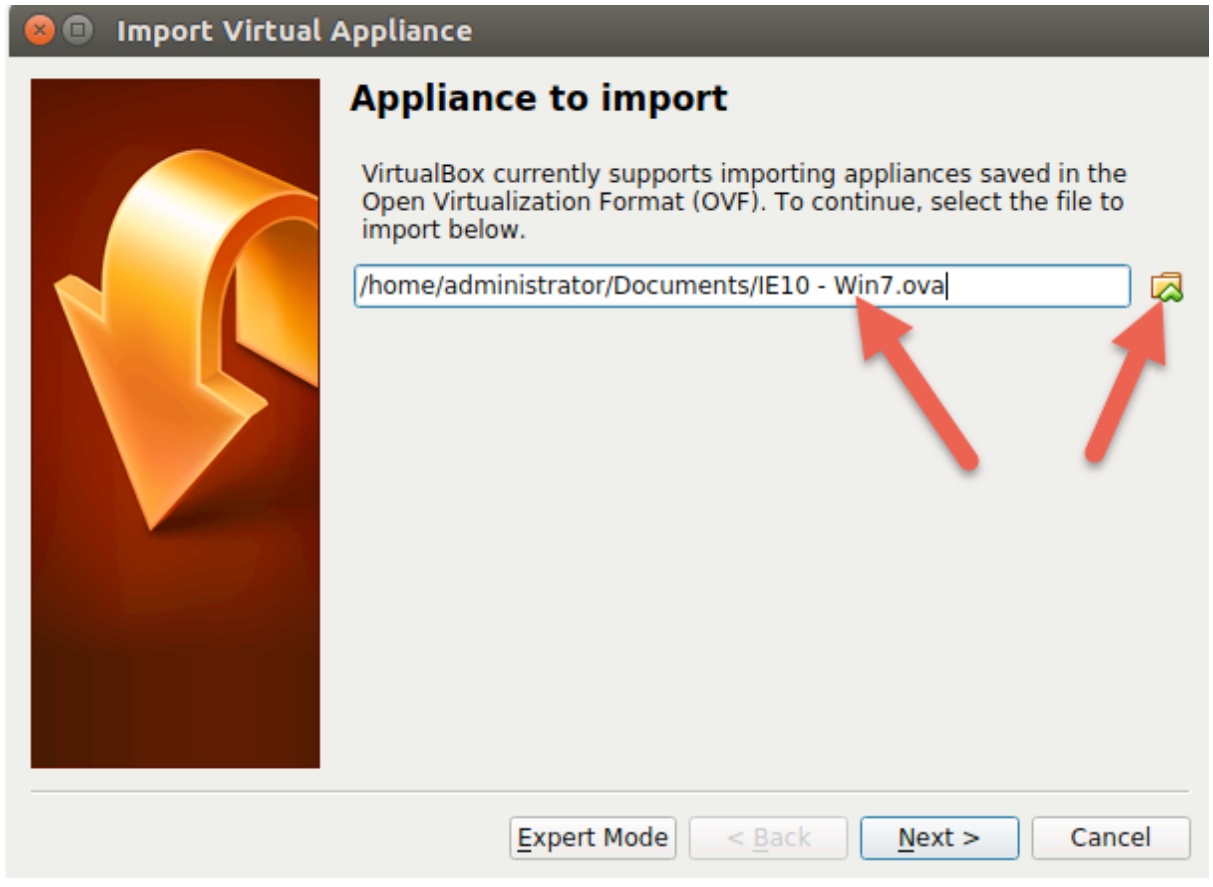
```
# Specify a comma-separated list of available machines to be used. For each
# specified ID you have to define a dedicated section containing the details
# on the respective machine. (E.g. cuckoo1,cuckoo2,cuckoo3)
machines = cuckoo1
```

If you are using the OVA file, follow the steps below to import it into VirtualBox.

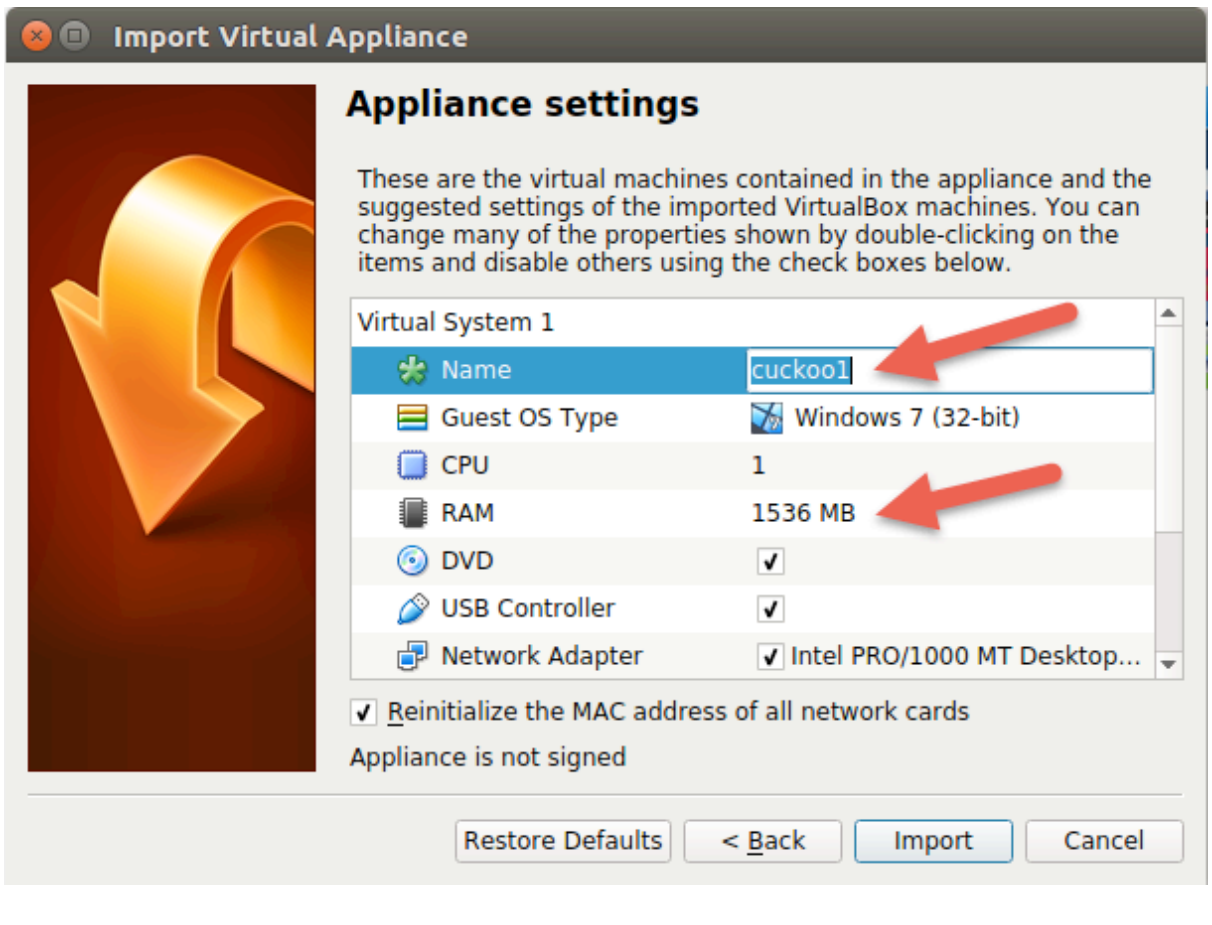
Open the File menu and select 'Import Appliance':



Select the OVA file and click Next:



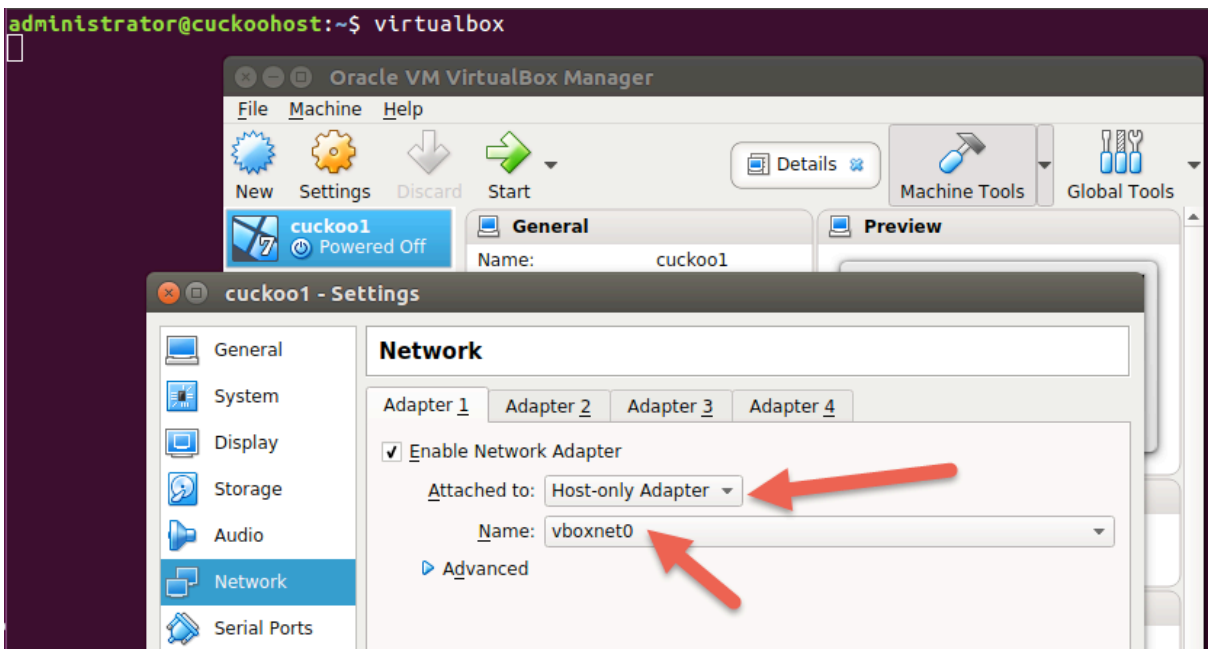
Double click the **Name** and change it to **cuckoo1** (this name is important!) and also change the RAM size to at least **1536** MB, then click Import:



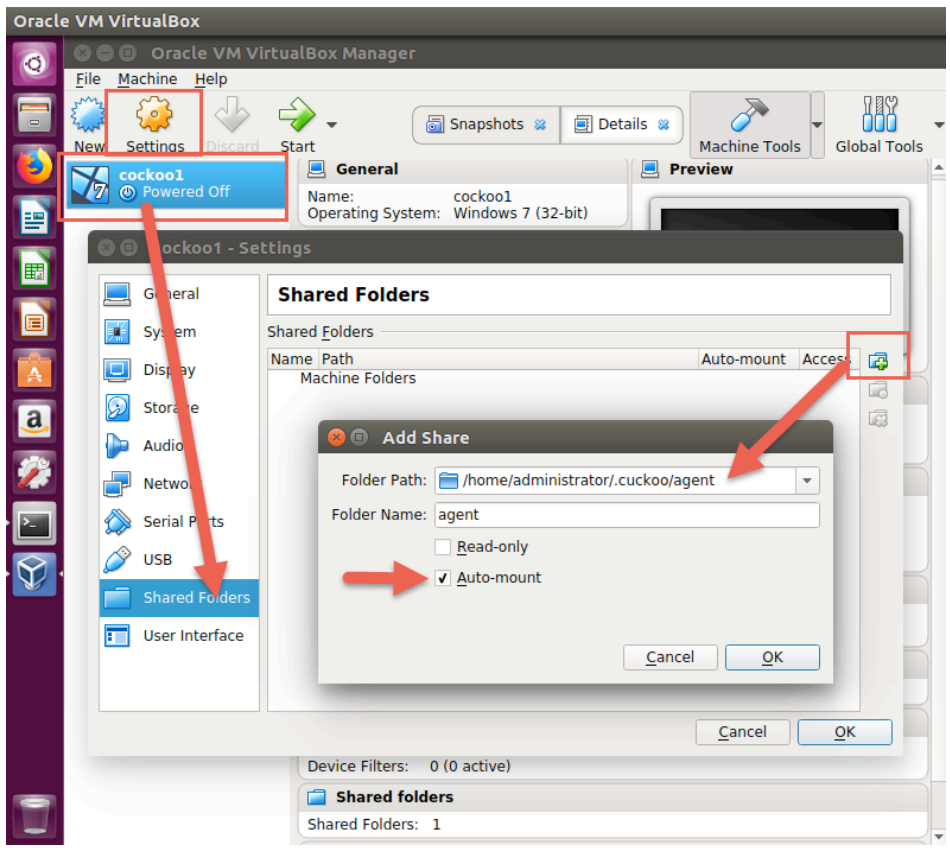
Let the import complete before you move on to the next step.

Regardless of your choice (using ISO image or OVA file), at this point, you should have a Windows 7 VM in VirtualBox ready for further configurations.

While the WindowsVM is OFF, open the "**Setting**" and change the network adapter to **Host-only** and **vboxnet0.**



Share the cuckoo agent (found in ~/.cuckoo/agent on Ubuntu machine) with this Windows GuestVM. Go to the **Shared Folders** setting on the right pane, and click on the **Add** (+) button. Note that you need to type the shared folder name directly into the folder name field, and check the **Auto-mount** checkbox:

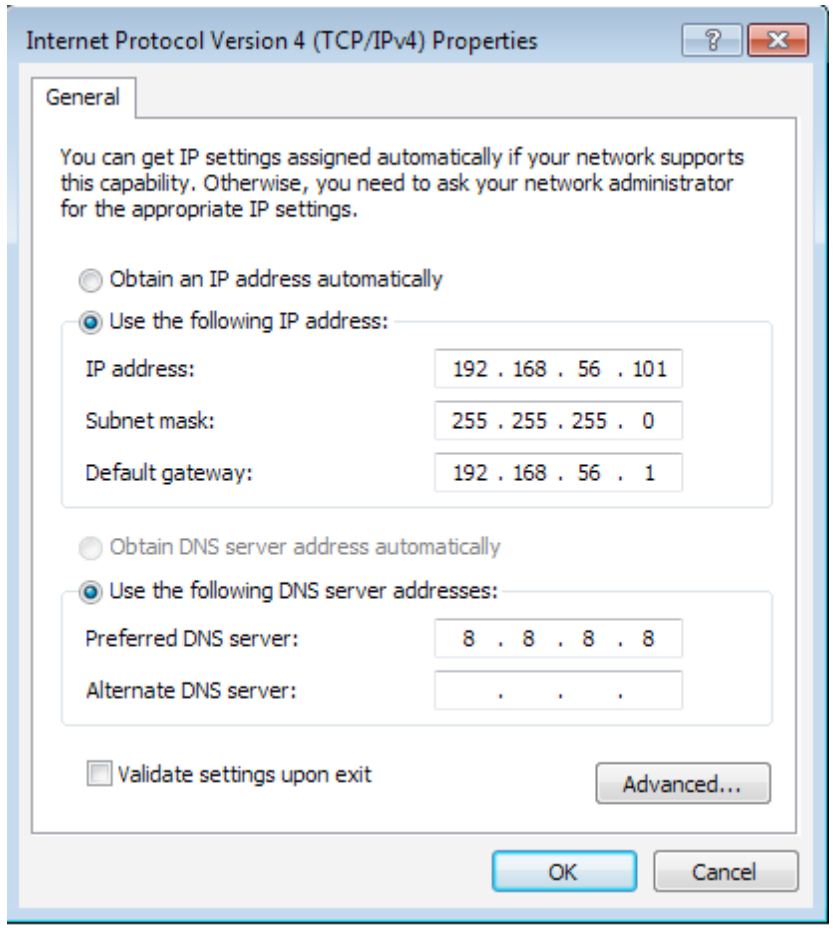


Once done, **start** the Windows 7 VM and edit the network settings for **IPv4** (also disable IPv6). NOTE: If you are using the ISO image, before starting the machine, remember to remove the ISO from the storage settings of VirtualBox or else it will take you through the whole setup process again.

Static IP: **192.168.56.101**

Default Gateway: **192.168.56.1**

DNS: **8.8.8.8** (or any publicly accessible DNS server)



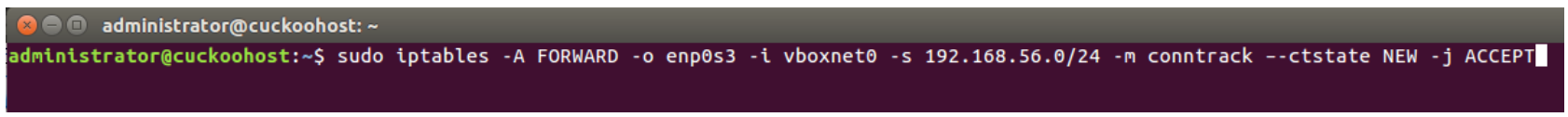
The IP address must be the same as the setting in the virtualbox.conf configuration file:

```
# Specify the IP address of the current virtual machine. Make sure that the
# IP address is valid and that the host machine is able to reach it. If not,
# the analysis will fail.
ip = 192.168.56.101
```

At this moment, the Windows 7 VM should only be able to communicate with the UbuntuVM, not to the Internet. However, the GuestVM also needs Internet access to be able to download some components into this Windows 7 VM as well as some analysis. We will need to modify the routing table and iptables rules in the UbuntuVM as the following.

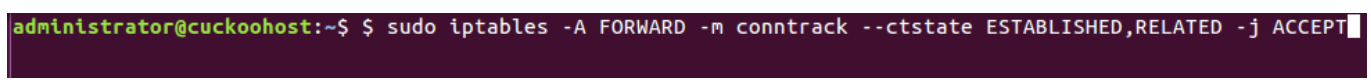
Open another Terminal in UbuntuVM and type the following commands.

```
$ sudo iptables -A FORWARD -o enp0s3 -i vboxnet0 -s 192.168.56.0/24 -m
conntrack --ctstate NEW -j ACCEPT
```

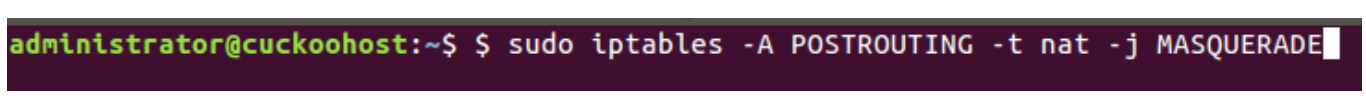


Then run the following two commands to complete the setting for the outbound firewall rules:

```
$ sudo iptables -A FORWARD -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
```




```
$ sudo iptables -A POSTROUTING -t nat -j MASQUERADE
```



Once the firewall has been modified, we need to route the traffic between the 2 interfaces.

Change to root user temporarily:

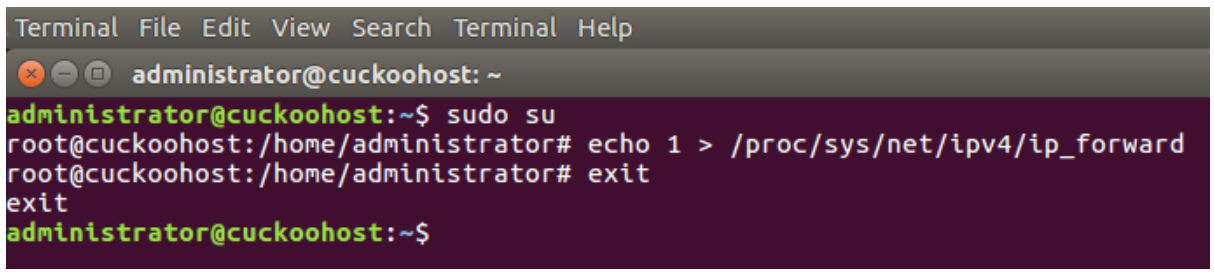
```
$ sudo su
```

Then use the following command to change the kernel routing:

```
$ echo 1 > /proc/sys/net/ipv4/ip_forward
```

Then go back to the normal user prompt:

```
# exit
```



Now go back to the GuestVM (Windows 7) and verify it has an internet connection.

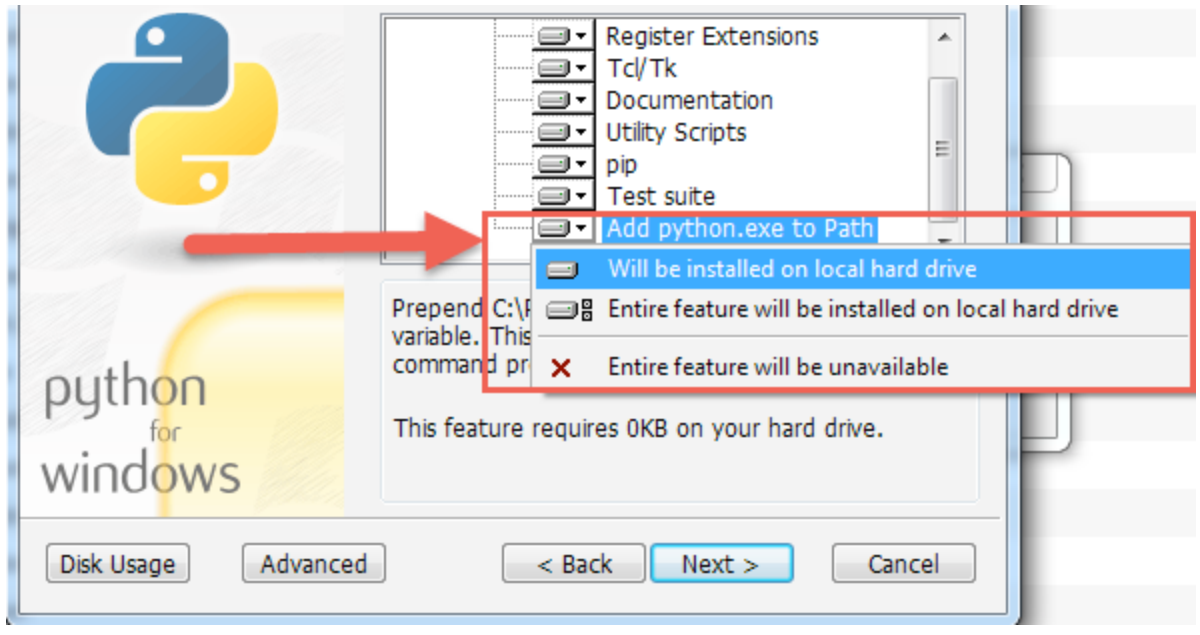
Open a command prompt window (cmd) and ping 8.8.8.8 and see if you have any reply.

Once you verified that the Windows GuestVM has Internet access, you might want to download and install Google Chrome web-browser. We need a newer browser to download Python 2.7.13.

Open Chrome and download Python 2.7.13 for Windows 7 32bit at this URL:

https://www.python.org/download/releases/2.7/

Once downloaded, install it and, during installation, check the option to include the PATH (important):

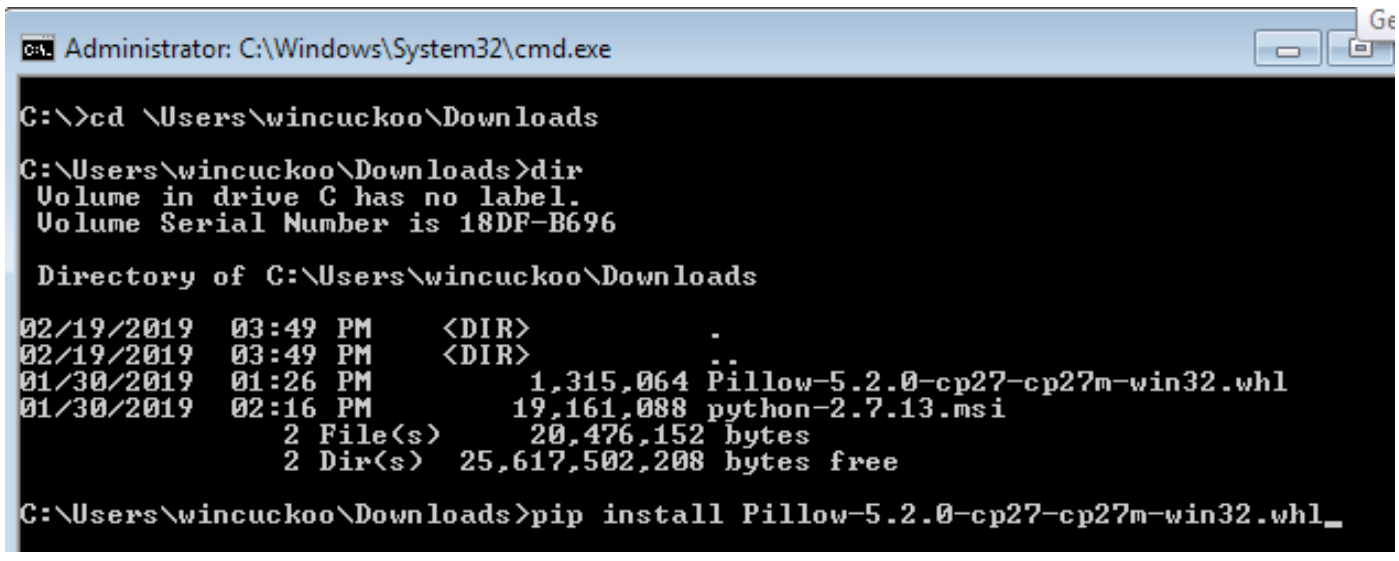


Next, download Python Pillow Pillow-5.2.0-cp27-cp27m-win32.whl (for python 2.7 and win32):

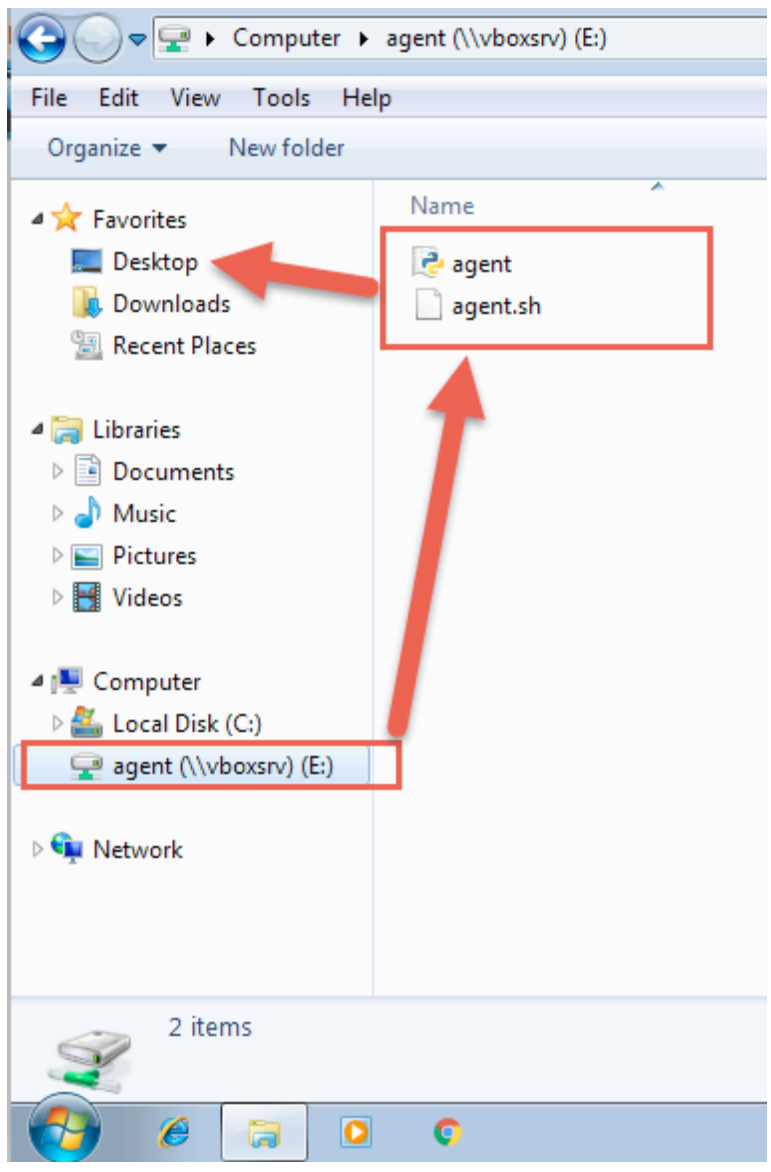
`https://pypi.org/project/Pillow/5.2.0/#files`

To install Pillow, first open a terminal console and navigate to where you downloaded the file. Then use the following command to install it:

```
pip install Pillow-5.2.0-cp27-cp27m-win32.whl
```



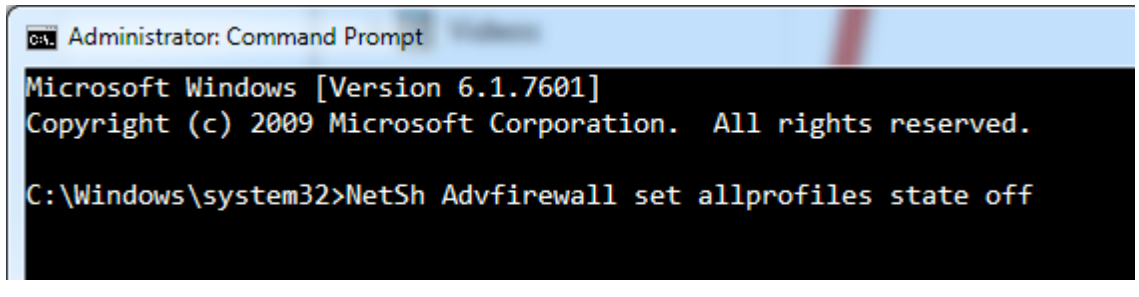
Next, you can install Virtualbox Guest Additions on the Windows Guest virtual machine (and reboot the VM if necessary). We now need to copy the Agents from the UbuntuVM. Open **File Explorer** and go to the shared folder and copy both **agent.py** and **agent.sh** files to the **Desktop** folder:



Disabling Windows Firewall

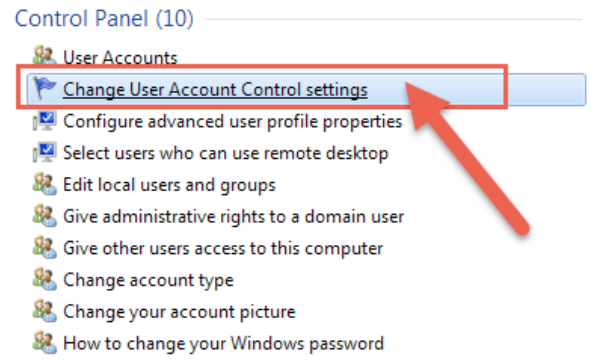
You now need to disable Windows Firewall. Open a cmd window and type the following:

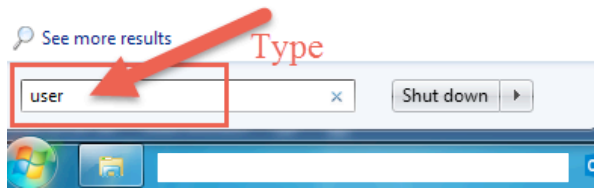
```
NetSh Advfirewall set allprofiles state off
```



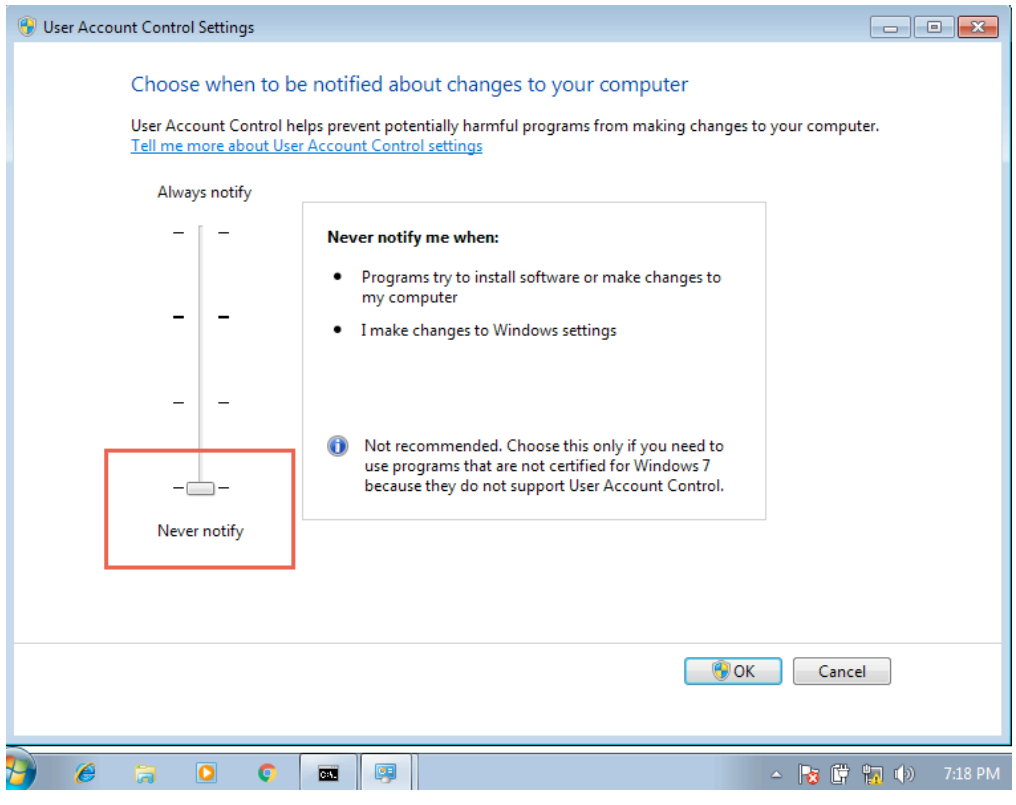
Disabling Windows UAC (User Account Control)

Now click the Start key and type in "user", then click "change user account control settings":



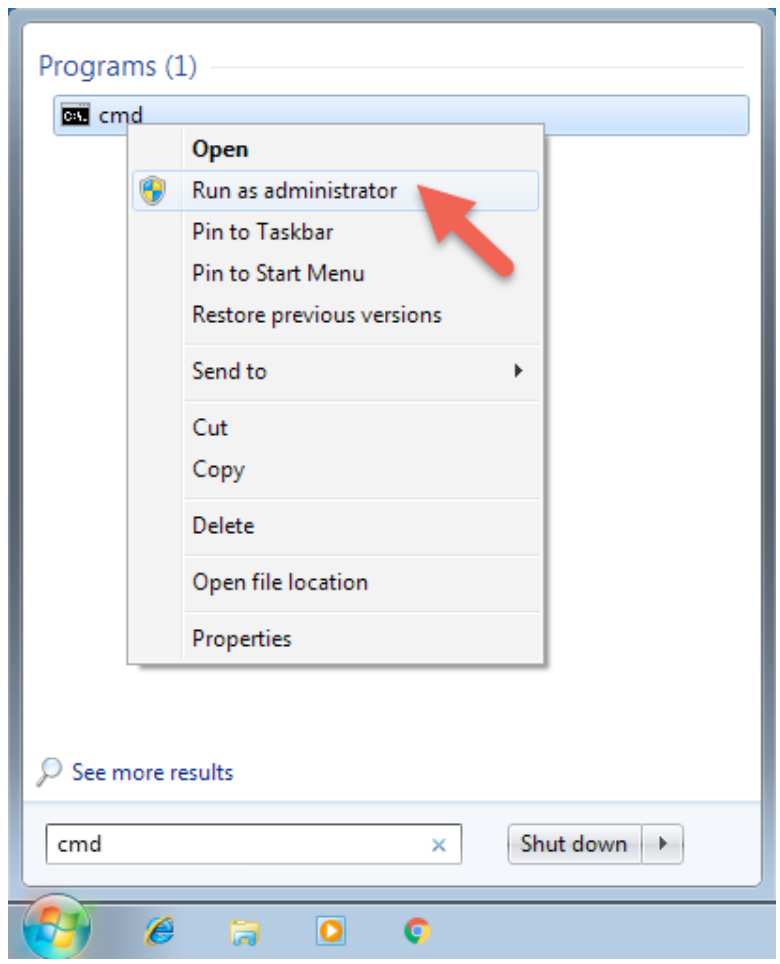


Drag the slider down to "never notify", and click OK:

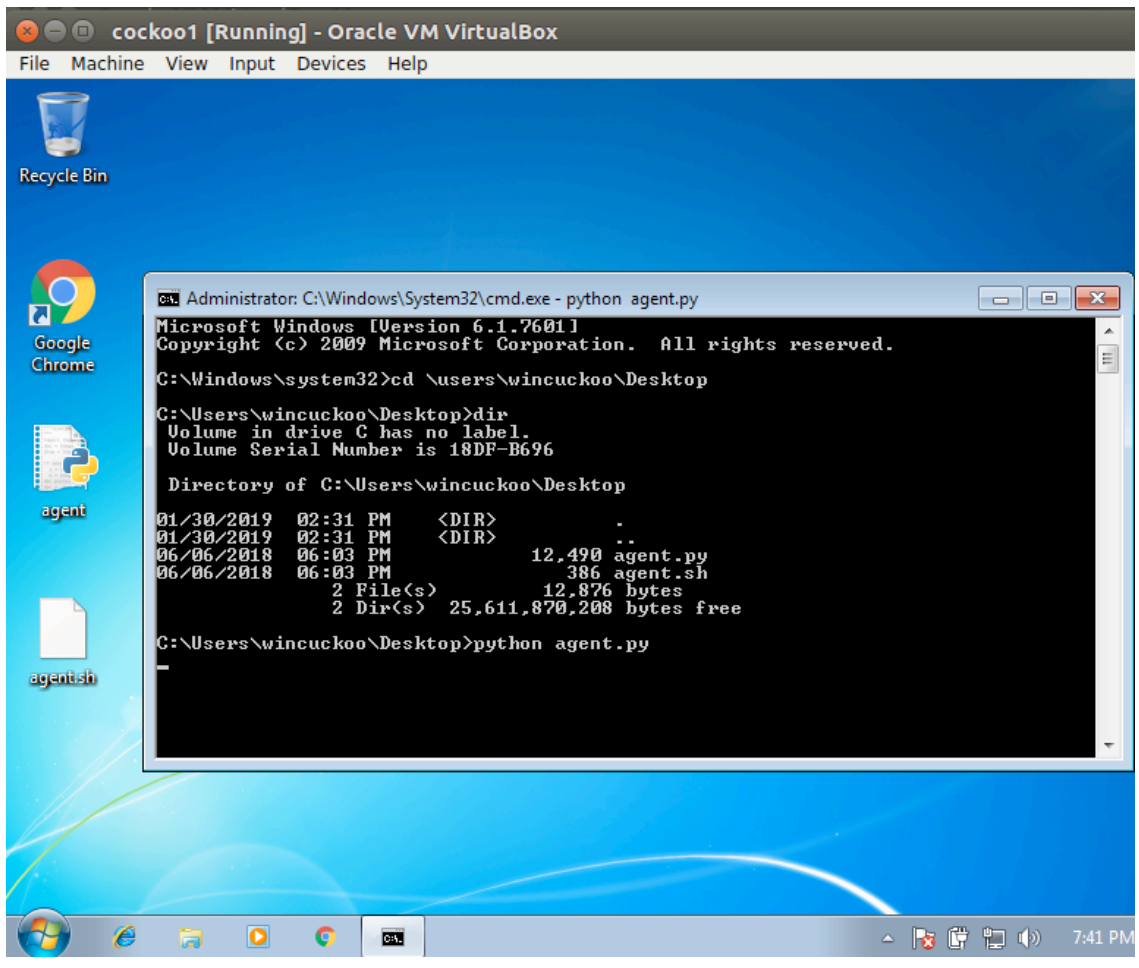


Running Cuckoo's Agent in Windows GuestVM

Once both the Firewall and UAC are off, open a command prompt window as Administrator:



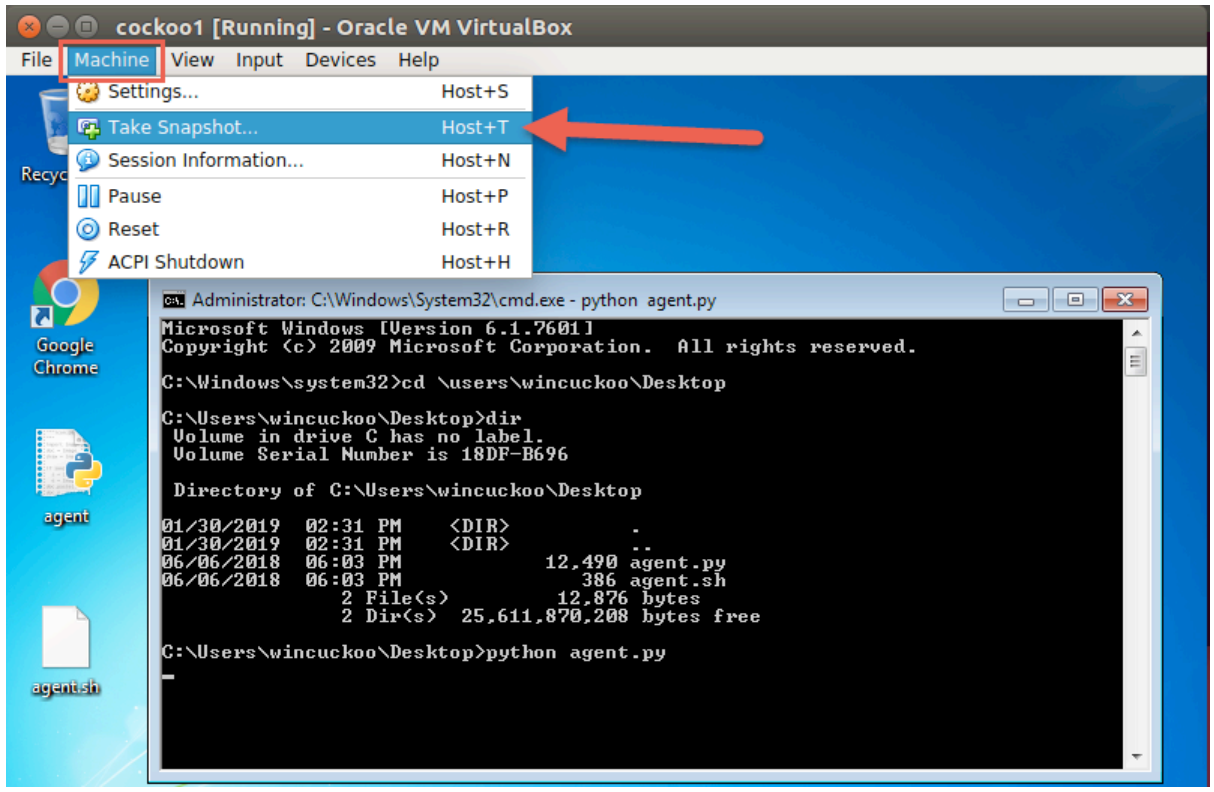
Now navigate to the folder where you copied the agent.py (Desktop) and run it:

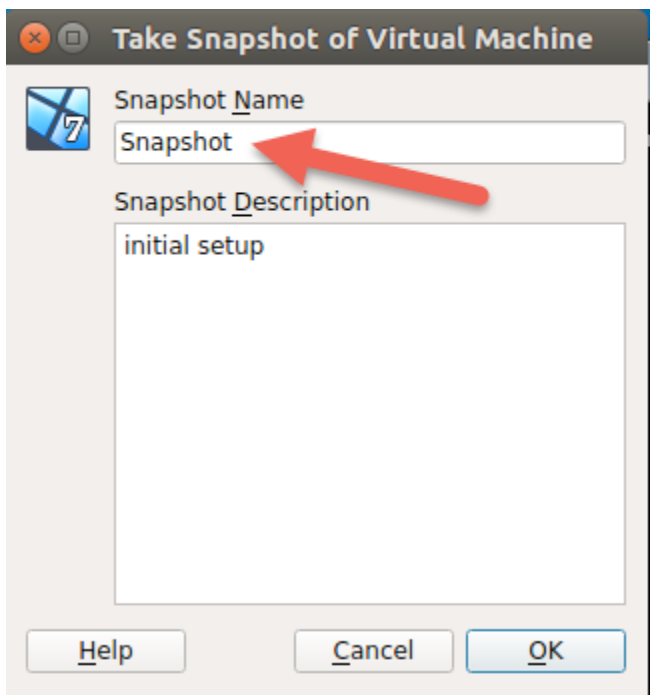


Taking a Snapshot of the Windows GuestVM

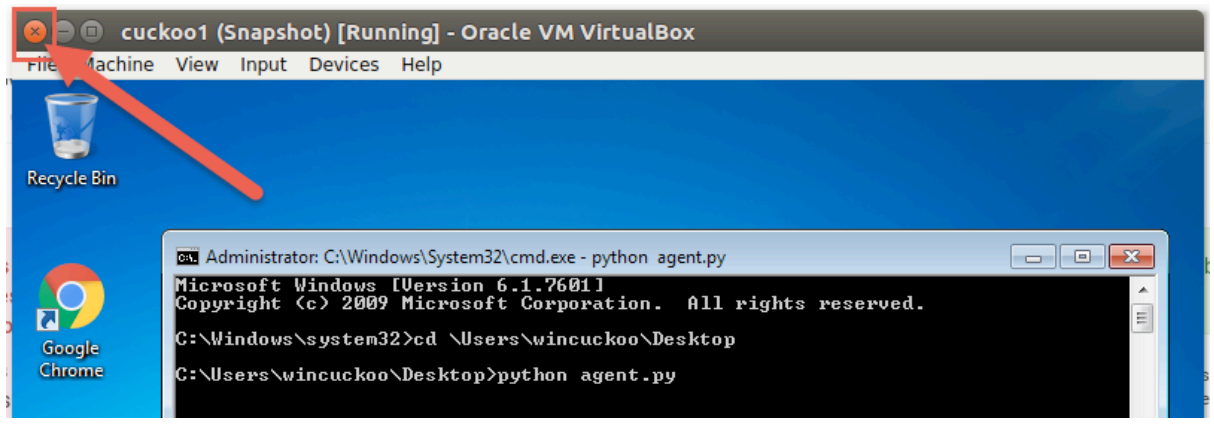
Once the agent.py is running (blinking cursor), this GuestVM is in a clean state and ready to receive and analyze any file from the Cuckoo Host machine. We ALWAYS need to come back to this clean state EVERY time we want to analyze any malware, so we must take a snapshot that we can always revert to in the future.

On the menu bar, click on the 'machine' and click 'take snapshot'. IMPORTANT NOTE: Name this file '**Snapshot**' NOT 'Snapshot 1':

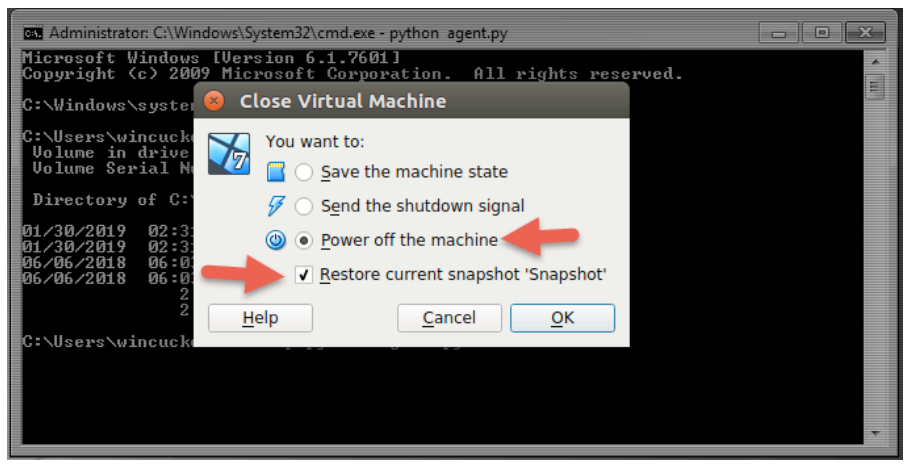




Once the snapshot has been taken, close (click the "x" on the menu bar) the Windows GuestVM but **DO NOT** close the agent:



Click 'Power off the machine' and remember to click "Restore current snapshot 'Snapshot'" before shutting it down:



That's it, setting up the environment is now completed. Next, we will test the whole Cuckoo setup to perform malware analysis.

Testing the Setup

In the Cuckoo Host (UbuntuVM), open 2 (**two**) different Terminal windows and on each Terminal window, navigate to Cuckoo's Working Directory (CWD):

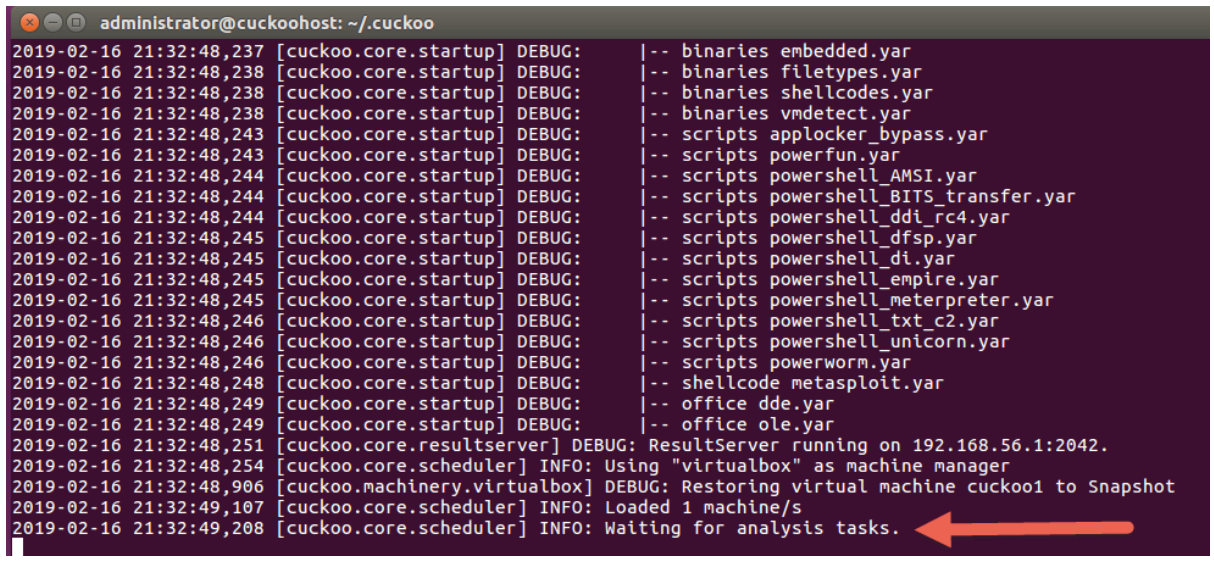
```
$ cd ~/.cuckoo
```

On the **first** terminal, run the main Cuckoo command:

```
$ cuckoo -d
```

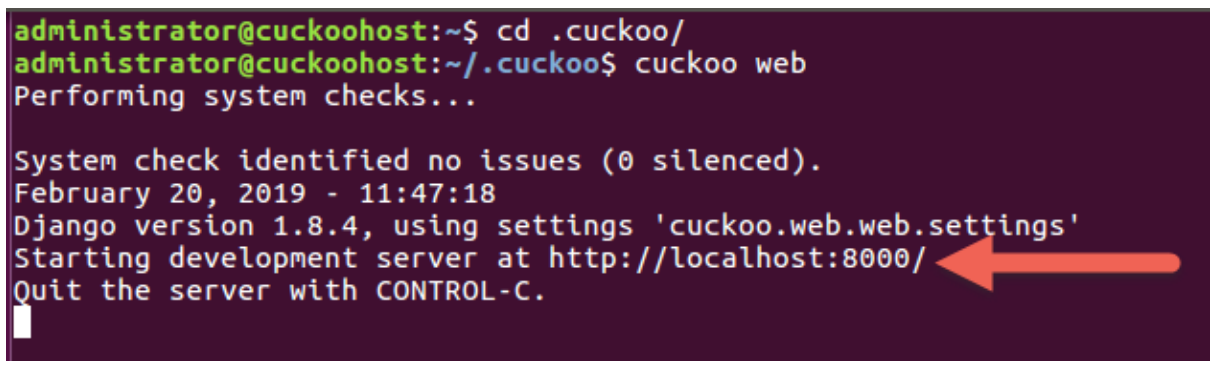
```
administrator@cuckoohost:~$ cd .cuckoo/
administrator@cuckoohost:~/.cuckoo$ cuckoo -d
```

This will start cuckoo, and wait till you see the line **INFO: Waiting for analysis tasks**.

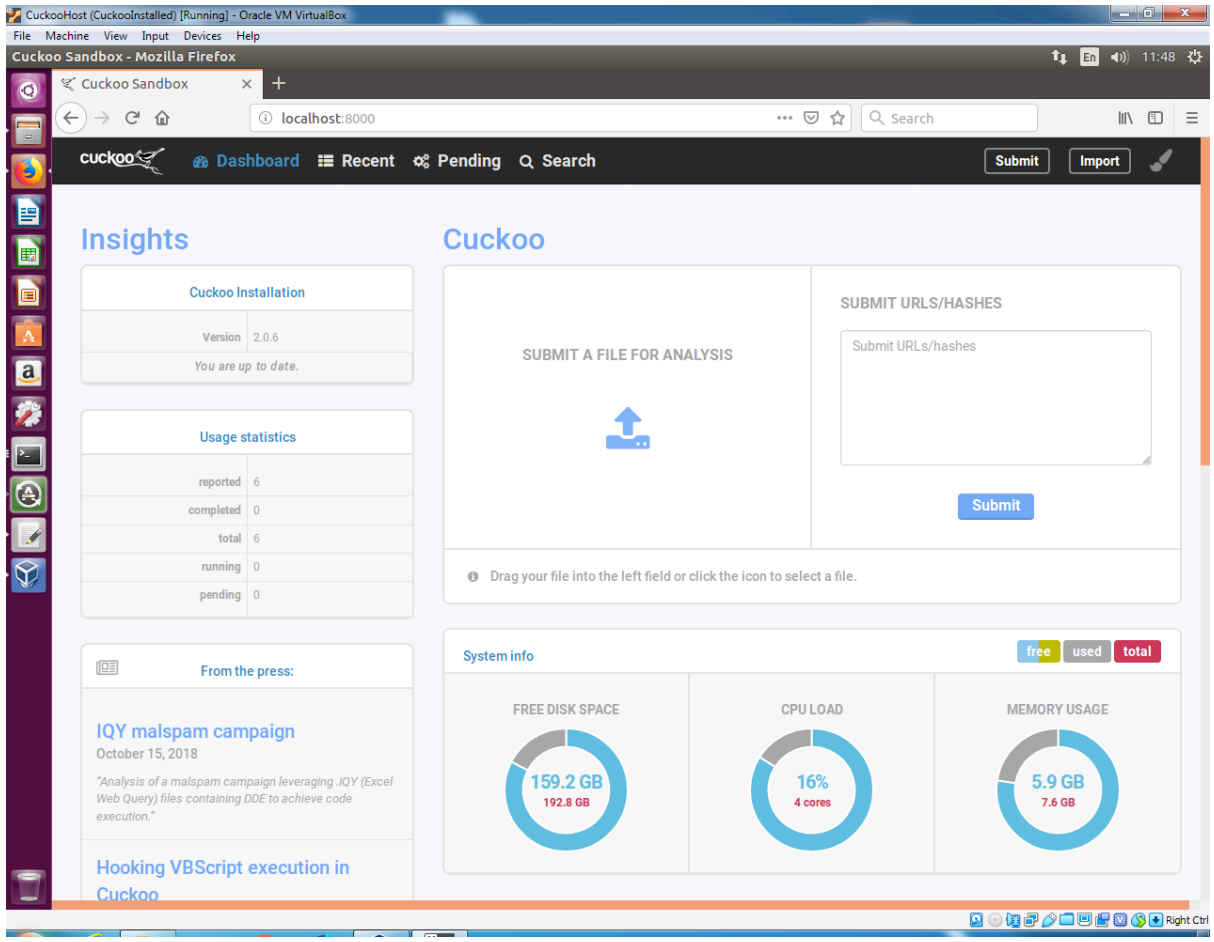


On the **second** terminal, run the following command:

```
$ cuckoo web
```

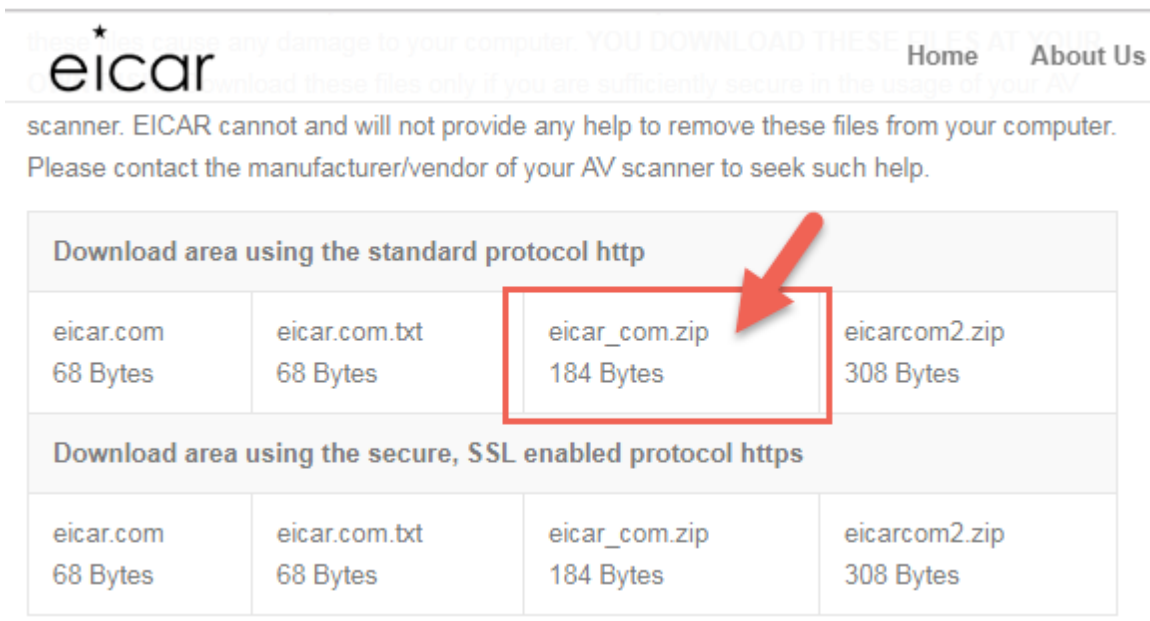


This will start Django and it will give you a link that you need to open, something like 'http://localhost: 8000/'. Open this link in a browser and you should see Cuckoo's web application:



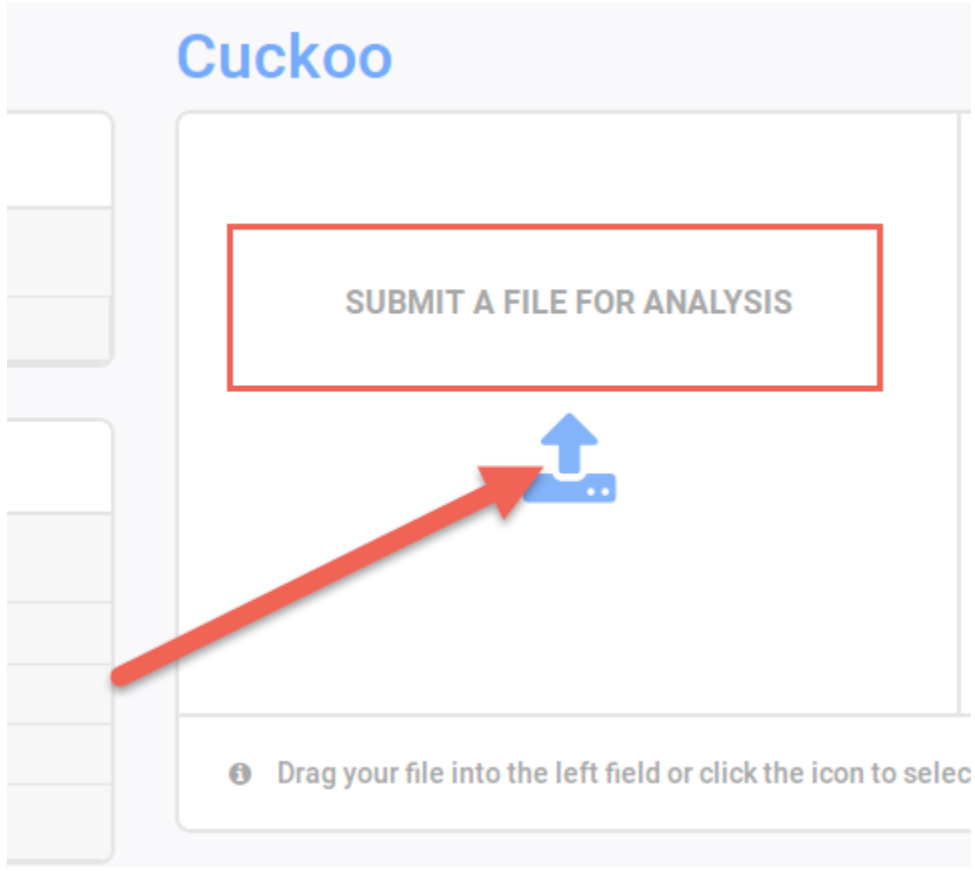
At this point, we can test and submit any file into the Web UI to analyze it. For testing purposes we are going to download a harmless malware Eicar.zip file from the following URL:

https://www.eicar.org/?page_id=3950

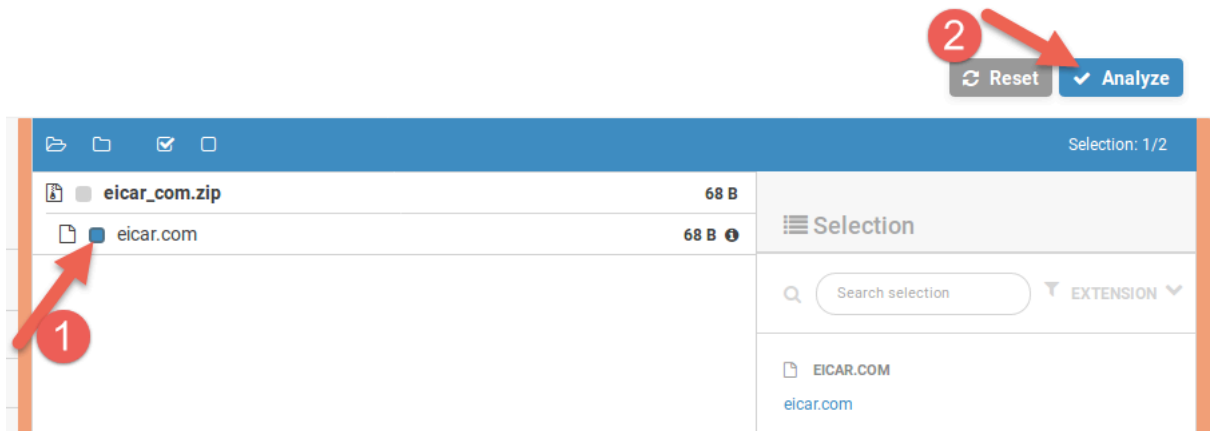


Download the file and save it somewhere in the UbuntuVM.

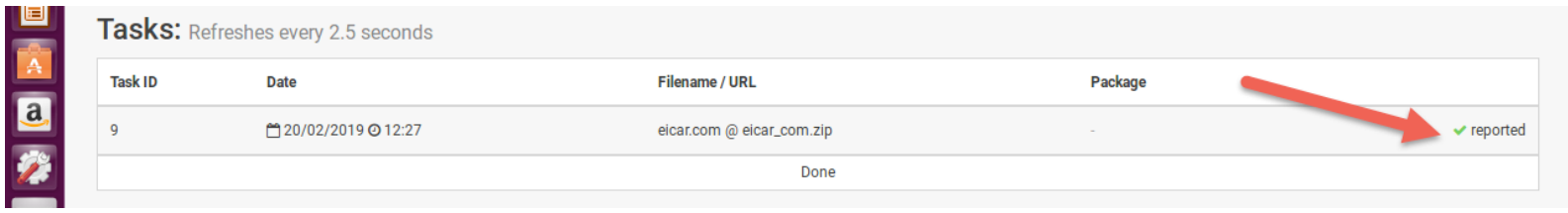
On Cuckoo Web GUI, click the Submit button:



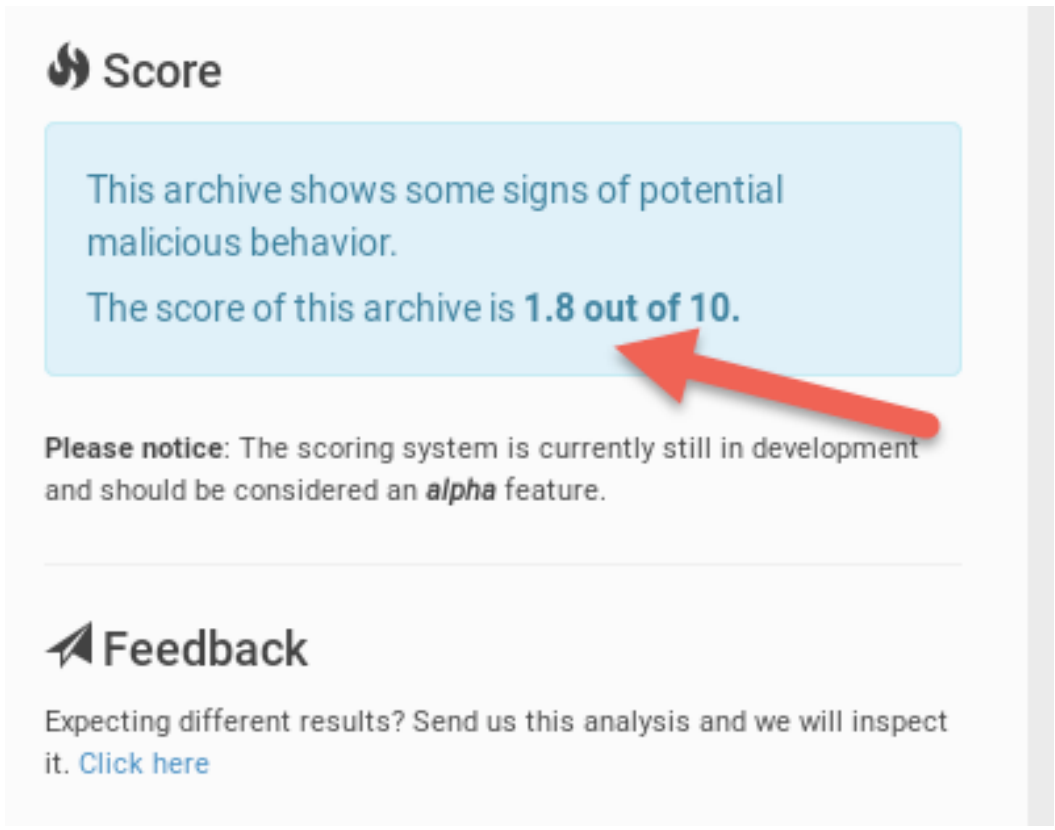
Once uploaded, select the File and click on the Analyze button:



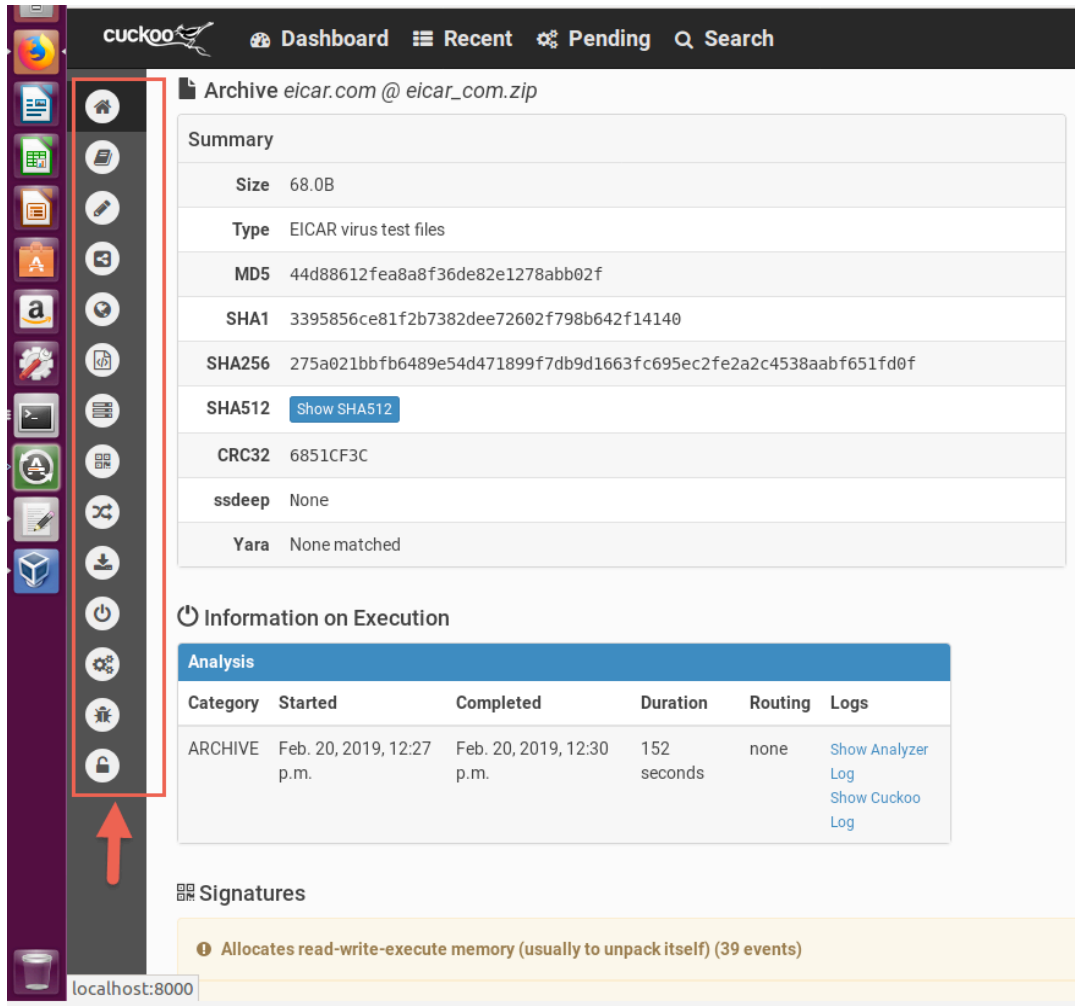
Cuckoo will start the analysis and it will take some time. Wait till it is done and you can double click the line to see the full report:



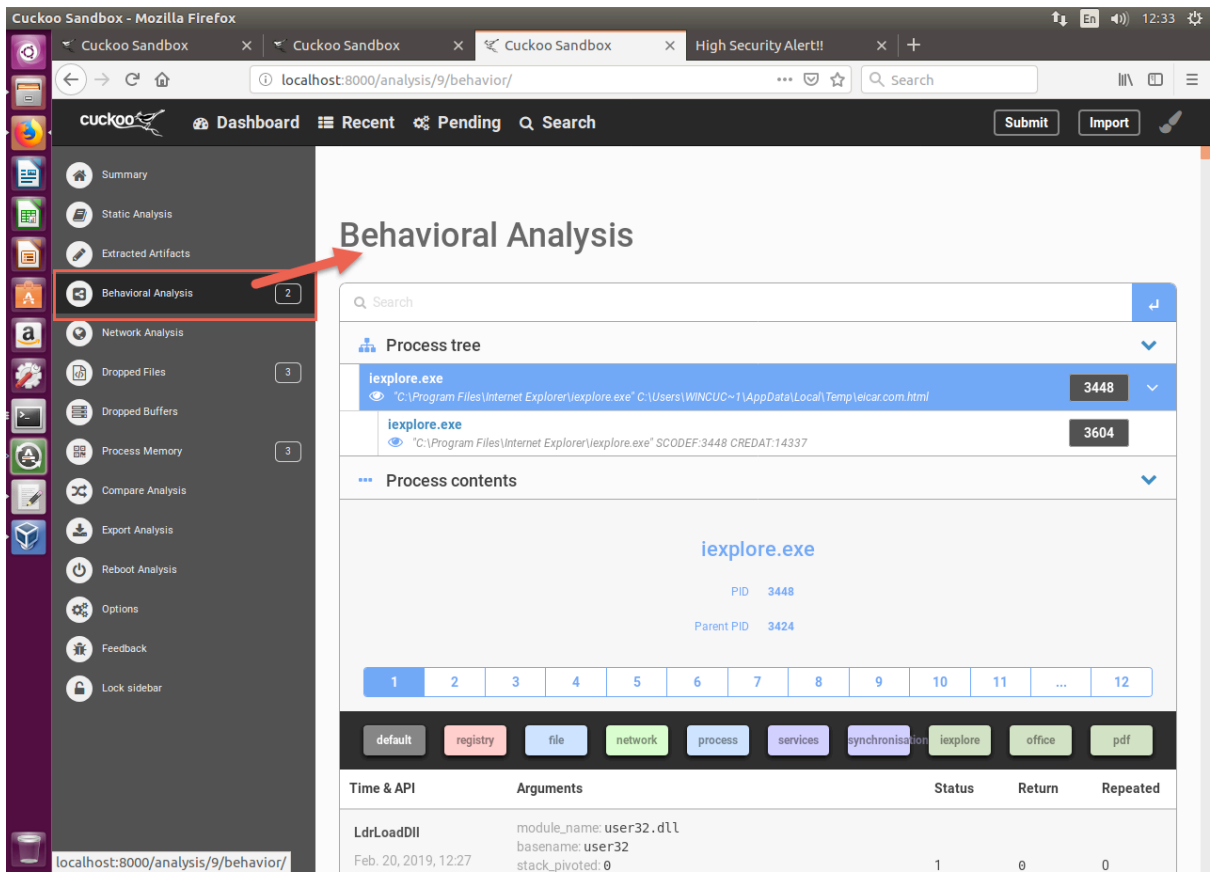
The report says that Cuckoo decides that the eicar_com.zip file has the Score of 1.8 of 10 (not really a malware):



The report has a comprehensive information about the file:



Clicking the Behavior Analysis icon gives something like the following:



We now have a fully working malware analysis Lab that is secure and isolated in a sandbox environment. The next step would be for you to start using it to conduct analysis for your own personal, or your organization's, purposes. We are just touching the surface of what Cuckoo Sandbox can do. It's now for you to explore it further to find out how deep the rabbit hole is!

Things to Try

Cuckoo Configurations

Check out the following pages regarding other Cuckoo Configuration Files that you might need to modify:

https://cuckoo.sh/docs/installation/host/configuration.html

Network Routing

Our setup is using Global Network Routing as the default route, which gives the same route (to and from) the Cuckoo Host (UbuntuVM) and the Windows GuestVM no matter how many analyses you are doing at any given time.

Since Cuckoo 2.0-rc1, it is possible to feature per-analysis network routing. In other words, if you have one VM and three samples to analyze, it is possible to deny internet access for the first analysis, route the second analysis through a VPN, and pull the third analysis through the Tor network. Check out this page for more information about Per-Analysis Network Routing[8]:

https://cuckoo.sh/docs/installation/host/routing.html

Internet Simulator

You might want to analyze and monitor all network traffic from and to the Windows GuestVM (including network to and from the Internet). You can have a network that will "simulate" the Internet by using INetSim. InetSim is a project that provides fake services for malware to talk to. In order to use InetSim routing, one will have to setup InetSim on the host machine (or in a separate VM) and configure Cuckoo so that it knows where to find the InetSim server[9]. Check out INetSim on the following page:

https://cuckoo.sh/docs/installation/host/routing.html#inetsim-routing

Final Words

I hope that you enjoyed this article and now that we have a working lab environment, in future articles, I will write more in depth about using Cuckoo Sandbox to do complete analysis of real malware. In the meantime, you can browse the Internet, download some malware from known malware repositories and study them, just be careful when you are working with real malware!

Warm Regards,

Januar Sugeng

About the Author

Januar is a security and network infrastructure technologist and currently a full time senior consultant/trainer in COMAT Training Center (Singapore) delivering classroom cybersecurity training programs. He has over 25 years of professional experience in consulting and training in the ICT industry and a proven track record of successfully designing and implementing cybersecurity systems for multinational corporations and government agencies in the region. His focus today include malware analysis, reverse engineering, memory forensics and threat intelligence.

[1] https://digiaware.com/2018/03/best-tools-for-malware-analysis/

[2] https://digiaware.com/2018/03/best-tools-for-malware-analysis/

[3] https://cuckoo.readthedocs.io/en/latest/installation/host/requirements/

[4] https://www.alienvault.com/blogs/security-essentials/explain-yara-rules-to-me

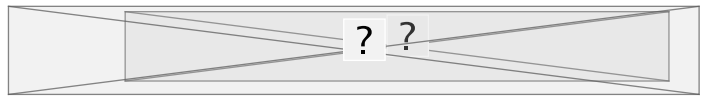
[5] https://mitmproxy.org/

[6] https://www.informationsecuritybuzz.com/articles/handy-tools-and-websites/

[7] https://pypi.org/project/M2Crypto/

[8] https://cuckoo.readthedocs.io/en/latest/installation/host/routing/

[9] https://cuckoo.sh/docs/installation/host/routing.html



Real-Time Network Intrusion Detection using SNORT

by Ummed Meel

A Network Intrusion Detection System configured with the updated set of rules can make the network secured against the intrusion attack. Through this article, one can have complete understanding and knowledge of deployment of the SNORT, which is an Open Source Network Detection System, with the real-time detection of an intruder in the network.

A network infrastructure that has several servers and devices is always a target of hackers to breach security to access confidential data. Any unauthorized activity from unknown servers or devices can cause a breach of security, which may lead to loss of confidential data. Network security can be breached as the network intrusion can become a threat through remote or physical access. Only an expert with a deep understanding of hacking and who knows how to prevent the network from the unwanted attacks through various channels can detect the intruders in the network and can make sure the next intrusion attack is prevented easily. A Network Intrusion Detection System configured with the

updated set of rules can make the network secured against the intrusion attack. Through this article, one can have complete understanding and knowledge of deployment of the SNORT, which is an Open Source Network Detection System, with the real-time detection of an intruder in the network.

The network is the third layer of the OSI model. When the data arrives at the network layer from the data link layer, it contains source and destination IP address. The network layer always plays an important role as it makes sure (It always make sure if data has reached its final destination or not. If so, this layer formats data into packets and deliver it to the transport layer. Otherwise, the network layer updates the destination address and pushes the frame back to lower layer.) by checking the delivery of the data by its state. A malicious request sent by the intruder always hits the network layer first of the target network. When the intruder tries to enumerate the network, it becomes more important as it detects and triggers the alerts per a predefined set of rules by admin.

Know about SNORT

SNORT is the Network Intrusion Detection and Prevention (IDS/IPS) System that is simply an open source tool developed in 1998 by Martin Roesch (a former founder and CTO of SourceFire). Cisco had acquired SourceFire in 2013 and has been developing SNORT since then. SNORT can detect real-time attacks and can protect the network infrastructure from a security breach. It can also detect intrusions from intruders on any website and servers as well. SNORT deeply analyzes the traffic on the network with a predefined set of rules.

SNORT can be easily deployed in the following phases.

Phase 01: Basic Installation

SNORT can be set up on multiple operating systems. We can set up SNORT on Windows, however, for now, we are going to set up SNORT on UBUNTU. Open a terminal in UBUNTU server and run the following command as shown in the image.

Command: `"sudo apt-get install snort"`

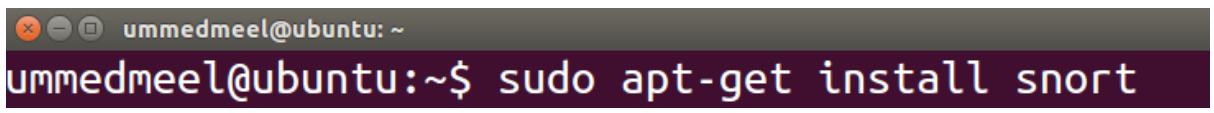


Fig 01: Install SNORT

Instructions: You would be asked to define the network range to configure SNORT in technical term CIDR for the Home_NET for all the rules defined by the user. To have a completed knowledge of the subnet mask mechanism of the system, you can use the following command.

Command: `"Ifconfig"`

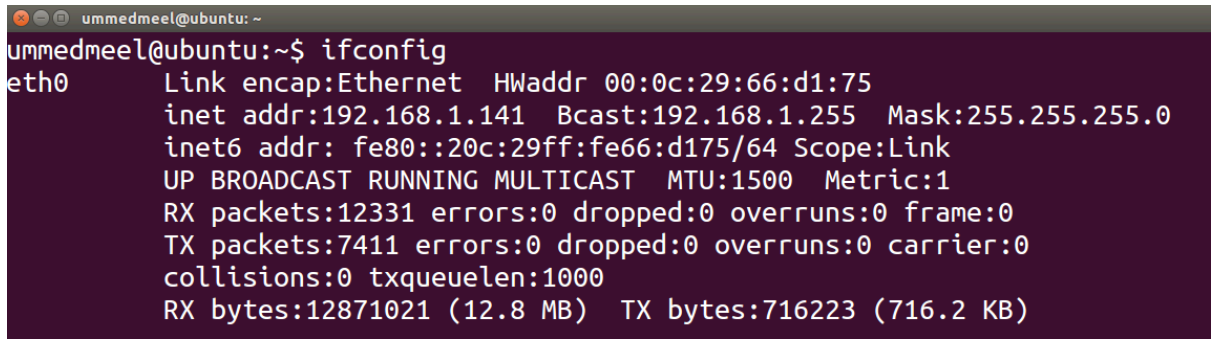


Fig 02: Ifconfig

We are using 255.255.255.0 subnet mask as shown in the snapshot, doing so we are declaring the network range as 192.168.1.0/24.

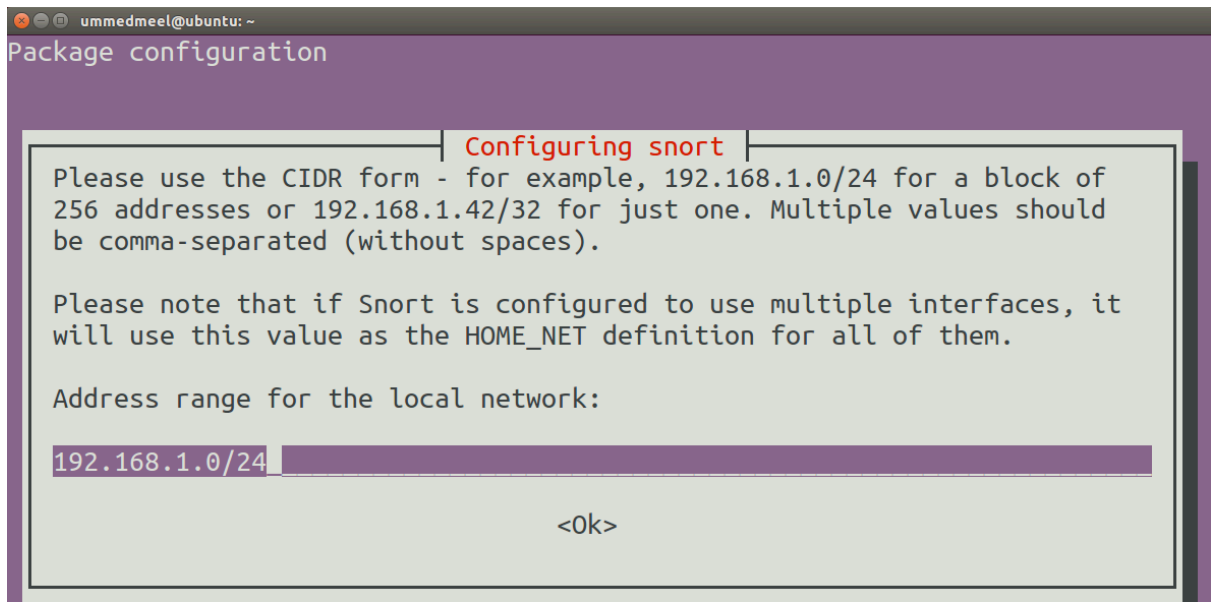


Fig 03: Subnet Mask

Now run the following command to validate the installation, it would check the entire SNORT configuration with predefined sequence during the installation and would display a successful installation message. Before running this command, you are required to create a log directory to save alert logs (mkdir.log).

Command: "`sudo snort -l ./log -b -c /etc/snort/snort.conf`"

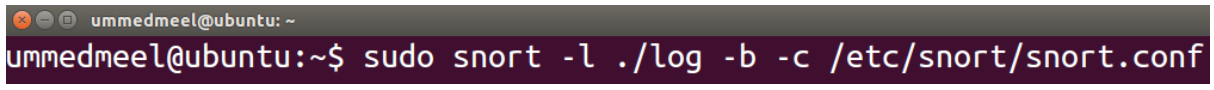


Fig 04: Installation Check

Phase 02: Customization of SNORT Rules

So far, we have learned how to install SNORT and going further we will learn how to customize SNORT with RULES.

Before customizing or setting up the rules on SNORT, an expert is expected to gather knowledge of updated cyber-attacks triggered recently on various platforms, like network infrastructure, web application, and servers, so that rules can be set in SNORT considering any type of threat, breach of security or cyber-attack. Run the following command to create a file designed with intrusion detection rules.

Command: "`sudo touch /etc/snort/rules/custom.rules`"

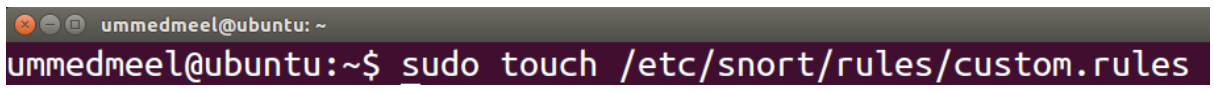


Fig 05: Custom rule file creation

Run the following command to make modification in SNORT configuration file.

Command: "`sudo nano /etc/snort/snort.conf`"

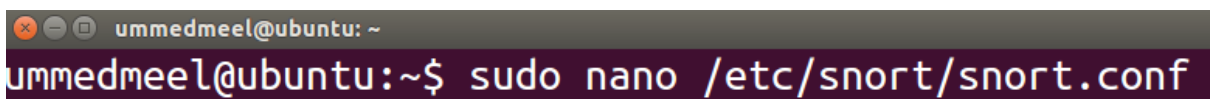


Fig 06: Configuration edit

Now in order to add the custom rules in SNORT, we need to include the newly created rule file into the SNORT configuration file. To save the configuration file, use “CTRL+X” followed by “Y” key and hit the enter key.

Add new line as ”`include $RULE_PATH/custom.rules`"

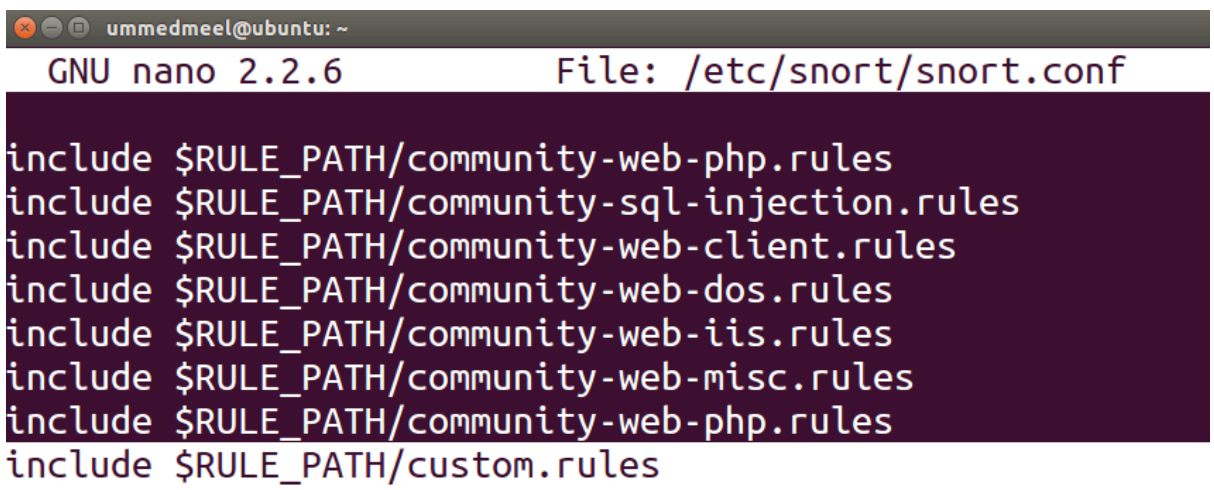


Fig 07: Add new rule file

Before adding a new rule, if you want to check and update the default set of rules in SNORT, you need to go to rules directory by using the following command.

Command: "`cd /etc/snort/rules`"

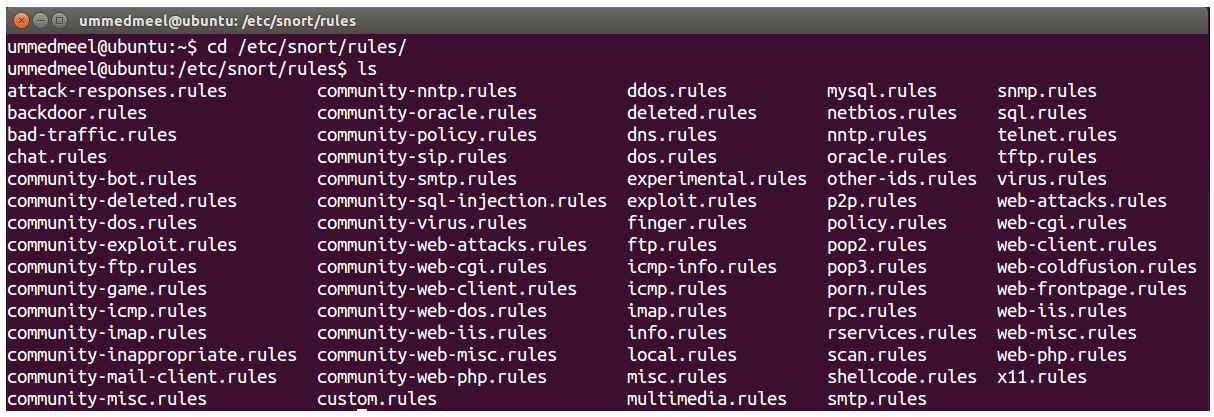


Fig 08: SNORT default rules

In order to customize the custom file with a new set of rules, you need to open the custom file "custom.rules". Run the following command.

Command: "`sudo nano etc/snort/rules/custom.rules`"

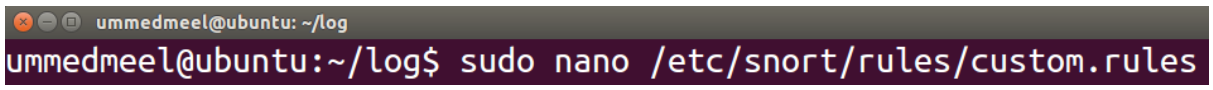


Fig 09: Edit custom rule file

One can add a new set of rules according to the latest vulnerability identified. Here we are trying with a simple rule to detect the flow of large ICMP packets in the network. In custom rule, as mentioned in the snapshot first, "any any" where first "any" denotes the attacker's IP address and second "any" denotes the port number of the attacker respectively. Similarly, the second "any any" after the arrow where first "any" denotes the victim's IP address and second "any" denotes the victim's port number respectively. A user can also add a customized alert message for clear understanding during the real-time monitoring. Expert or Tech also needs to assign the unique identification numbers (SID) to the app package and size of ICMP packets.

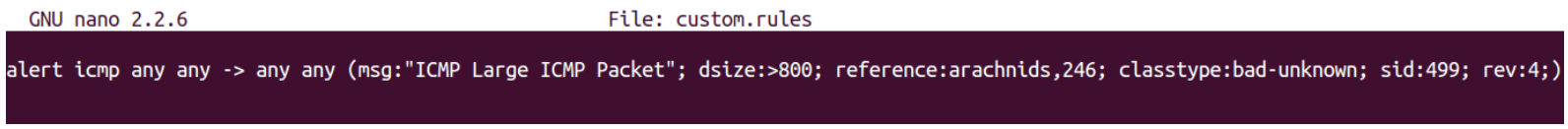


Fig10: Add new rule

Phase 03: Intrusion Log Storage

So far, we have learned the deployment and customization of rules of SNORT, going forward we will learn how logs get stored in SNORT.

Every time it is not possible to monitor the traffic on the network on a real-time basis to detect the cyber intrusion, that's why experts need to create a folder to save logs of activities that take place during an attack in SNORT. Run “mkdir logs” command to create a directory.

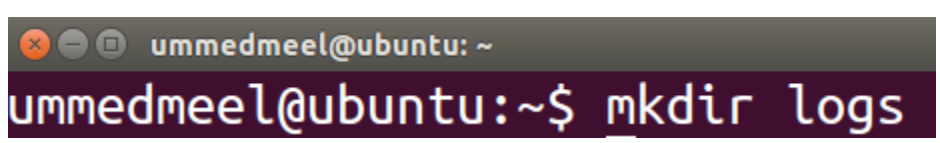


Fig 11: Create log directory

Move to the log folder and check that SNORT has started saving alert logs successfully.

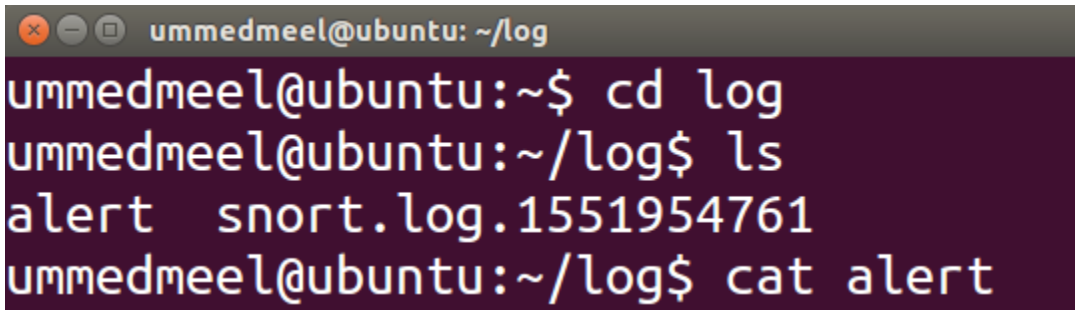


Fig 12: Check log directory

For example, if an intruder tried to scan the network or server using NMAP or other network mapping tools, SNORT will save this as an alert to the logs folder.

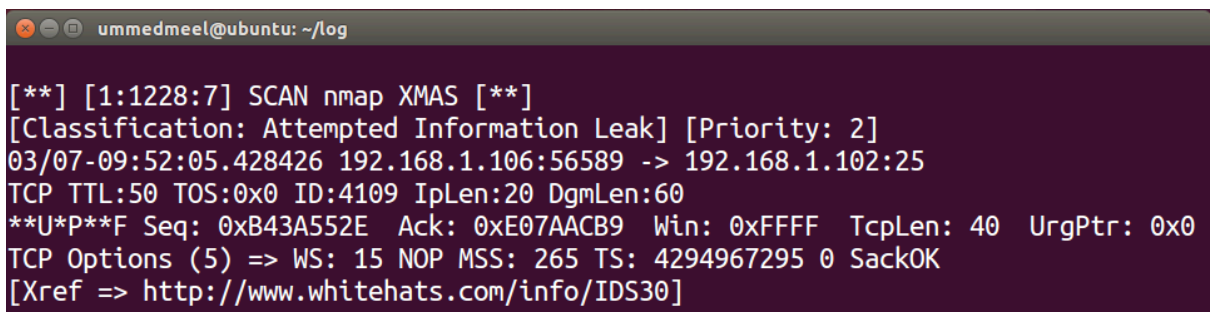


Fig 13: NMAP alert message

Similarly, if any user tries to attack the server or network with DDoS that has SNORT configured, it would be detected and the same activity would be logged with an alert message in the log file as shown below.

Fig 14: DDoS alert message

Phase 04: Attack using vulnerability scanner tool (SPARTA)

Now we are done with the deployment, customization and setting path for logs in SNORT. Going forward we will test SNORT potential with a test attack using a tool called SPARTA.

Sparta is a vulnerability scanning tool that scans the open ports and runs the service of the target server or network completely. We would be initiating a network scan and vulnerability finder attack on the target server. Later on, Nikto scan open ports discovered by network mapping tool again for the vulnerability on the running applications. During the network mapping and vulnerability scanning, SPARTA is going to hit the target server actively with tons of malicious requests.

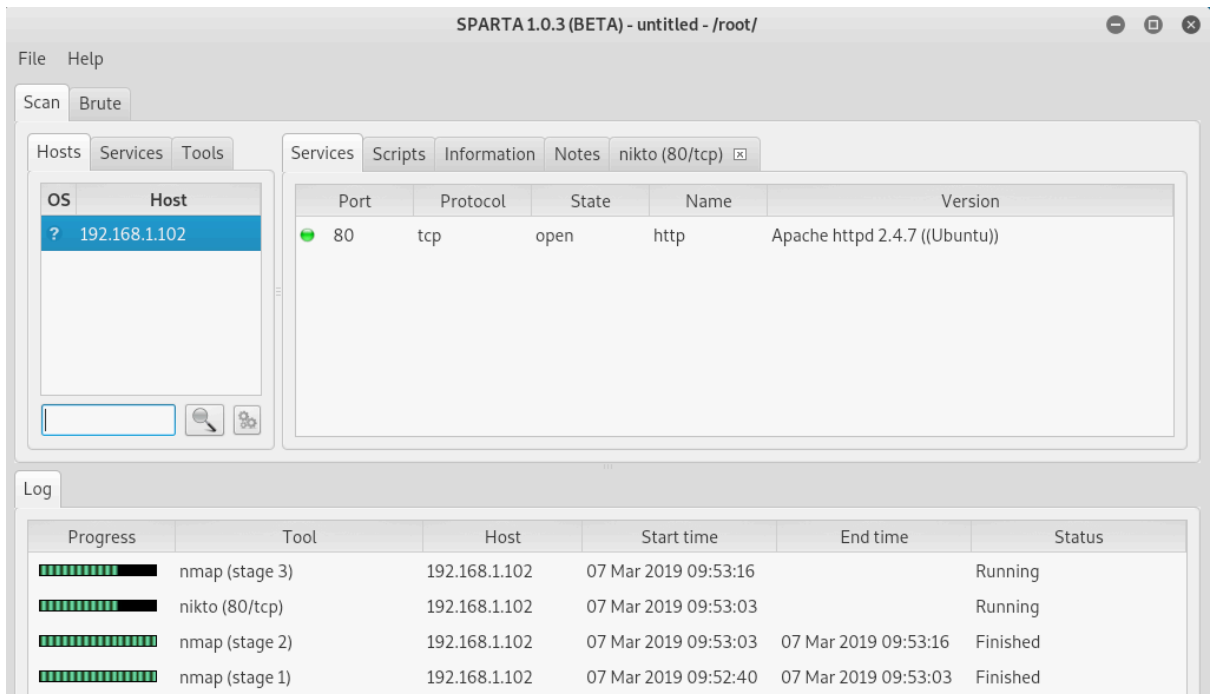


Fig 15: SPARTA scanning tool

Phase 05: Live Logs

During the test attack on SNORT configured network, we can monitor the live logs. Previously, we created a log folder that saves all the possible alerts detected in SNORT. In order to monitor the real-time traffic on a network configured with SNORT, please run the following command.

Command: "`Sudo snort -A console -q -u snort -c /etc/snort/snort.conf -i eth0`"

Note: In the above-mentioned command "eth0" defines the adapter of the physical machine through which we are intercepting network traffic.

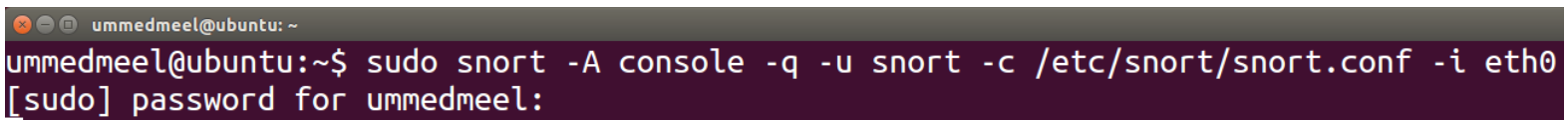


Fig 16: Real-time log monitoring command

After running the previous command, the network admin or user needs to keep his eyes on the terminal continuously. Every malicious request sent by the intruder could be published here real time with the alert message and attack category. Network traffic intercepted at the ethernet adapter first goes through the alert rules defined by the SNORT and user, after that, if any malicious request or scanning attack request is found, then SNORT displays an alert message live on the terminal and also saves a copy for the user for future reference.

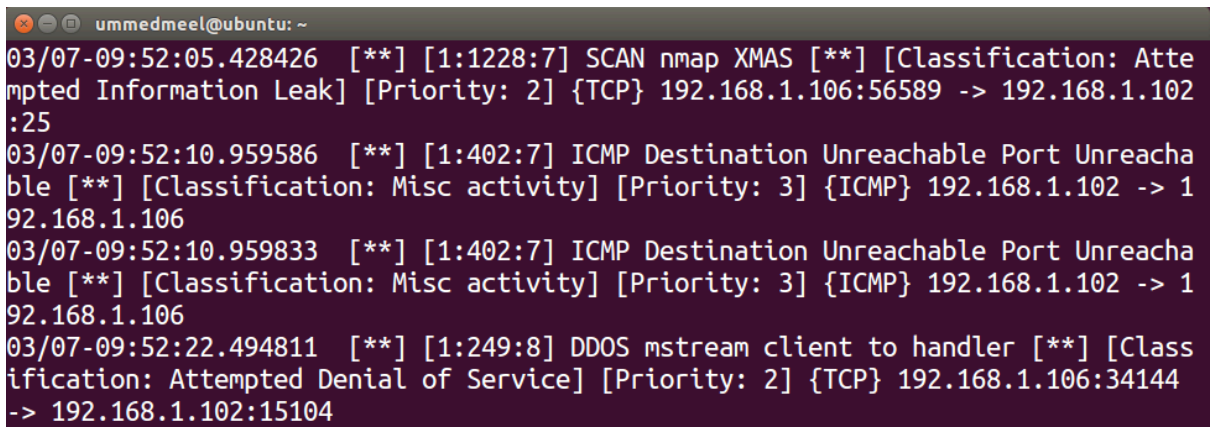


Fig 17: Real-time attack log

After real-time detection of cyber intrusion, the user can prevent the intruder from getting into the network. In the case of a DDoS attack on the network, the admin can directly block the attacker's IP address. Now we can see clearly that SNORT has detected a real-time intrusion attack from 192.168.1.106 IP to the 192.168.1.102 IP address and displayed a message as "Attempt to Information Leak".

Conclusion:

So far we should have the complete knowledge of SNORT and its requirement for the security of network infrastructure, Web-application, and Servers. SNORT, which is an open source software, becomes very essential for security purposes as it's easy to configure and can be used on multiple

operating systems with few sets of commands. A tech with knowledge of hacking can design the set of rules and protect the infrastructure of any organization using SNORT.

About the Author

Designation: Cyber Expert (Police and Defence Trainer)

Education: B Tech (ECE), CEH (EC-Council), Diploma in Cyber Law, LLB (Pursuing)

Ummed is closely associated with Indian Police, Air Force, BSF and higher defence authorities in India for the last 5+ years. Ummed has conducted 100+trainings and workshops for Indian Police of Delhi, Uttar Pradesh, Rajasthan, Haryana and Himachal police, etc., and trained more than 8,000 police and other LEA's officers. He has 5+ years of experience in Cyber Security, Vulnerability Assessment and Penetration Testing, Cybercrime investigation, Digital Evidence seizure and Digital Forensics. He has been interviewed by several news channels, newspapers and magazines for Cyber Crime Safety and forensics. He has also conducted 50+seminars for government/non-government organizations, school, colleges and other law enforcement agencies on Cybercrime awareness.

Contact: +91 8010 6363 59

Email: ummedmeel@gmail.com

Website: www.ummedcyber.com

The First Responder CSIRT on a SECURE Drive

by John Walker

Looking back on the problematic events relating to security matters, and others born out of procurement and licensing issues I encountered when working within areas of South America and sub-continent, I arrived at the firm conclusion that to get the job done, it was of obvious and paramount importance to carry the entire *secure* CSIRT (Computer Security Incident Response Team) Toolkit to the locations to be attended – for this is the only way the attending professional (me) could be sure they had arrived fully equipped with the necessary tools, support materials and facilities to coordinate and effect the operation in hand, at the time of responding.

No matter the title, *First Responder*, *CSIRT Investigator*, *Digital Forensics Practitioner*, be they *internal*, *external*, *independent* or *consultant*, such a task can be challenging, and dictate not only a need to possess a wealth of deep knowledge and techniques, but also to ensure that the necessary tools are *always* available at the scene-of-event/crime/incident, and available for use as and when they *are* required – in particular, when the concerned professional is attending an event away from their home-base, or at a global location of a domicile where it may prove *difficult* or *impossible* to acquire the required materials for a host of reasons - these range from legislation through to country specific regulations and prohibitions ***[1].*** It may also be that the logical and physical environments the attending professional is accommodated with do not meet the exacting and robust needs of the task for any one of many reasons:

1. Physical Security and Storage may not be adequate
2. Logical Security and safeguarding of assets may be of concern – Trust issues
3. Until facts are fully established (Friends or Foe) potentials for Case Contamination and Compromise could exist
4. The requirement of Case Segregation proves to be a challenge – potentially exposing the investigation and outcome
5. Communications Security may not be trusted

As a testament to the aforementioned areas, in my own personal experience, they have all come into play in two *real-life* cases. I can recall an example in which one such case brought into play points **1**, **2**, and **3,** with the second attracting all but point **1**, with a heavy emphasis of point **3**, making both activities extremely difficult to manage whilst under investigation.

The Solution – CSIRT Toolkit on a Go-Disk

Looking back on the problematic events relating to security matters, and others born out of procurement and licensing issues I encountered when working within areas of South America and sub-continent, I arrived at the firm conclusion that to get the job done, it was of obvious and paramount importance to carry the entire *secure* CSIRT (Computer Security Incident Response Team) Toolkit to the locations to be attended – for this is the only way the attending professional (me) could be sure they had arrived fully equipped with the necessary tools, support materials and facilities to coordinate and effect the operation in hand, at the time of responding. Granted, these could be stored on a working laptop, but that brings with it several concerns, potential security issues, and exposures which could impact or compromise the acquired materials and artifacts – for example:

1. The laptop does not provision a fully secured physical environment to assure segregation of sensitive materials and artifacts
2. Laptops are common-garden tools and used for multiple activities – e.g. Email
3. Not all laptops are provisioned with High-Grade Encryption (technological support that most practitioners recognise the value of when storing and transporting evidence)

4. Laptops have a higher exposure to compromise as they are subject to external interfaces

Looking at the aforementioned areas, one could still argue that a dedicated laptop could be used in place of a secure segregated drive – but taking into account the cost, and that such a system will require updating from external, internet located sources, not forgetting that any materials written to a laptop are not physically transportable between systems, and agnostic to the O/S, unlike those which are stored on a segregated drive. In my case, I have discounted the laptop as not meeting my basic requirements as specified standards as outlined above.

At this stage, I am also considering the types of cases, and seeking the required burden of proof I may need to exercise to demonstrate to a *court of law*, *tribunal*, or to satisfy a *legal challenge*, that my acquired artifacts are *as was* at time of acquisition, and that they are *robust* in their profile and overall *integrity*. Granted, I am somewhat paranoid over such matters, but then I have been in attendance when a prime piece of evidence could not be tracked down on the day of a court case, which resulted in a *robust*, and *successful* challenge against the *integrity* of the artifact. So here I am also considering storing my acquired *best evidence* onto a secured medium that will be carried under separate cover, or even by another party (with no logical access). At this point, I can imagine comments – 'what about the cloud?' Granted that this is an option, but for myself this is the very last fall back – it does *not* tick all my boxes for obvious reasons.

The Shopping List

Thus, given my expectations, I have come up with a shopping list of requirements for my CSIRT Toolkit which fall into, but are not limited to, the following domains that will be stored on my Secure CSIRT Toolkit Disk:

1. Tools to carry out and support the Acquisition Phase – Here my preference is the Belkasoft Evidence Center (BEC) 2019
2. A tool with the capabilities to conduct analysis on any acquisitions – again BEC offers very high capabilities to perform this requirement
3. Ancillary tools to support micro-operations, e.g. to support creation of HASH Signatures, or say to capture a dynamic RAM image of a live system

4. An area where case management may be maintained
5. Documentation Sets Repository – for example, Statement Templates, Evidence Acquisition Logs and Tags
6. Repository to separate Evidential Materials, Artifacts, and Completed Statements (One directory for each version (Prime, First Working Copy, Second Working Copy) and another for any encrypted drives as they are of a different profile, yet ostensibly contain the same content as the unencrypted images)
7. An On-Line Browser Tool that will capture any screen interactions with visited URLs and support Case Management – Here Paliscope is my preferred choice
8. Secure Communications Application + a Trusted VPN (ProtonMail, Hushmail, and ProtonVPN)
9. A GRC (Governance – Risk – Compliance) Repository where the pertinent operational document sets may be stored for consultation as required

All of which will be maintained with the most up to date releases and stored on a secure Certified Secure Drive to FIPS-140/2, and with the certified status to accommodate and store UK, and International Government Data (specific to the region of operations) – for example, NCSC (CPA) UK, NLNCSA BSPA (NL), and NATO RESTRICTED Level. **Fig 1** below is an example of the CSIRT drive at its current version.

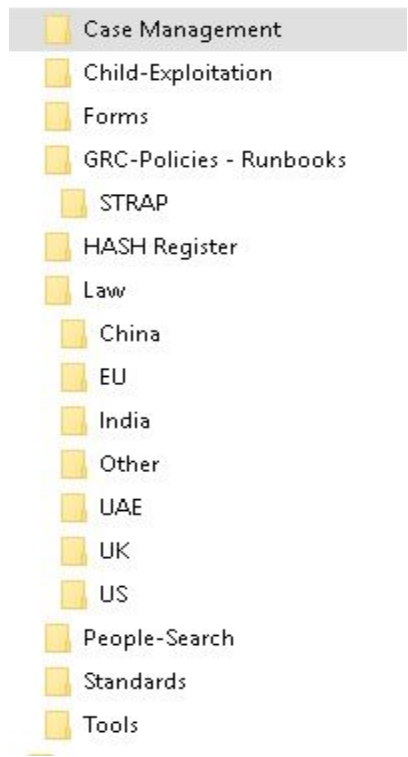


Fig 1 – CSIRT Disk Structure

To accommodate the physical medium in the form of a disk, my choice over the years has been the iStorage, now at the version of diskAshur Pro (available up to 5TB) for on-the road engagements, and a secondary office-based secure drive facility in the form of a iStorage diskAshur DT (available up to 14TB), which is used to backup any Case Management materials and acquired artifacts – again, stored *securely*, and *completely* off-line when access is *not* required.

Case Management and Browsing

One of my most favourite tools to use when attending a scene of incident, or investigating, is Paliscope (a tool used by law enforcement on an international scale). This is what I regard as my case management browser tool which captures all my online case interactions in a logical *contemporaneous* manner, and saves them into a specific case file, documenting – See **Fig 2/3** below – capturing:

- Events
- People
- Web Pages
- Images
- Physical and Virtual Locations.

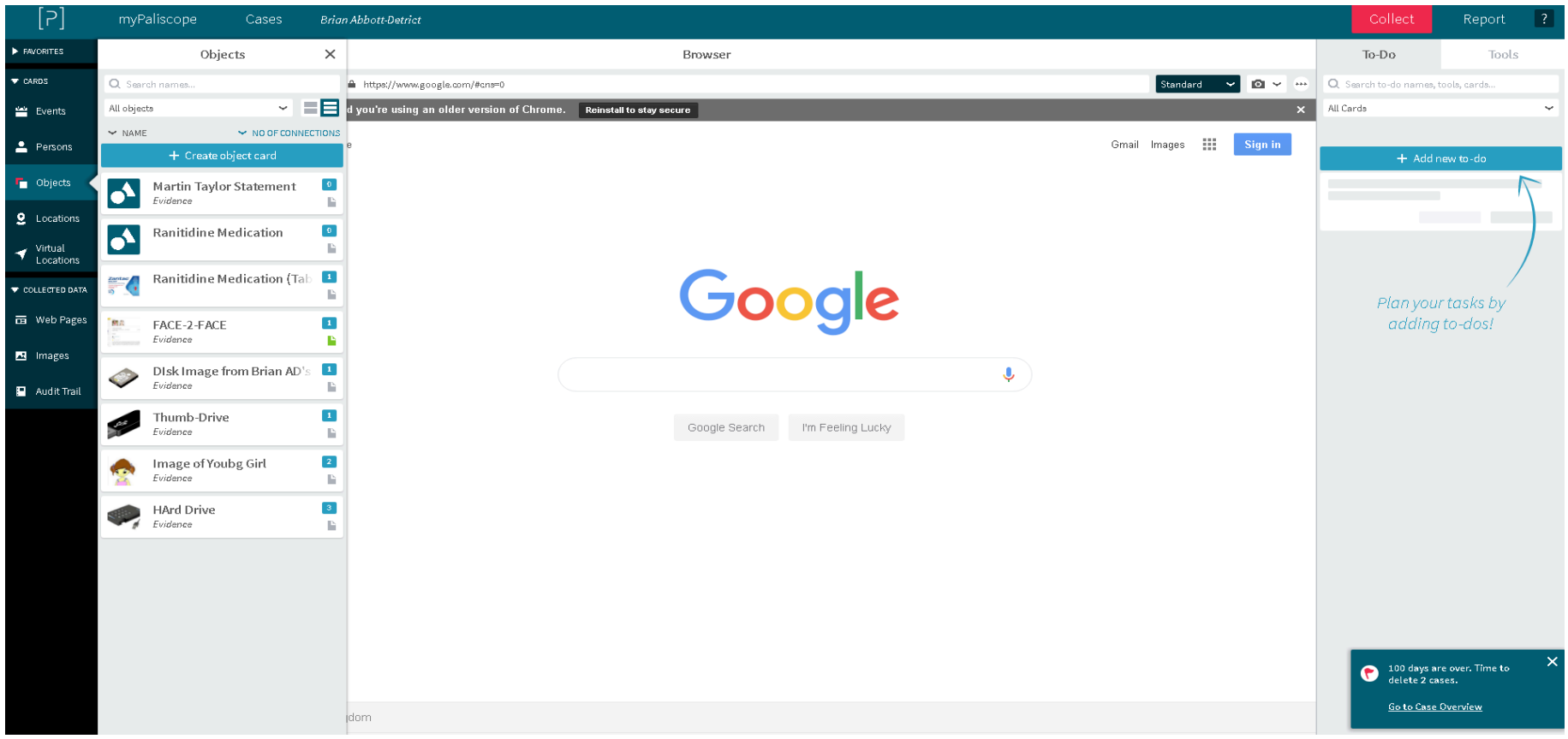


Fig 2 – Paliscope GUI

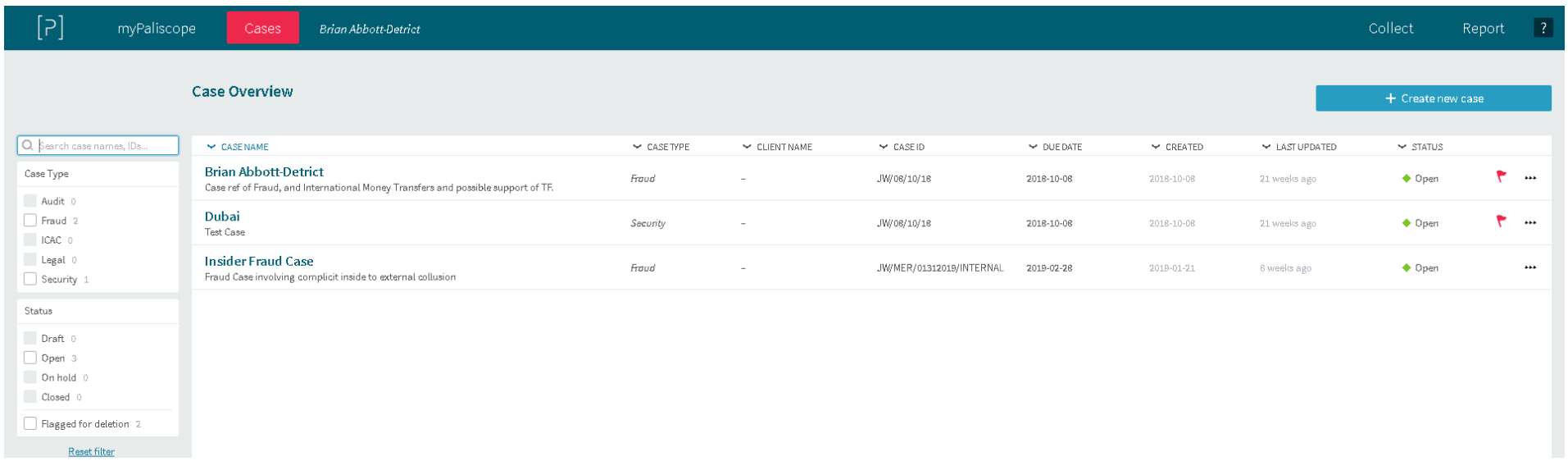


Fig 3 – Paliscope Case Management Interface

The Paliscope Tool is fully equipped with multiples of sub-technologies that assist with the investigation of images, and has face recognition plug-Ins which, in one example, has successfully been leveraged in multinational people trafficking investigations.

Conclusion - The Built Drive

As introduced above, the complete solution in this case is built on my iStorage diskAshur PRO secure drive. This solution not only accommodates the capacity for all the CSIRT Team Responder will require, but it goes well beyond that by accommodating the necessary security requirements to assure that the content, and any investigative related materials are secured and robust – thus, as the incumbent professional, one may safely testify in a Court of Law to the best of their knowledge and actions that the contained artifacts and objects have been *fully* secured and maintained in a controlled manner whilst at *rest*, or in *transit*, and that the *integrity* of all object has been *maintained* (unlike that of a laptop used for multiple purposes, and communications – thus can, and does, expose the content from corruption). And of course, the real value in utilising a CSIRT Toolkit on a drive is, it is ready to go at any time, and in one case I am aware of the concerned organisation leverages multiple disks to accommodate a team of First Responder Professionals who may attend a global scene at the drop of a hat. One of the most obvious advantages of using a secure drive is that it may be carried in hand with all the other necessary tools as hand-baggage, thus avoiding the potential of loss-in-transit. And on the financial front, this is an extremely cost-effective way of maintaining a secure, dedicated resource in a small footprint, cost

effective profile that may be issued on demand to the selected professionals – removing the preparedness stage of the headless chicken, which so often comes out to play at times of stress of an incident or suspected hack.

As a closing remark, I have all too often observed the practice of technology is king/queen, with the most important factors being ignored – that is to say, PROCESS, where deployments to scenes-of incidents have been commissioned with just a laptop armed with the required digital forensic tool, and nothing more – thus *not* accommodating the attending professional with all the tools necessary to carry out their mission. With the ready to go Toolkit on a Drive, this is overcome as no matter the skill level of the attending responder, they are *always* furnished with the materials, forms and direction they require to robustly engage.

[1] Dependent on the engagement, the side you are playing for, or other complications, in my experience for reasons various, attendance of an outsider, or third party may not always be welcome so it pays to attend the site under investigation fully equipped, for help may not always be on hand!

About John

A background of 22 years in Royal Air Force Security/Investigations/ Counter Intelligence operations [Overt/Covert] service, working with GCHQ, CESG, UK and US Agencies. ITSO and Systems Security Manager for CIA Accredited SCIF (Sensitive Compartmented Information Facility) Systems, Visiting Professor School of Science/Technology - Nottingham Trent University [NTU], Practising and Registered Expert Witness, Certified Forensics Investigator Practitioner [CFIP], Editorial Member at MedCrave Research for Forensics & Criminology, ENISA CEI Listed Expert, Editorial Member of the Cyber Security Research Institute [CRSI], Digital Forensics/Cyber Security Listed Trainer at Meirc [Dubai] delivering specialist Certified Digital Forensics, Audit and Cyber Security courses, Fellow of Royal Society for the Arts [FRSA], and a Belkasoft (Digital Forensics) Partner.

Using Cuckoo API (with Python) to submit and extract data from Cuckoo Sandbox

The author of the article requested to remain anonymous

In this post, we will look at how to use Cuckoo's lightweight API to extract information after submitting the sample to sandbox. In order for us to use this API functionality, we need a working Cuckoo Sandbox. It is highly recommended to install Cuckoo (version 2.0.6 preferred) in virtual environments like ***virtualenv*** or ***pipenv***, to make sure all the dependencies are separately and specifically installed for it.

Below are the links from one of the best tutorials I found which I also used as a reference, to install Cuckoo Sandbox. Thanks to the author who put really good effort in making the post!

- *Part 1*
- *Part 2*

During this installation phase, Cuckoo relies on a few main configuration files. These configuration files help us define general and behavior analysis options, set up auxiliary and processing modules, define virtualization options and reporting formats.

What is API?

Before we jump into knowing what Cuckoo API is and how to use it, let's first see what API means, and look into a few terminologies that come along. API stands for Application Programming Interface, which is basically an intermediary that allows two different applications within or outside a system to talk to each other and get the work done.

To explain this better, let us take a familiar example.

Most of us use smartphone apps these days. Let us take one such smartphone app, say *Grubhub,* which we use to order food. In this case, consider an initial system, which has you, and the restaurant, which basically has its kitchen that prepares food you want to eat. We can see here that the link between you and the kitchen who prepares the food is missing. This is when we introduce this *Grubhub* app into the system, which acts as a link or messenger or API that sends our request to the restaurant's kitchen.

The app here not only provides us with options like selecting the food type, restaurant, giving ratings, etc., it also provides options for the restaurant like managing the cost of the food, modifying the menu, giving you an estimated time for preparing food and delivering, etc. Similarly, the API also provides some options for us, and some to the application we are talking to.

Earlier, an API was often described as generic connectivity interface to an application, but now the modern API extends with some extra functionality that makes it more useful:

- Believing in standards (HTTP and REST)
- Very well documented, and versioned so it's easy to understand and access
- Designed for specific target audience (mostly developers)

What is REST API?

REST stands for Representational State Transfer. The REST API is basically designed to make use of existing protocols. When using Web APIs, REST usually takes advantage of the HTTP protocol. However, REST can be used on nearly any protocol. Also, unlike SOAP (Simple Object Access Protocol), REST is not restricted to getting output in XML, but instead can return JSON, XML, YAML, and many other formats, which makes life easier for us to parse and view the output.

What is API Endpoint?

Endpoint is basically the dead-end or destination of that channel or communication that gives us some data. If we consider the above example again, for our easy understanding: the app gives you access to the menu, so just this whole visible menu is one endpoint. Selecting an item or multiple items from the

menu can be another endpoint within that endpoint. Similarly, adding to the cart is another endpoint, and placing an order at the end is another endpoint.

The common HTTP endpoint methods used are:

- sending information (POST)
- receiving information (GET)
- adding or updating information (PUT)
- deleting information (DELETE)

Example:

```
GET /restaurant

GET /restaurant/menu

POST /items/submit
```

In the above example, GET is the type of action performed by the Grubhub app to get restaurant details, which is the first endpoint. Then we can see the menu endpoint, which gives the menu of that specific restaurant.

Similarly, we can see POST action performed by the API to submit the items from the cart, so the restaurant can see them.

HTTP Status Codes

Basically, every HTTP communication has a status code that is sent back by the server to tell us how the server handled the communication. Below are the top few common HTTP status codes:

1. *200 OK*
2. *400 Bad Request*
3. *401 Unauthorized*
4. *403 Forbidden*
5. *404 Not Found*

6. *500 Internal Server Error*

7. *503 Service Unavailable*

8. 550 Permission denied

If you want to know more about how APIs and API Endpoints work, please check these links below:

API

API Endpoints

Status Codes

What is Cuckoo API?

Cuckoo API is basically a lightweight REST API server. It is implemented using flask, a microweb framework written in Python.

As we already know, once the Cuckoo Sandbox is installed and the VM is ready, there are four different ways we can submit a sample for analysis, and extract information about that sample. They are:

1. Using a Cuckoo Submission Utility (cuckoo submit)

2. Cuckoo API

3. Distributed Cuckoo

4. Python Functions

In this part, we will only look into submitting a sample and extracting useful information using Cuckoo API. Using Cuckoo API is preferred by most security researchers because the functionality of this API can be scriptable using Python, and automating both analysis submission and extraction of the data it is always easier when REST APIs are used.

The Python functions, on the other hand, provide us with a few functions we can use to perform independent tasks. For example, if we want to just submit a file to the sandbox we can quickly use ***add_path()*** function provided by Cuckoo core database, which adds a local file from that path to the list of pending analysis tasks in the sandbox.

Installing Cuckoo API

We don't have to specially install Cuckoo API. When you install your Cuckoo Sandbox using `pip install -U cuckoo`, this API server gets installed automatically. If the Cuckoo Sandbox is installed without any errors, we can start the API server by typing the below command in the terminal

```
cuckoo api
```

You can also change the host values using the following command:

```
cuckoo api -H <host IP> -p <port number>
```

or for newer versions

```
cuckoo api <host IP> -p <port number>
```

Example: `cuckoo api -H 10.10.10.10 -p 1414`

Default is: `localhost:8090`

While the above default method of launching the API server works fine, some users prefer to deploy a server exposing the API as a Web Server Gateway Interface (WSGI) application.

Things we need to access Cuckoo API Server using Python

Two main things we need in order for us to access Cuckoo API server using Python are:

1. API Access Token

2. HTTP library for Python (e.g. requests), which can be installed using `pip`

The API Access Token is a secret value that needs to be set in one of the configuration files, *cuckoo.conf*. To access the API, we must send the *Authorization: Bearer token* header with all your requests using the same token used in the configuration file. Up until version 2.0.6, Cuckoo Sandbox did not have any form of authentication, meaning anyone could connect to Cuckoo instance if they knew the IP and port number where it was deployed. The *Bearer token* was implemented in Cuckoo 2.0.6. Users that upgrade from an older version need to require this Bearer token for backwards compatibility reasons. But clean installations automatically generate a random token under `<cuckoo root directory>/conf/cuckoo.conf` under the `api_token` field under `[cuckoo]` section.

Cuckoo API Endpoints

The Cuckoo API provides various endpoints to extract different types of information. Choosing the endpoints is completely dependent on what type of data we are trying to submit or extract from that endpoint for the sample submitted, using our Python code.

In this post, we will try to cover only a few important endpoints that help us extract important information from the sample.

Below are the HTTP methods with all the Cuckoo API Endpoints, and few examples in Python using them with few required and optional parameters.

`POST /tasks/create/file` - Adds a single file to list of pending tasks for analysis

Parameters available:

- `file (required)` - sample file
- `package (optional)` - analysis package to be used for the analysis
- `timeout (optional)` - analysis timeout (in seconds); int
- `priority (optional)` - priority to assign to the task (1-3); int
- `options (optional)` - options to pass to the analysis package
- `machine (optional)` - label of the analysis machine to use for the analysis
- `platform (optional)` - name of the platform to select the analysis machine from (e.g. “Linux”)
- `tags (optional)` - define machine to start by tags. Platform must be set to use that. Comma separated
- `custom (optional)` - custom string to pass over the analysis and the processing/reporting modules
- `owner (optional)` - task owner in case multiple users can submit files to same cuckoo sandbox
- `clock (optional)` - set virtual machine clock (format %m-%d-%Y %H:%M:%S)
- `memory (optional)` - enable the creation of a full memory dump of the analysis machine

- `unique (optional)` - only submit samples that have not been analyzed before
- `enforce_timeout (optional)` - enable to enforce the execution for the full timeout value

`POST /tasks/create/url` - Adds a URL to the list of pending tasks for analysis

Parameters available:

- `url (required)` - URL to analyze
- `package (optional)` - analysis package to be used for the analysis
- `timeout (optional)` - analysis timeout (in seconds); int
- `priority (optional)` - priority to assign to the task (1-3); int
- `options (optional)` - options to pass to the analysis package
- `machine (optional)` - label of the analysis machine to use for the analysis
- `platform (optional)` - name of the platform to select the analysis machine from (e.g. "Linux")
- `tags (optional)` - define machine to start by tags. Platform must be set to use that. Comma separated
- `custom (optional)` - custom string to pass over the analysis and the processing/reporting modules
- `owner (optional)` - task owner in case multiple users can submit files to the same cuckoo instance
- `memory (optional)` - enable the creation of a full memory dump of the analysis machine
- `enforce_timeout (optional)` - enable to enforce the execution for the full timeout value
- `clock (optional)` - set virtual machine clock (format %m-%d-%Y %H:%M:%S)

`POST /tasks/create/submit` – Adds multiple files, or archive of files to the list of pending tasks

Parameters available:

- `file (optional)` - backwards compatibility with naming scheme for /tasks/create/file
- `files (optional)` - sample(s) to inspect and add to our pending queue
- `strings (optional)` - newline separated list of URLs and/or hashes
- `timeout (optional)` - analysis timeout (in seconds); int
- `priority (optional)` - priority to assign to the task (1-3); int
- `options (optional)` - options to pass to the analysis package
- `tags (optional)` - define machine to start by tags. Platform must be set to use that. Comma separated
- `custom (optional)` - custom string to pass over the analysis and the processing/reporting modules
- `owner (optional)` - task owner in case multiple users can submit files to the same cuckoo instance
- `memory (optional)` - enable the creation of a full memory dump of the analysis machine
- `enforce_timeout (optional)` - enable to enforce the execution for the full timeout value
- `clock (optional)` - set virtual machine clock (format %m-%d-%Y %H:%M:%S)

`GET /tasks/lists/<limit or offset>` - Returns a list of tasks from database. We can set limit of entries to return.

Parameters available:

- `limit (optional)` - maximum number of returned tasks; int
- `offset (optional)` – data offset; int

`GET /tasks/sample/<sample_id>` – Returns list of tasks stored in database for a given sample

Parameters available:

- `sample_id (required)` – sample id to list tasks for; int

`GET /tasks/view/<id>` – Returns details about a task assigned to a specific ID

Parameters available:

- `id (required)` - ID of the task to lookup; int

`GET /tasks/delete/<id>` - Removes the given task and then deletes the results

Parameters available:

- `id (required)` - ID of the task to delete; int

`GET /tasks/reschedule/<id or priority>` - Reschedule a task with the specified ID and priority Default priority:1

Parameters available:

- `id (required)` - ID of the task to reschedule; int
- `priority (optional)` - Task priority; int

`GET /tasks/screenshots/<id>` - Gets one or all the screenshots associated with a task ID

Parameters available:

- `id (required)` - ID of the task to get the report for; int
- `screenshot (optional)` - numerical identifier of a single screenshot

`GET /tasks/report/<id or format>` – Gets report of the analysis associated with a task ID Default: JSON

Parameters available:

- `id (required)` - ID of the task to get the report for; int
- `format (optional)` - format of the report to retrieve [json/html/all/dropped/package_files]. If none is specified, the JSON report will be returned.

GET /tasks/rereport/<id> - Reruns reporting for the analysis associated with a task ID

Parameters available:

- `id (required)` - ID of the task to rerun report; int

GET /tasks/reboot/<id> - Add a reboot task to database from an existing analysis ID

Parameters available:

- `id (required)` - ID of the task to rerun report; int

GET /memory/list/<id> - Returns list of memory dump files associated with a task ID

Parameters available:

- `id (required)` - ID of the task to get the report for; int

GET /memory/get/<id or pid> - Gets only one specific dump file associated with a task ID

Parameters available:

- `id (required)` - ID of the task to get the report for; int
- `pid (required)` - numerical identifier (pid) of a single memory dump file

GET /files/view/<id> - Search the analyzed binaries/files by MD5, SHA256, or internal ID

Parameters available:

- `md5 (optional)` - MD5 hash of the file to lookup
- `sha256 (optional)` - SHA256 hash of the file to lookup
- `id (optional)` – ID of the file to lookup; int

GET /files/get/<sha256> - Returns the content of a binary with a specific SHA256

Parameters available:

- `sha256 (optional)` - SHA256 hash of the file to lookup

`GET /pcap/get/<task_number>` - Returns the content of the PCAP associated with given task

Parameters available:

- `task` – task number; int

`GET /machines/list` - Returns the list of analysis machines available to Cuckoo

`GET/machines/view/<machinename: str>` - Returns details on the analysis machines associated with specific name

`GET /vpn/status –` Returns VPN status

`GET /cuckoo/status` - Returns cuckoo version installed/upgraded to, disk space available, used and total, hostname, machine name and cpuload (which loads for the past minute, the past 5 minutes, and the past 15 minutes, respectively; only available under Linux machines)

`GET /exit` - Shuts down the API Server

Note:

All the endpoints mentioned above may not give the same response status codes because the response completely depends on the type of action the endpoint performs.

For example:

Case1: Let's say we are submitting a file to the sandbox using API endpoint *POST /tasks/create/file*. We can only get 200 or 400 status codes. 200 for successful submission of the file to sandbox, and 400 if a duplicate file is detected.

Case2: Let's say now we are submitting a URL to the sandbox using API endpoint *POST /tasks/create/url*. In this case, we can only get status code 200 to show the successful submission, and we can't get 400 as status code because we submitted the URL only and there is no file in our submission to check for redundancy.

Examples:

In this first pseudo example, let's submit a sample (single file) to the Cuckoo instance.

From figure 1, we can see that I am using requests library and Cuckoo API's /tasks/create/file endpoint to do POST operation. In order to achieve that, I am authenticating to the API using the token in the headers. So, I am basically submitting a sample from a local path to the sandbox with some optional parameters. This way, we already have some details about this sample, because if I pull this sample in future, it will have all that metainformation (as optional parameters) and will be easy for me to verify.

```
import requests

# ------ POST /tasks/create/file ------
URL = "<IP:Port Number>/tasks/create/file"
FILE = "<Path to sample>"
HEADERS = {"Authorization": "Bearer <token>"}

with open(FILE, "rb") as sample_file:
    files = {
              "file": ("temp_name", sample_file),            // required
              "priority": <set between 1 -3>,                // optional
              "machine": <label of analysis machine>,        // optional
              "platform": <name of platform, eg: windows>,   // optional
              "tags": <comma seperated words/tags, eg:UPX>,  // optional
              "owner": <name of the user who submitted file>, // optional
              "memory": <enable creation of memory dump>,    // optional
              "unique": <only new samples>                   // optional
    }

    r = requests.post(URL, headers=HEADERS, files=files)

# check the status code
print(r.status_code)      // either 200 or 400

task_id = r.json()["task_id"]
# check task ID
print(task_id)
```

Figure 1: Example of *POST /tasks/create/file*

In this second pseudo example, from figure 2, I am using the requests library again and Cuckoo API's /tasks/create/url endpoint to do POST operation. In order to achieve this, I am again authenticating to the API using the token in the headers. So, I am basically submitting a URL that has a sample to the sandbox with a few optional parameters. This way, I know some details about this URL when I extract in future.

```
30  ------------------------------------------------------------------------------
31  import requests
32
33  # ------- POST /tasks/create/url -------
34  URL = "<IP:Port Number>/tasks/create/url"
35  FILE = "<Path to sample>"
36  HEADERS = {"Authorization": "Bearer <token>"}
37
38  input = {
39            "url": <sample_url                    >        // required
40            "priority": <set between 1 -3>,                // optional
41            "machine": <label of analysis machine>,        // optional
42            "platform": <name of platform, eg: windows>,   // optional
43            "tags": <comma seperated words/tags, eg:UPX>,  // optional
44            "owner": <name of the user who submitted file>, // optional
45            "memory": <enable creation of memory dump>,    // optional
46            "unique": <only new samples>                   // optional
47      }
48
49      r = requests.post(URL, headers=HEADERS, data=input)
50
51  # check the status code
52  print(r.status_code)      // 200 only
53
54  task_id = r.json()["task_id"]
55  # check task ID
56  print(task_id)
57
```

Figure 2: Example of *POST /tasks/create/url*

And, in the last pseudo example, from figure 3, we can see that I am just posting multiple URLs and multiple samples to the sandbox using the same token-based authentication but Cuckoo API's /tasks/create/submit endpoint.

```
59  ------------------------------------------------------------------------------
60  import requests
61
62  # ------- POST /tasks/create/submit -------
63  URL = "<IP:Port Number>/tasks/create/submit"
64  FILE = "<Path to sample>"
65  HEADERS = {"Authorization": "Bearer <token>"}
66
67  ## In this case we can either submit file as one file/archive or multiple files
68  ## If we submit as archive - it's same as POST /tasks/create/file
69  ## But if we submit as multiple files, we have to do as follows
70  URL = "<IP:Port Number>/tasks/create/submit"
71  HEADERS = {"Authorization": "Bearer <token>"}
72  r = requests.post(URL, files=[
73      ("files", open("sample_1.exe", "rb")),
74      ("files", open("sample_2.exe", "rb")),
75  ], headers=HEADERS)
76
77  ## for multiple URLs, we have to give as follows:
78  urls = ["xyz.com", "abc.com", "pqr.org"]
79  r = requests.post(URL, headers=HEADERS, data={"strings": "\n".join(urls)})
80
```

Figure 3: Example of *POST /tasks/create/submit*

Similarly, we can use other Cuckoo API endpoints as well, and write automation scripts that can get us all the required information we need from the sandbox.

References:

- https://cuckoosandbox.org/
- https://secure-seed.com/2018/04/17/setting-up-cuckoo-2-0-5-part-1/
- https://secure-seed.com/2018/04/18/setting-up-cuckoo-2-0-5-part-2/
- https://blog.nviso.be/2018/04/12/painless-cuckoo-sandbox-installation/
- https://libraries.io/github/SparkITSolutions/cuckoo

Narrowing down a location of an image

by Joshua Richards

This article will go through how you can narrow down a location for where the image was taken. The same generally applies for videos, too, as you can visually analyse these in the same way. Techniques such as reverse image searching with different sources, metadata, and visual analysis will all be explained.

1. Introduction

With modern day smart phones and cameras, photos are everywhere now. On average, a total of 1.8 billion images are uploaded to the internet every single day (Eveleth, 2015). This can be good because it allows people to see all parts of the world without needing to really go anywhere. However, sometimes you will see a photo but not know where it was taken, whether it's a random picture found on the internet, or part of a criminal investigation, the possibilities are endless.

This article will go through how you can narrow down a location for where the image was taken. The same generally applies for videos, too, as you can visually analyse these in the same way. Techniques such as reverse image searching with different sources, metadata, and visual analysis will all be explained.

How can geolocation be useful? There have been many instances where geolocation was used to track people down, like ISIS supporters who were posting images of themselves holding pieces of paper with writing on it, saying what country they are in and that they are a supporter. These people were located just because they showed some of their surroundings in the picture. Another example is when Europol

posts pictures asking for the public's help identifying objects and locations to help in child abuse cases. If people can find out where some of these locations are, lives will be saved.

(Link to the Europol 'Stop Child Abuse' page)

(https://www.europol.europa.eu/stopchildabuse)

2. Metadata

When people look at a picture, you see colours, objects, whatever has been captured or edited into the image. This isn't all there is though, there is a lot of data making up this image, and within that data is a lot of useful information we can use in investigations. One good website for getting picture metadata is (http://exif.regex.info/exif.cgi). You can either upload a file from your device or enter a direct URL to an image. It will show basic details like what camera was used, details about the lens, and the exact date and time that the image was taken according to the time set on the camera. This can already be great for certain scenarios, like if you wanted to prove someone took a certain photo, you could use this to show it was taken by a particular camera and then check if that person has the same one.

Camera:	Sony DSC-W130
Lens:	5.3 mm (Max aperture f/2.8)
Exposure:	Auto exposure, Portrait, 1/1,250 sec, f/7.1, ISO 400
Flash:	On, Return not detected
Focus:	AF-C AF Area Mode: Multi
Date:	**December 29, 2013** 1:12:00PM (timezone not specified) (4 years, 8 months, 12 days, 6 hours, 12 minutes, 4 seconds ago, assuming image timezone of US Pacific)
File:	**3,264** × **2,448** JPEG (**8.0** megapixels) 2,730,488 bytes (2.6 megabytes)

http://exif.regex.info/exif.cgi

There are a lot more details that can be gathered when you test this for yourself. Not every image will have the same results, some may return hardly anything if the metadata has been stripped, like it does when you upload photos to most social media websites now.

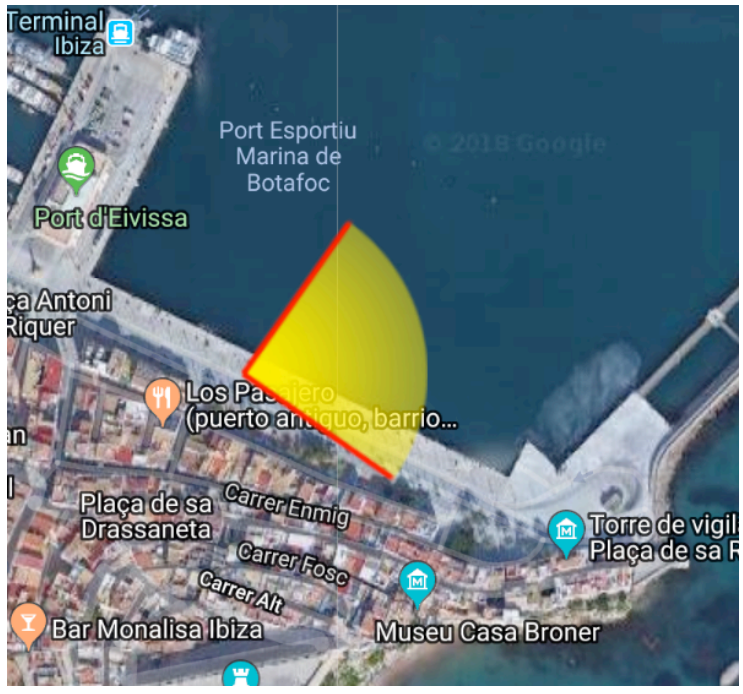
One very interesting piece of metadata you can get is a thumbnail of the original image. This won't be in all images, but it usually is shown when the photo was taken with a digital camera that still has all the metadata intact. Below shows the main image that you uploaded in good quality. If you click on it, you will be taken to a new page that only shows the image at full size.

http://exif.regex.info/exif.cgi

If a thumbnail is found, it will be shown to the left of the main image seen above. Text is shown saying what the size is of the thumbnail that it extracted. It will be very poor quality because thumbnails are very small, so when they are zoomed in, they get pixelated. This can be incredibly useful if someone used a basic online tool to crop an image that their face was in, for instance, it could still be shown in the thumbnail. It wouldn't be shown if it was edited on software like Adobe Photoshop, though, because that updates all the metadata.

http://exif.regex.info/exif.cgi

The other main type of result you can get is location data. When you take a photo with some cameras and phones, they can have settings on them that store the location of where each photo was taken. You will be shown a set of coordinates pinpointing the exact location and you get a map showing it along with the field of view based on the details of the camera.



http://exif.regex.info/exif.cgi

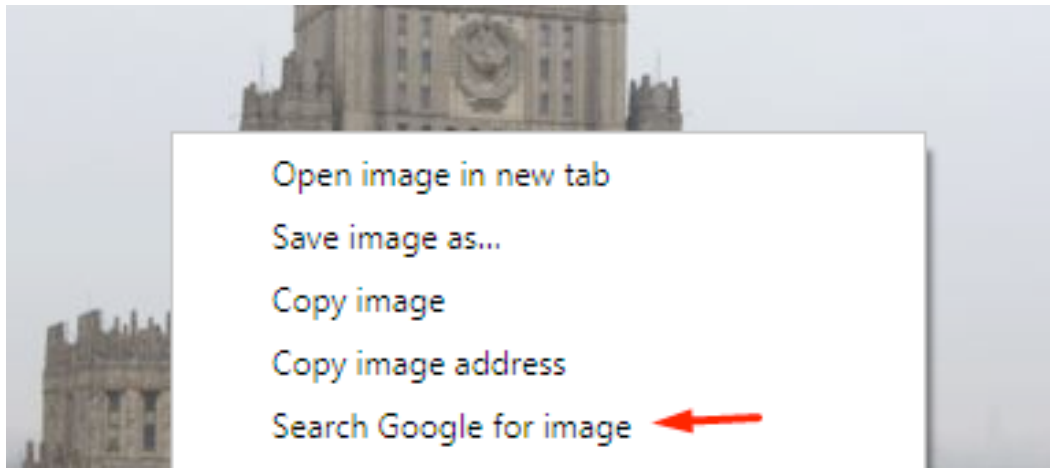
These are some of the basic yet important things you can gather from image metadata. There are many ways that these could help. For instance, finding out what camera the person used to take it or the camera could be made and used only in a particular country so that could narrow your search down to there. If you found that an image had been cropped and you got the thumbnail of the full image, it could reveal even one small hint, like a shop sign, that could narrow down the search. The location data, of course, is the most useful if your objective is to find out where it was taken, as that will tell you exactly where it was taken, if the metadata hasn't been tampered with in any way.

Metadata is a big part of forensics and will have been covered a lot previously, so this section of the article is more brief and shows you the basics, so that you can go out and test on your own images or ones you find online, and you can see for yourself what sort of information you get.

3. Reverse Image Searching

Most people who use the internet will know about search engines like Google that let you search for keywords to get results and they may know you can find images from these keywords. However, what many people don't know is that you can search an image to try find where that image is used online or even find similar looking images. The image below will be used as an example.

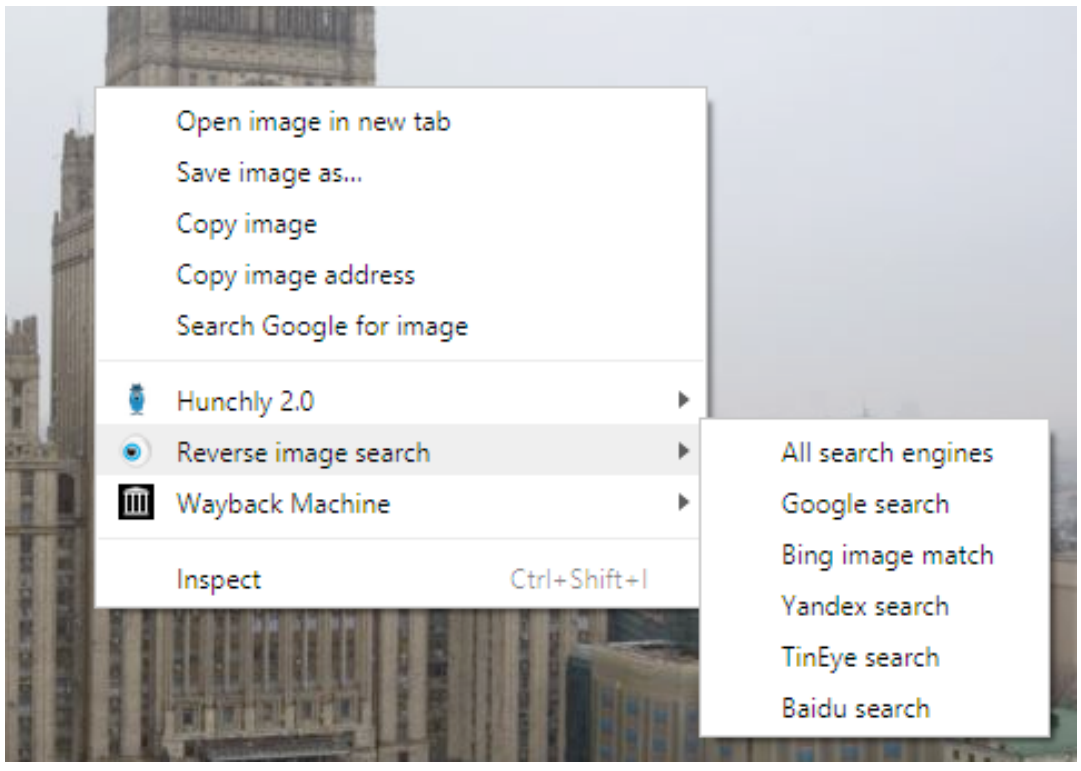
If you right click on the image with the Chrome browser, you can actually "Search Google for image" to instantly search it and it's so easy to do.



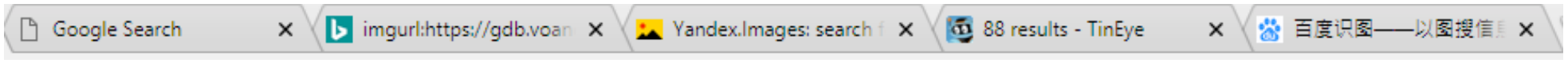
There is an extension for Chrome and Firefox called RevEye. This allows you to right click on an image and then click one single button to search the image on five different websites including Google, Bing, Yandex, TinEye, and Baidu:

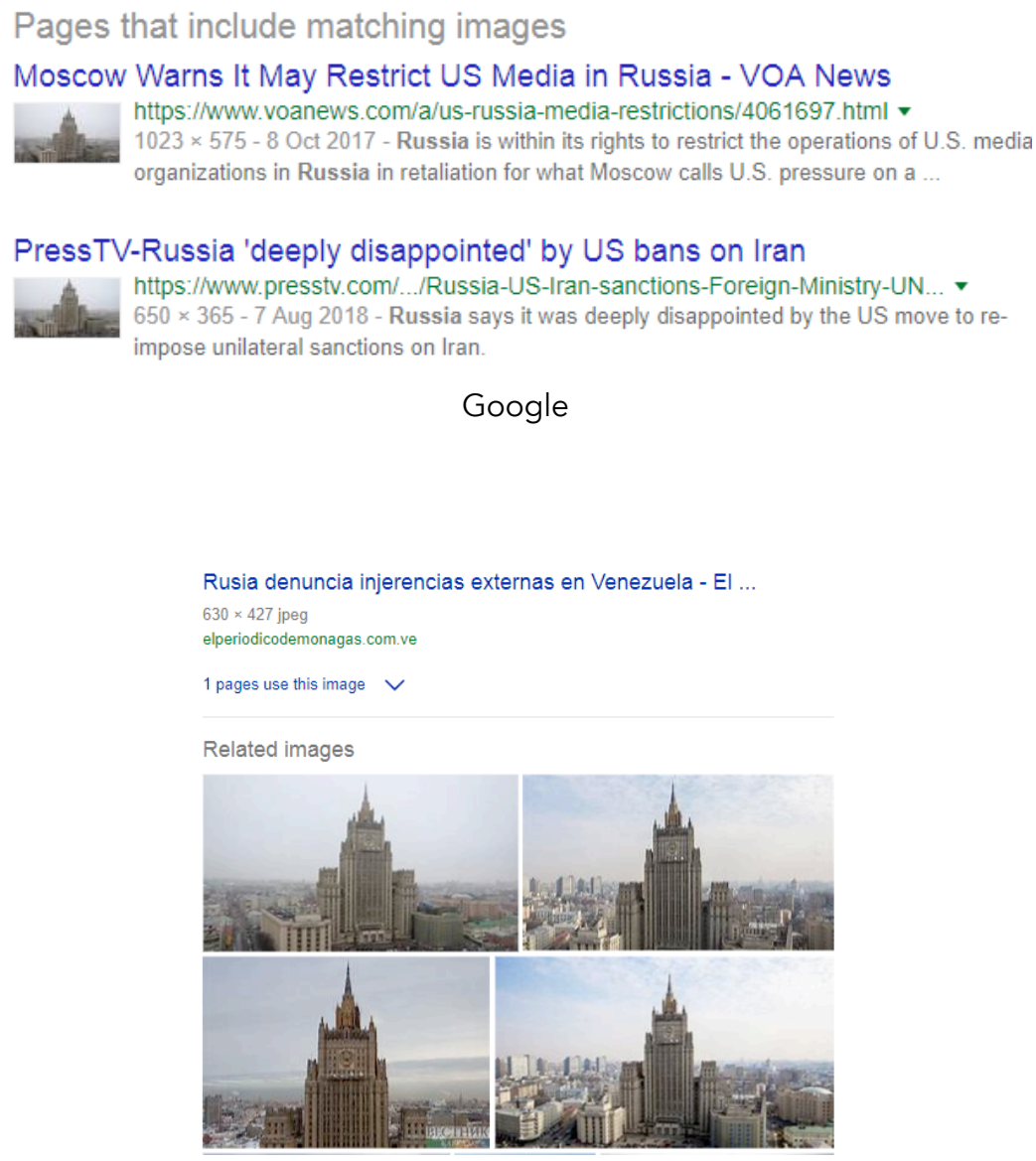
- https://chrome.google.com/webstore/detail/reveye-reverse-image-sear/keaaclcjhehbbapnphnmpiklalfhelgf
- https://addons.mozilla.org/en-US/firefox/addon/reveye-ris/

All you need to do after getting the add-on is right click on any image, hover over "Reverse image search" and it will then show you the options. You can click any of the individual ones to just search on that specific one or click "All search engines" to search it on all five.



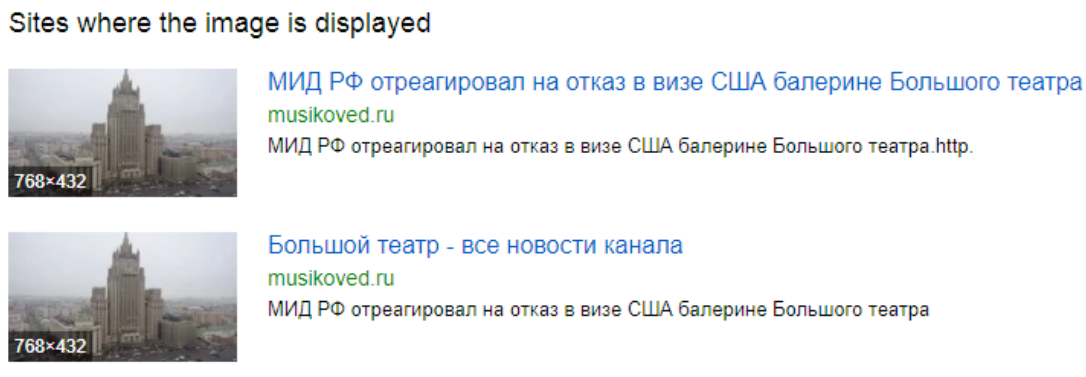
When you click to search on all search engines, it will open the five searches as new tabs.



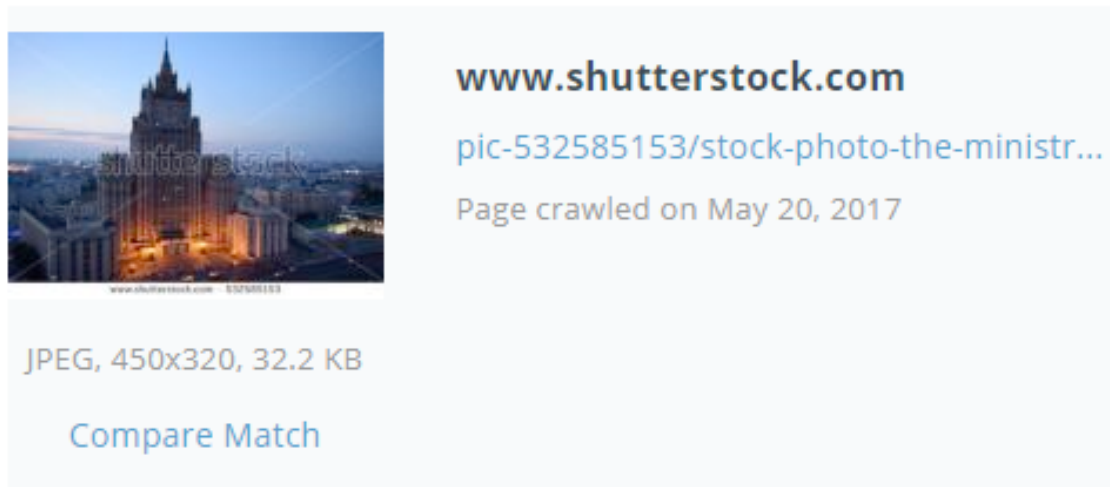


Google

Bing



Yandex



TinEye

Baidu

As you can see, all bring incredible results, except from Baidu… However, with Baidu bringing some unique results, to say the least, that could be helpful in some searches. There is no harm in using the all in one search and just taking a quick look anyway because there very well could be a good result there sometimes.

The example above showed where the search engines found the same image on pages, most of them also had another section to show images that it found that were just similar. This can be extremely useful because maybe it can't find your exact searched image because someone has taken a picture but not posted it anywhere for example, but it may be able to find something similar and that could lead you to determine the subject of the picture.

A commonly used tool that involves images is LightShot, also known as Prntscr. This basically allows you to take screenshots of anything on your screen extremely easily. You just press your print screen button, it then lets you select any part of your screen to screenshot specific parts. It doesn't end there, though,

because they give lots of options. You can upload it to their server so that a link is generated that you can send to other people so that they can see the screenshot. These links have changed from how they originally looked, but current they look like this (http://prntscr.com/krpi9g).

If you want to screenshot something just to put it into a document, for example, you don't need to upload it to copy, you can just select it on the screen then do CTRL + C to copy it, allowing you to paste it on any documents you want.

Another two options we have are to save the picture to your documents or print it straight out if you have a printer connected to your current device that you are using for this.

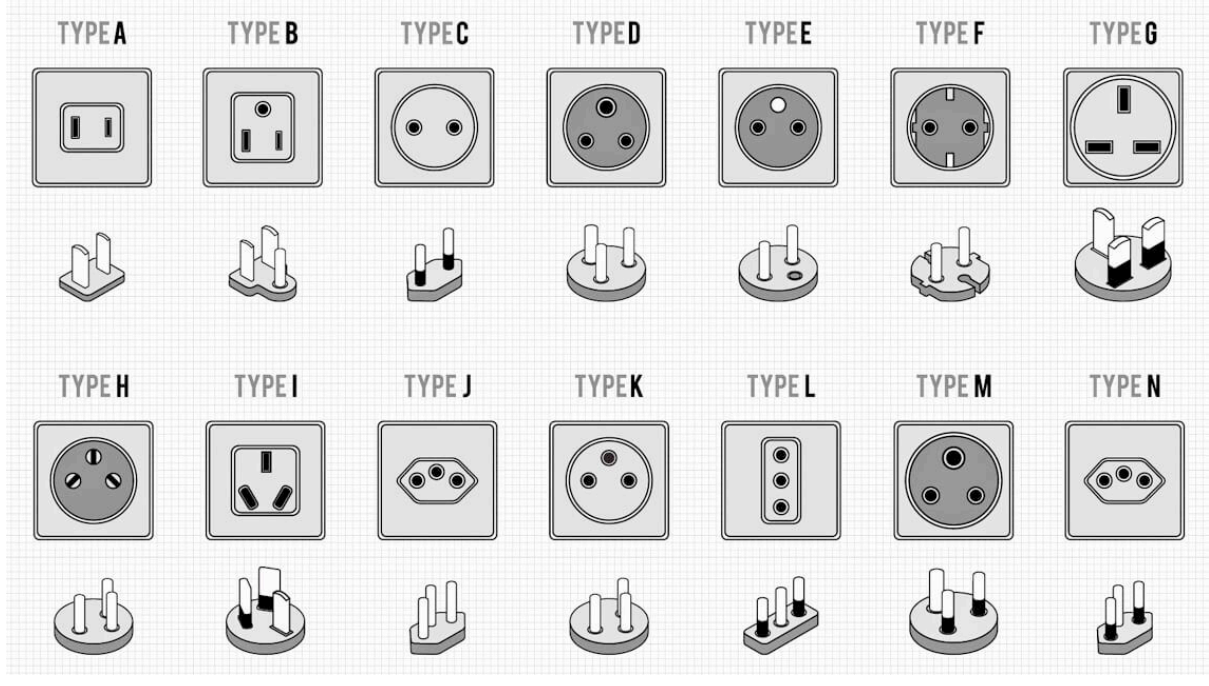
However, this is about reverse searching, which is where one of their other features comes into use. After you select something on your screen, you can click one single button, which is the letter "g" or you can do CTRL + G. It will automatically upload your image to their server then instantly reverse search it on Google to show you any results of similar images. Using this, there is no need to save images and upload them to Google, and sometimes with images, you can't actually copy them due to how the page is set up, so you'd have to go into the source code or inspect element and find it, but with this, you could just select it and search it straight away. Or if you only wanted to search one small part of a bigger image, that isn't a problem when using this. Another scenario could be if you wanted to reverse search something that was in a video, this is the easiest way to do that.

4. Visual Analysis

There are so many methods to locate where a photo was taken, like the ones explained above in the article. However, sometimes they will provide you with no results, so you instead must analyse the picture yourself. This is where attention to detail comes in very handy. Anything in the picture could potentially lead you to the location. Maybe a shop sign with a name or logo on it, even if it just shows half of one, architecture of the buildings shown, license plates on cars, road signs, it could be anything.

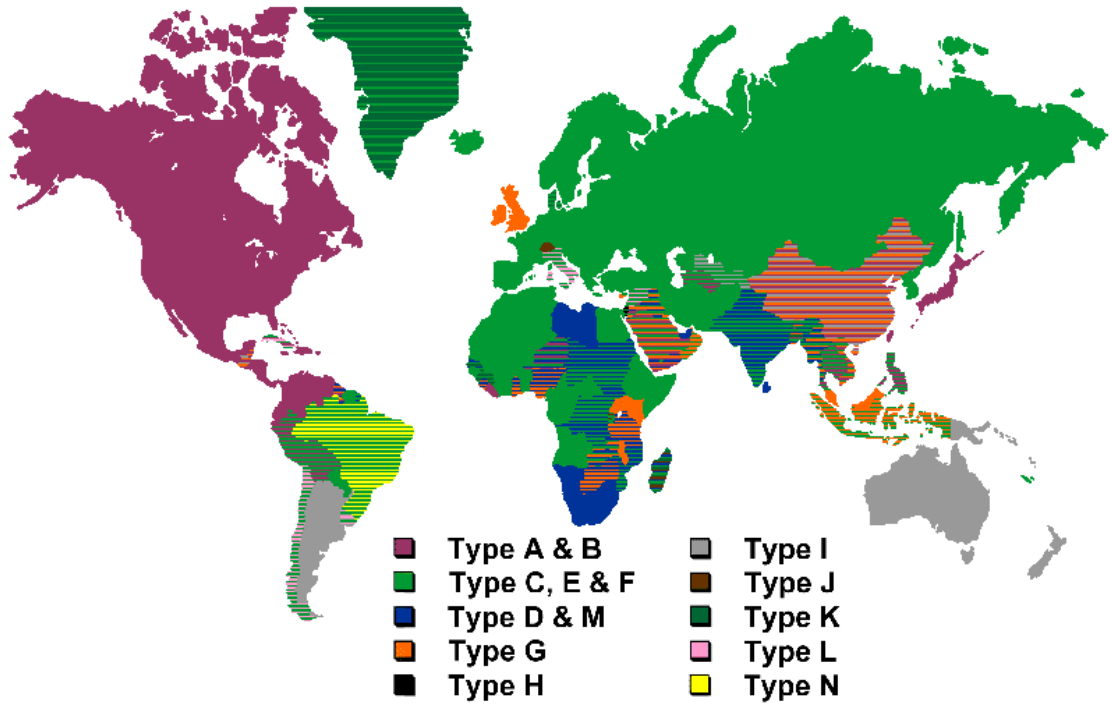
Sometimes there will be things so small and normal that you overlook them, but they could narrow your search a lot. For example, a wall socket, these are often different depending on the country you are in. While some countries do use the same ones, at least you have narrowed it from being anywhere in the world to just a couple possible countries. Below is a picture showing fourteen different sockets, each

labelled by a type letter. Now, if you spot a socket in an image, you can refer to this to see what type it is.



https://www.parisdiscoveryguide.com/image-files/1140-electric-socket-plug-types-16x9.jpg

Above was the picture showing different types of sockets, below is a map of the world showing by colour which sockets are used where. For example, if you saw that there was a socket and it was a type G socket, you can look here and see that is used in the United Kingdom and some others that are in orange.



https://www.power-plugs-sockets.com/wp-content/plugins/power-plugs-sockets/img/power-plugs-sockets-of-the-world.gif

Above was one example of a small detail people often overlook. There is so much more you can look for. For example, what can be seen below that could narrow it down?

While in theory this photo could be anywhere in the world, and no metadata to get coordinates, you must look at anything and everything you can to get clues. There is very little to go on from this picture, but at the very least, you can search the company on the lorry and find that they are based in the United Kingdom. While in this case it is still a large search area with little else to go off, at least you have a general idea. Some companies will work all over the world, so this won't be as easy to do with them. However, as can also be seen in the picture above, they often have domains written on them, which here says **owensgroup.uk**. If you saw **.nl**, it would likely be in the Netherlands, for example.

5. Conclusion

There are so many guides out there on geolocating images and videos. If you are interested in learning even more about this, it is certainly recommended that you check out Bellingcat. They do incredible investigations involving geolocation so there is a lot that can be learnt from them. There are certainly lots of interesting tutorials out there that you can view, and because these are all so well written, there is little point re-writing them into this, but above has shown some of the details you should look out for and basic techniques you can use when looking at images. Sometimes people will post pictures to Twitter and challenge people to find out where they took it from, so these methods could then be implemented as practice.

About the Author

Joshua Richards is a student at University of South Wales studying Cyber Security and Forensics. He works closely with a non-profit organisation called Trace Labs and is an administrator in the Word Class Investigator Slack community. His interests involve open source intelligence, forensics, and solving problems like riddles and puzzles. He enjoys meeting new people and helping to develop their skills.

Malware analysis with Cuckoo Sandbox

by Antonio Farina

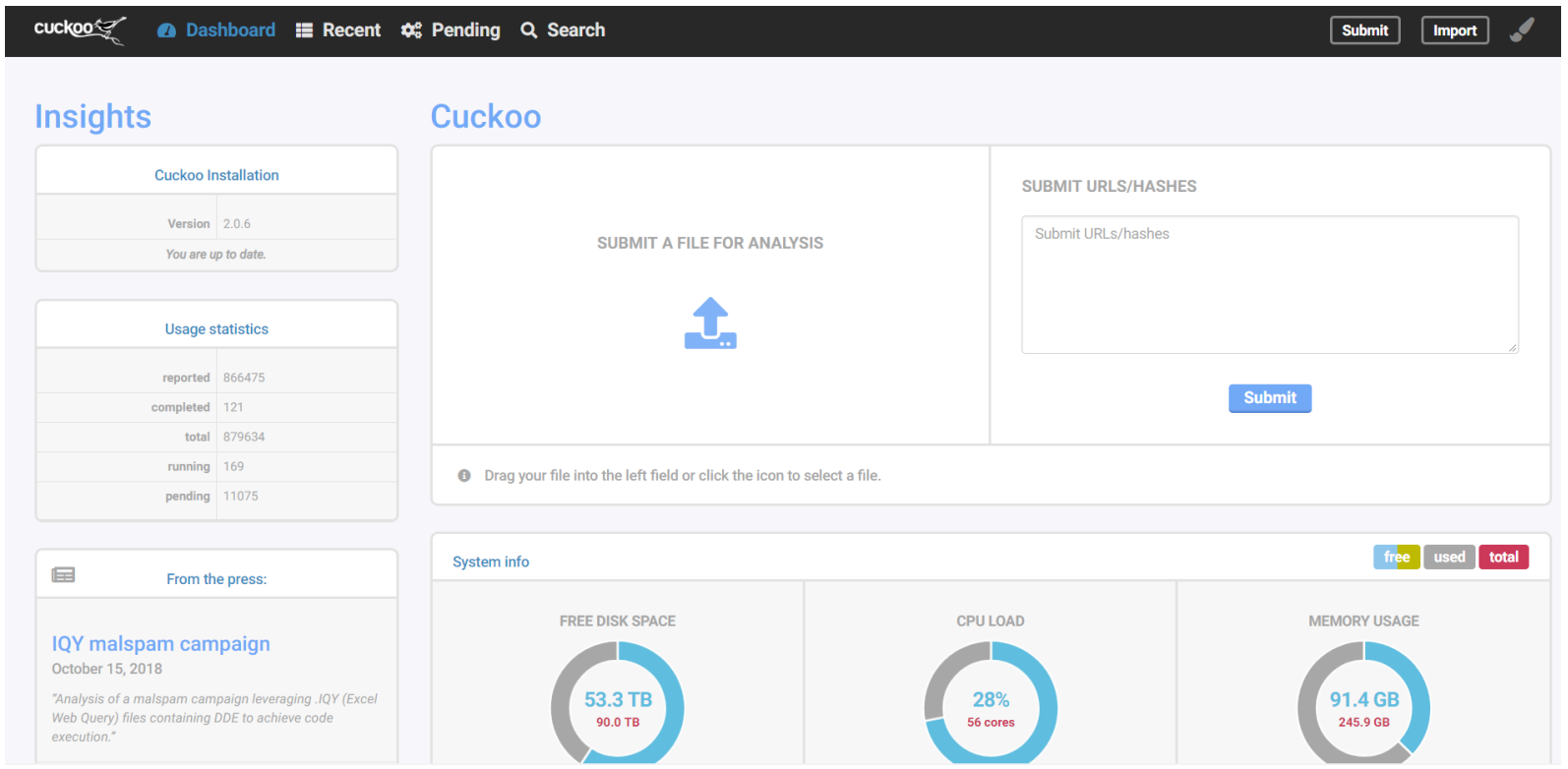
Using all the information extracted from the malware execution, Cuckoo will provide a detailed but easy to understand report containing the evidence to perform a first triage and attribution. Moreover, Cuckoo is designed in a modular way, so anyone could extend the sandbox functionalities writing his own module. Let's use it!

In malware analysis, the initial triage is a crucial phase to allow the analyst to understand briefly the entire malware's behavior and then to do a first attribution of the malware, linking it to a specific family or APT group. After this first phase, the analyst could dig into the sample through in-depth analysis, extracting evidence and specific characteristics that will be used to recover compromised systems and to avoid new attacks from the same source. Thus, the malware analysis' initial phase requires a virtual environment in which the malware can show its complete behavior without damages to the underlying system or infrastructure. This virtual system should be as similar as possible to a real machine, but it must be easy to recover and able to trace and analyze all malware activities. The sandboxes were born to fulfill these tasks. Cuckoo Sandbox is the most popular open source automated malware analysis system and it is able to analyze many different malicious files (executables, Office documents, PDFs, etc.) targeting the most common platforms, such as Windows, Linux, Mac OS X and Android. As expected, it can trace all the malware system calls, inspect the memory searching for malicious artifacts and analyze network traffic, even when it is encrypted. Using all the information extracted from the malware execution, Cuckoo will provide a detailed but easy to understand report containing the

evidence to perform a first triage and attribution. Moreover, Cuckoo is designed in a modular way, so anyone could extend the sandbox functionalities writing his own module.

Let's use it

Cuckoo's user interface is very intuitive. It is accessible using a common browser and looks like the following image:



On the top menu, you have different tabs referring to the sections handled by Cuckoo:

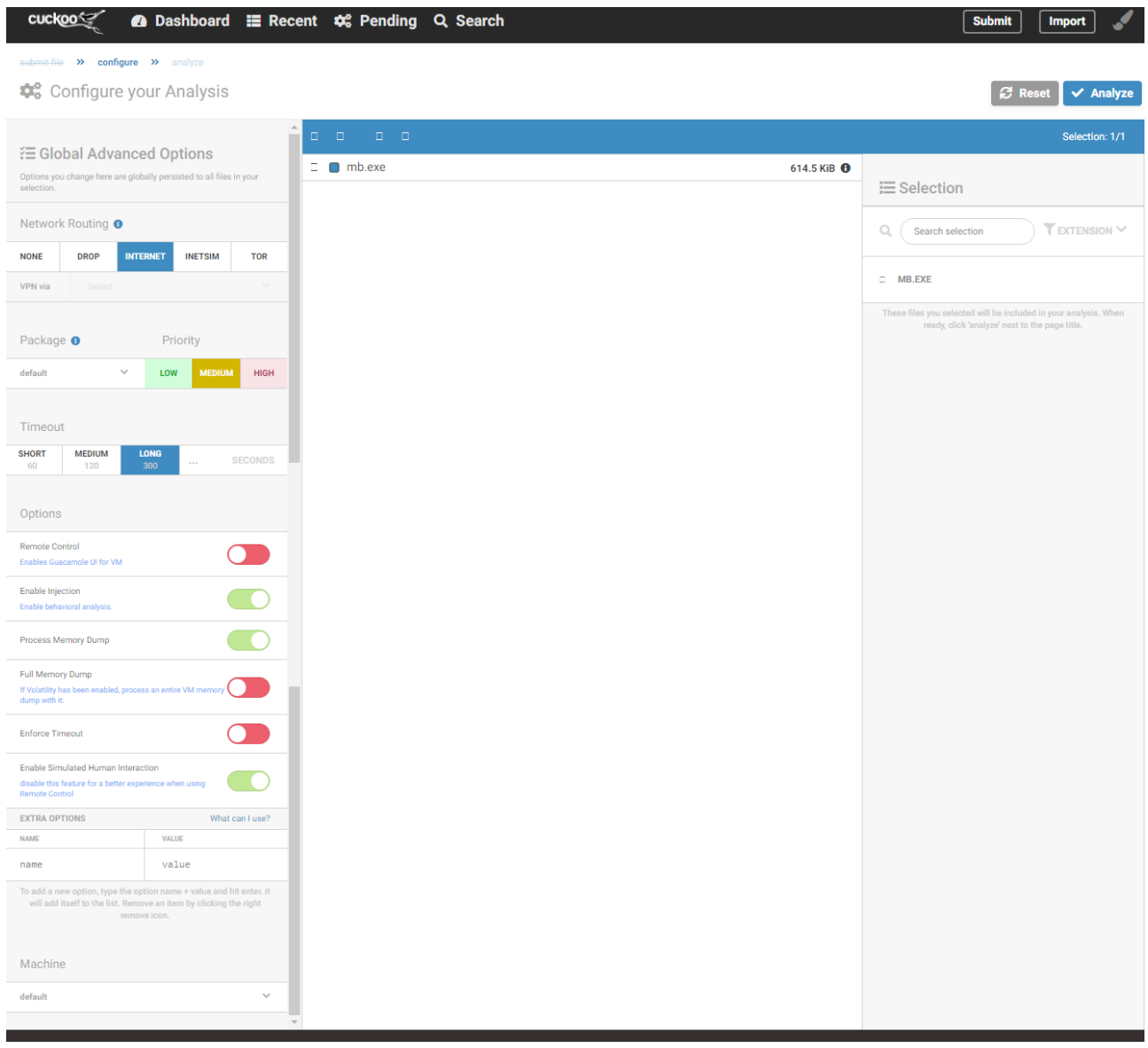
- *Dashboard* is the homepage of Cuckoo WebUI;
- *Recent* reports the latest analysis done through the sandbox;
- *Pending* contains a list with all the malware samples waiting to be launched into the sandbox;
- *Search* is used to find a specific report;
- *Submit* allows one to upload a new malware to analyze;
- *Import* allows one to open a file containing the report exported from another Cuckoo Sandbox.

Referring to the above image, the Dashboard section shows information about Cuckoo and the system in which it is installed, i.e. the "System info" panel displays the amount of the resources consumed by

the sandbox in terms of disk space, CPU load and memory usage. The "Usage statistics" panel reports, instead, an overview about the total number of malwares submitted, the reported ones and those waiting to be analyzed.

You can submit a new malware easily by clicking on the big central panel. It opens a popup window in which it is possible to select from your machine the file you want to submit.

After selecting the file, the next step is to choose the best sandbox configuration to allow the malware to reveal its complete behavior.



In this page, you have many options to choose if you want to customize the analysis process and, consequently, the information extracted by the sandbox.

The "Network routing" section allows you to specify what type of connection is enabled into the virtual machine in which the malware will be executed:

- *NONE* indicates that Cuckoo Sandbox will not apply any kind of routing rule. Cuckoo allows the analysis to route as defined by a third party, in example by IPTABLES rules specified in the system in which the sandbox is installed;
- *DROP* specifies that the only possible traffic is internal Cuckoo traffic (between the Cuckoo processes and the virtual machine in which the malware is deployed) and, hence, any DNS requests or outgoing TCP/IP connections are blocked;
- *INTERNET* provides full internet access to virtual machines through one of the connected network interfaces;
- With *INETSIM* option, you can connect the Cuckoo instance with the Internet Services Simulation Suite, called INetSim (https://www.inetsim.org), in order to simulate a set of standard Internet services in a lab environment. This configuration is useful when you want to analyze a malware's network traffic without allowing it to communicate using the real network. If you want to understand what type of network request is done by a malware to its Command and Control, but you don't want the request to go to the real malicious server, this is the best option!
- *TOR* and *VPN* enable traffic to be sent through Tor network or through one of multiple predefined VPN endpoints. Many Command and Controls (C2Cs) are hosted on Tor network, so if this option is not enabled and a malware bot communicates with its Tor C2C, you will not be able to analyze the network traffic.

Moreover, Cuckoo allows you to specify a priority for each submitted sample in order to schedule first the analysis with higher priorities. Obviously, the analysis will not be unlimited, so you can indicate a max analysis time using the *Timeout* option.

Finally, the sandbox provides a series of advanced options:

- *Remote Control* allows you to view the virtual machine screen from within the Cuckoo WebUI and interact with analyses in real time. Remote control option needs the presence of Guacamole server (https://guacamole.apache.org) on the virtual machine and an appropriate advanced configuration of the Cuckoo instance.

- *Enable Injection* allows Cuckoo Sandbox to trace all the API functions invoked by the malware, in order to reconstruct its behavior flow. If the option is disabled, Cuckoo will be only able to report the final state of the infected system (i.e. files created or deleted, registry keys modified and network traffic) without specifying how the malware reached its malicious goal.

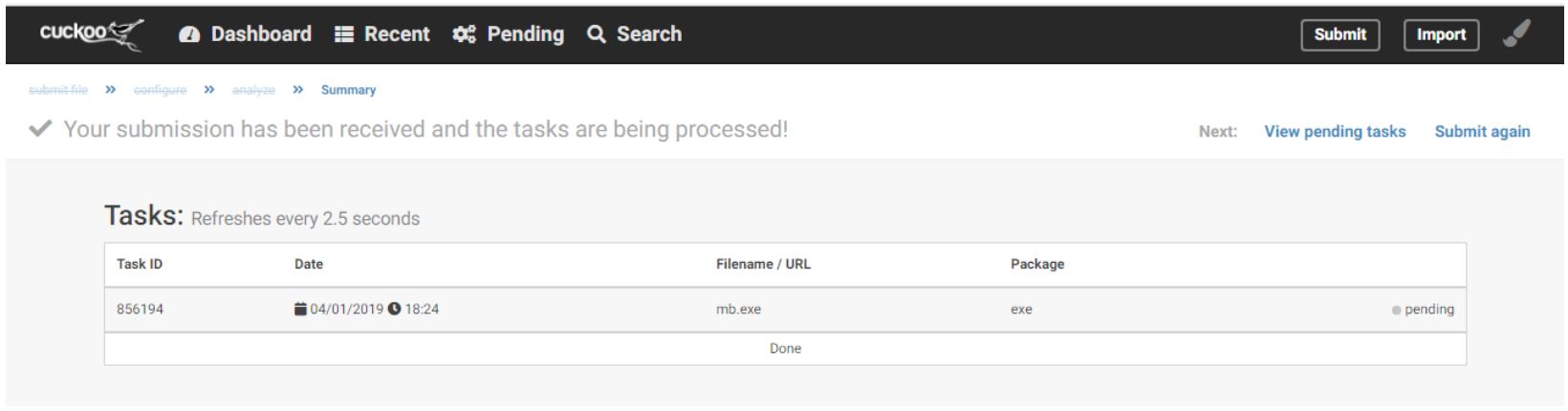
- Cuckoo Sandbox can take a snapshot of the process memory or the entire system memory inspecting each one with *Volatility* (https://www.volatilityfoundation.org) and *Yara engine* (*https://yara.readthedocs.io/en/v3.8.1/)* looking for malicious artifacts. These features can be enabled using the options *Process memory dump* and *Full memory dump*.

- *Enforce timeout* forces Cuckoo to continue the analysis until the end of the timeout.

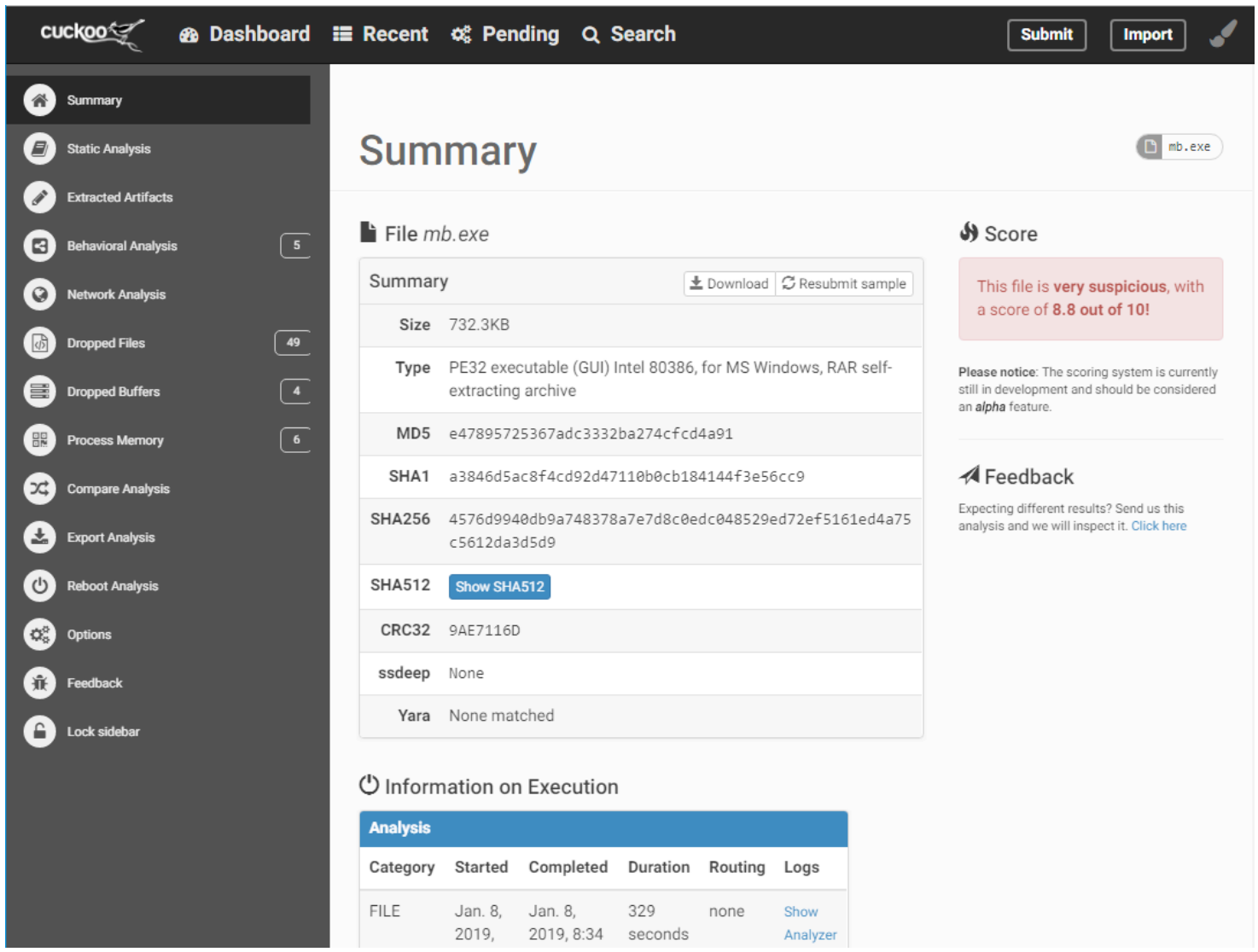
- Cuckoo has the capability to recognize message popup and button displayed by the executable into a virtual machine and can click on them simulating the human behavior. The feature is enabled using *Enable Simulated Human Interaction* option.

- The *EXTRA OPTIONS* section allows you to specify some parameters that will be sent to the Cuckoo modules, including custom ones.

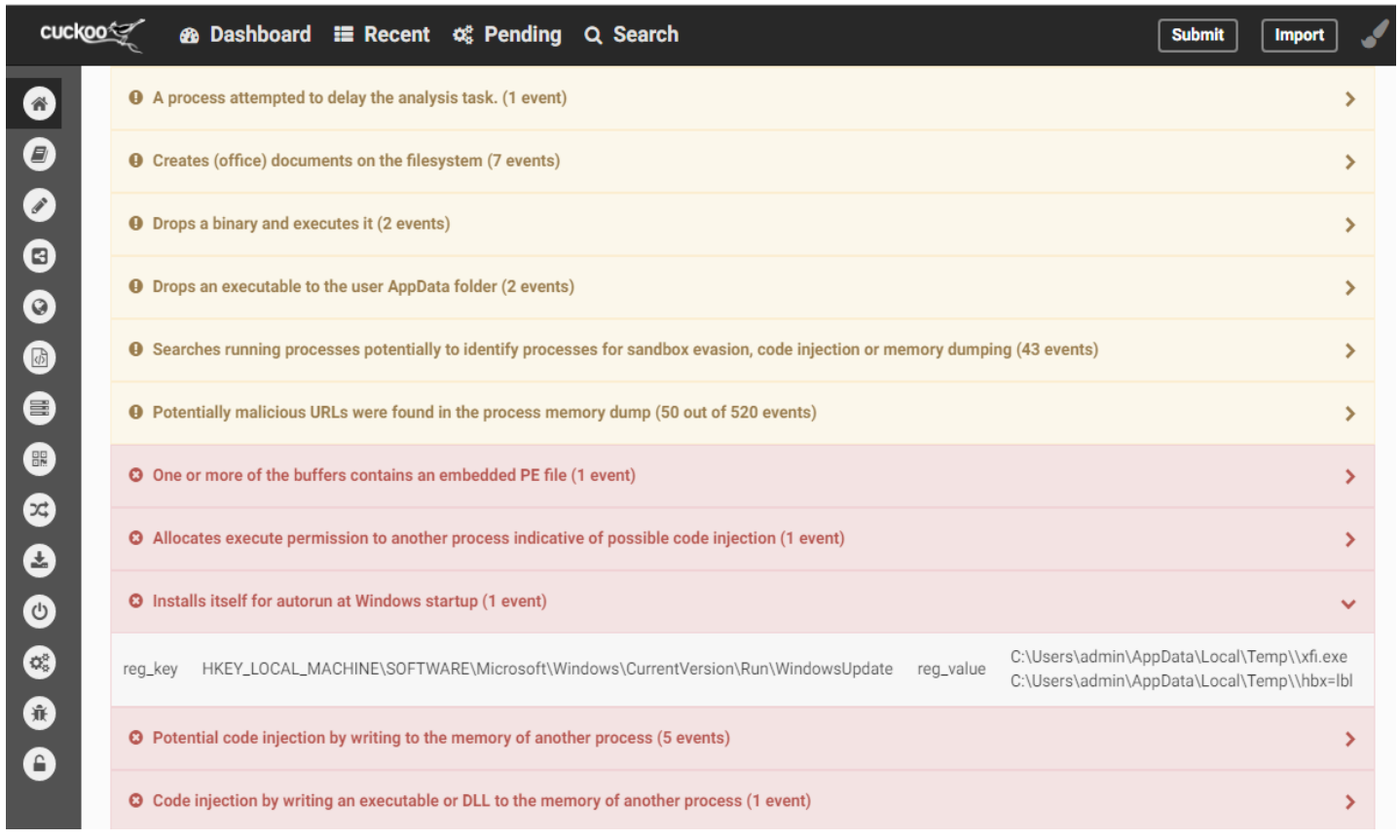
Clicking on *Analyze*, on top right, you can start the analysis.



Your analysis passes through several phases: *pending*, *running*, *completed* and *reported*. Once it is *reported*, you can click on it and go to the report view.

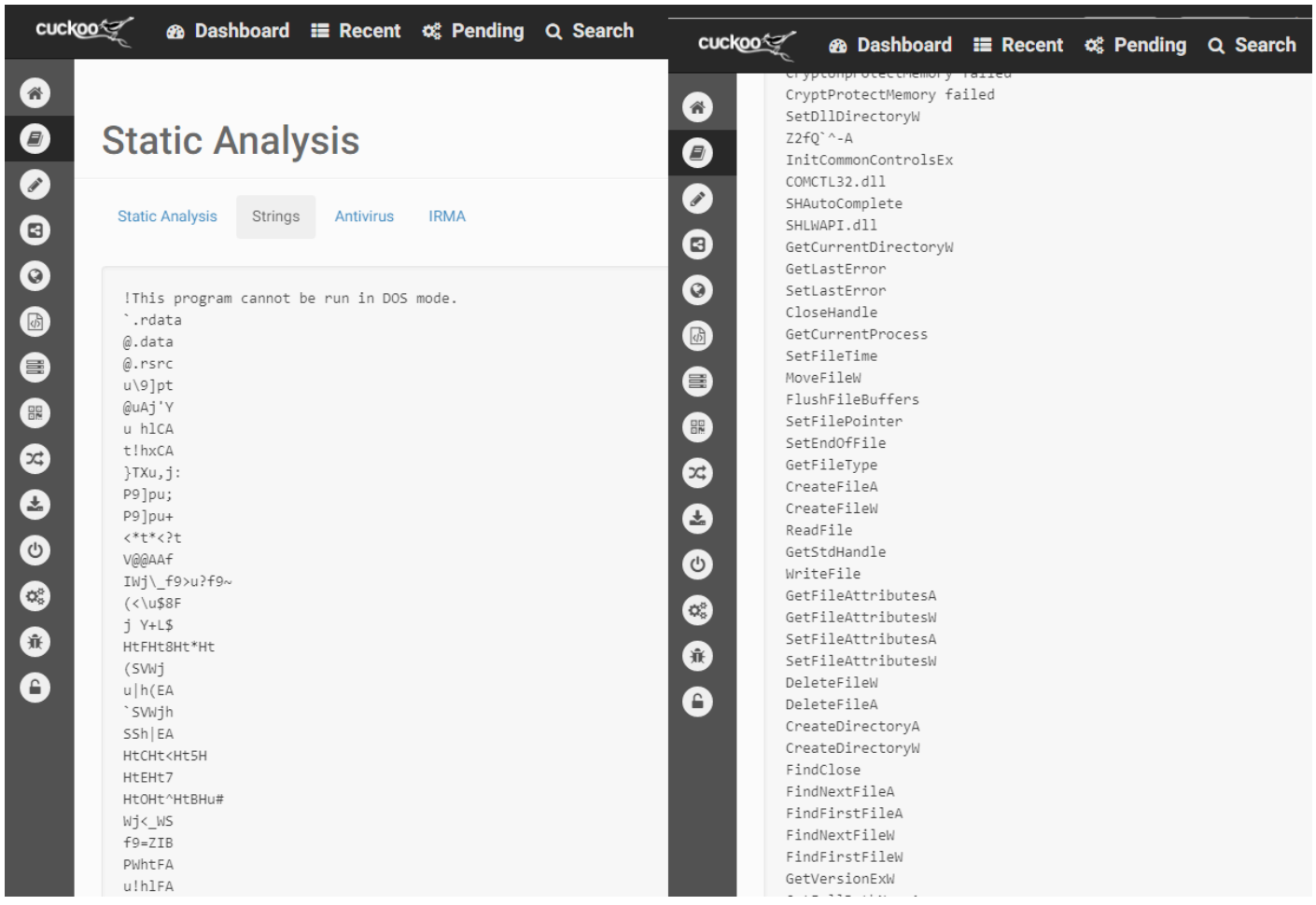


The first page you can see shows a summary of the analysis: the information related to the submitted file, such as its size, its hashes (md5, sha1, sha256) used to univocally identify it and a score indicating the dangerousness of the sample. Moreover, the summary includes many signatures that briefly identify the malware behavior: each signature is a potentially dangerous action executed by the analyzed file. In the following image, you have suspicious signatures in yellow and malicious ones in red. Among them, you can see the red signature *"Install itself for autorun at Windows startup"*, which means that the malware uses some technique to set its persistence; in this specific case, it writes the registry key `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\WindowsUpdate` setting it with malicious file path *"`%TEMP%\xfi.exe`"*.



On the left, you have the navigation menu, which includes several sections:

- *Static Analysis* includes all the information extracted analyzing the file without running it: the strings contained in the file can be an example of information extracted in a static way.



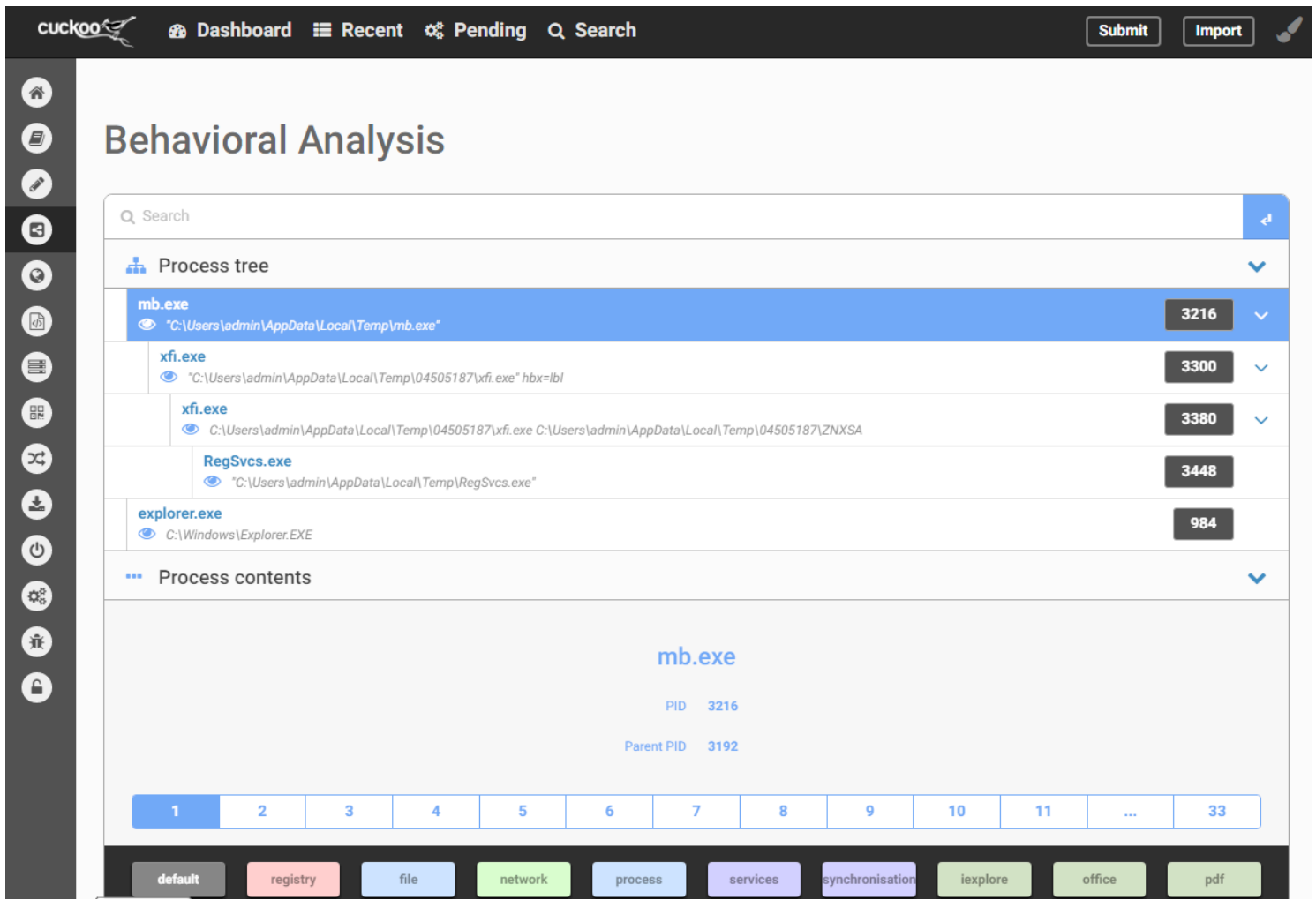
Analyzing the strings list, you can find interesting information like system libraries imported (i.e. *COMCTL32.dll*, *SHLWAPI.dll*) or some API functions that will be invoked during the malware execution (i.e. *ReadFile*, *WriteFile*, *GetFileAttributes*).

- *Extracted Artifacts* section contains all the artifacts identified by Yara rules during the analysis, such as other malicious scripts or code not stored in the file system but operating directly from memory.

- *Behavioral Analysis* is probably the most interesting section; it includes the malicious execution flow and all the system functions invoked by the malware. In the following screen, you have first the process tree created by the analyzed executable:

```
Mb.exe -> xfi.exe -> xfi.exe -> RegSvcs.exe
```

Cuckoo also reports the *explorer.exe* process, despite it being a legit system process. It is not unusual that the sandbox includes in the report some information related to legit artifacts. This problem derives from the difficulty that all sandboxes have to isolate the single malicious process. So, the analyst's task is to discriminate only the useful information.



Using the buttons displayed on the bottom of the above image, you can filter the API functions invoked by the malware depending on their type: functions operating on registry, on file, network activities, etc.

In this specific case, filtering by *registry,* Cuckoo shows the malicious activity already encountered in the *Signatures* section of the Summary page:

RegCreateKeyExW Jan. 8, 2019, 8:54 p.m.	regkey_r: SOFTWARE\Microsoft\Windows\CurrentVersion\Run base_handle: 0x80000002 key_handle: 0x00000114 class: options: 0 access: 0x00020006 disposition: 2 regkey: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	1	0	0
RegSetValueExW Jan. 8, 2019, 8:54 p.m.	key_handle: 0x00000114 regkey_r: WindowsUpdate reg_type: 1 (REG_SZ) value: C:\Users\admin\AppData\Local\Temp\\xfi.exe C:\Users\admin\AppData\Local\Temp\\hbx=lbl regkey: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\WindowsUpdate	1	0	0
RegCloseKey Jan. 8, 2019, 8:54 p.m.	key_handle: 0x00000114	1	0	0

You can see the complete flow of the malicious activity:

1. The malware searches the regkey

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

using the *RegCreateKeyExW* function.

2. It writes a new key

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WindowsUpdate setting as value the paths to other malicious files "`%TEMP%\xfi.exe %TEMP%\hbx=lbl`".

3. It closes the handle related to regkey.

You can filter by *file* and get all the files read or created by the malware, as shown below.

FindFirstFileExW Jan. 8, 2019, 8:54 p.m.	filepath_r: eau.ppt filepath: C:\Users\admin\AppData\Local\Temp\04505187\eau.ppt		4294967295	0
GetFileAttributesW Jan. 8, 2019, 8:54 p.m.	file_attributes: 4294967295 filepath_r: eau.ppt filepath: C:\Users\admin\AppData\Local\Temp\04505187\eau.ppt		4294967295	0
NtCreateFile Jan. 8, 2019, 8:54 p.m.	create_disposition: 5 (FILE_OVERWRITE_IF) file_handle: 0x00000154 filepath: C:\Users\admin\AppData\Local\Temp\04505187\eau.ppt desired_access: 0x40100080			
	(FILE_READ_ATTRIBUTES|SYNCHRONIZE|GENERIC_WRITE) file_attributes: 0 () filepath_r: eau.ppt create_options: 96 (FILE_NON_DIRECTORY_FILE|FILE_SYNCHRONOUS_IO_NONALERT) status_info: 2 (FILE_CREATED) share_access: 1 (FILE_SHARE_READ)	1	0	0

C:\Users\admin\AppData\Local\Temp\04505187\eau.ppt is another malicious file used by the malware to contain part of its payload.

- *Network Analysis* contains all the network traffic intercepted by Cuckoo, also including non-malicious one.

Network Analysis

Download pcap

Hosts 3 | DNS 3 | TCP 1 | UDP 20 | HTTP(S) 1 | ICMP 0 | IRC 0 | Suricata | Snort

IP Address	Status	Action
192.3.162.161	Active	Moloch
23.62.99.27	Active	Moloch
8.8.8.8	Active	Moloch

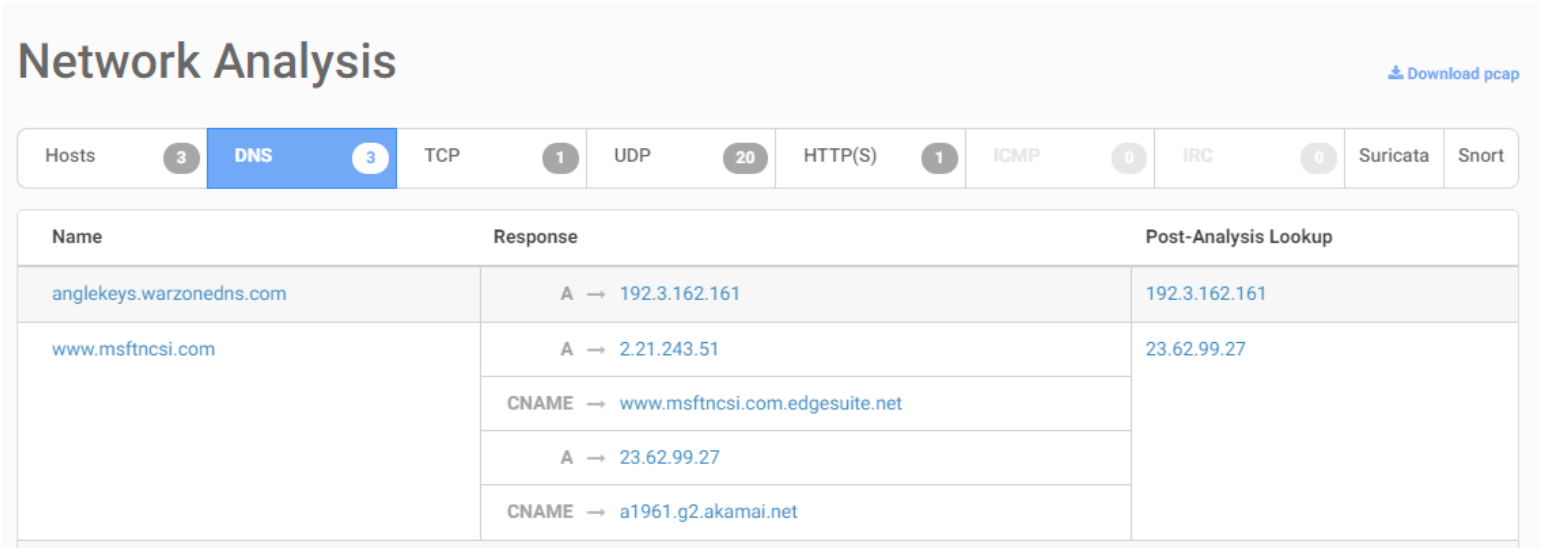
Using the tabs, you can filter by traffic type and protocols. The *Hosts* tab contains only a few bits of information; in fact, it shows only the contacted hosts' IP Addresses:

- 8.8.8.8 is legit Google DNS;

- 23.62.99.27 is related to Microsoft's servers;

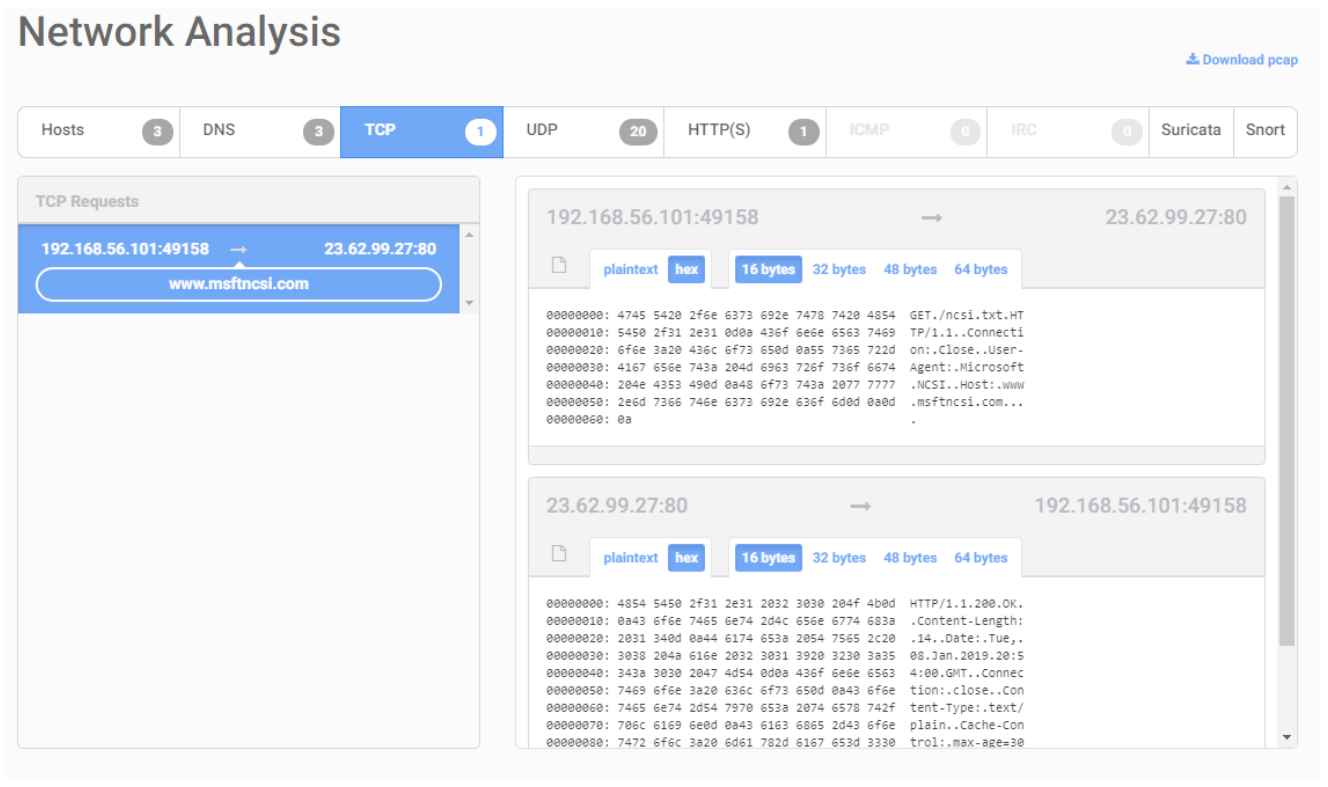
- 192.3.162.161 is probably related to some malicious server.

You can have a confirmation using the *DNS* tab:



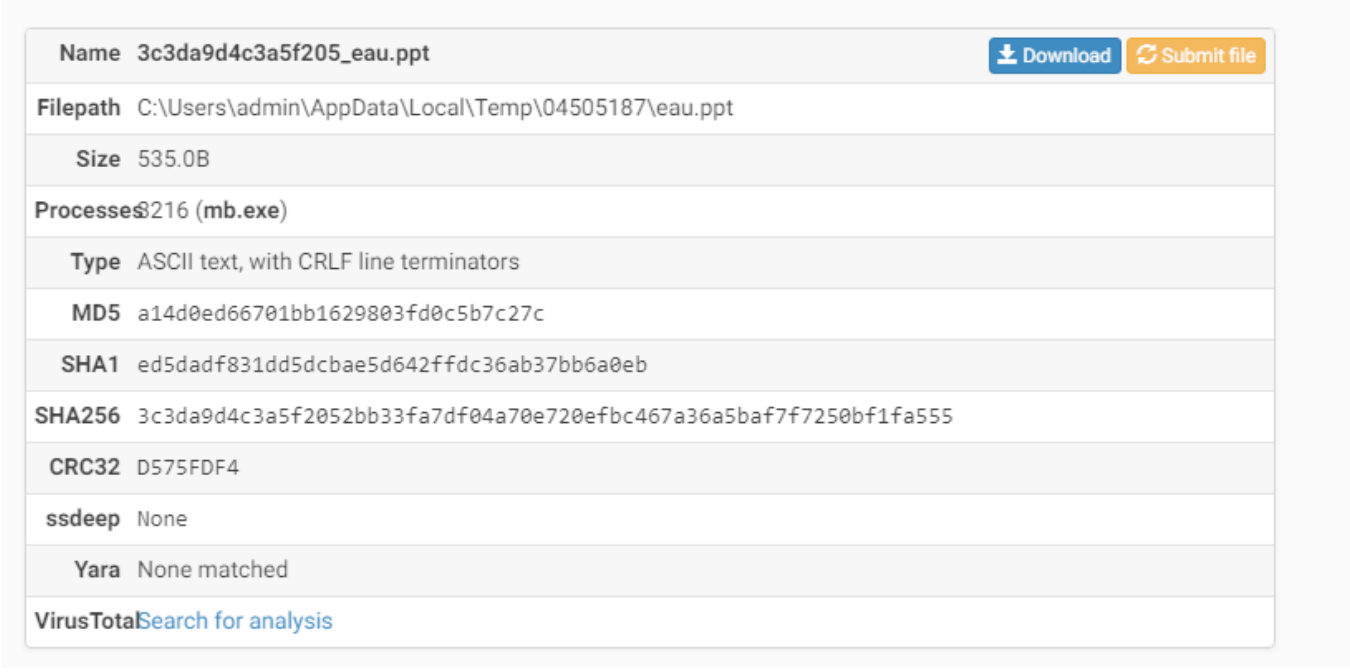
So, 192.3.162.161 is related to *anglekeys.warzonedns.com* that is a suspicious URL and probably it is the malware's C2C.

The following tabs can show you the data contained in network requests:



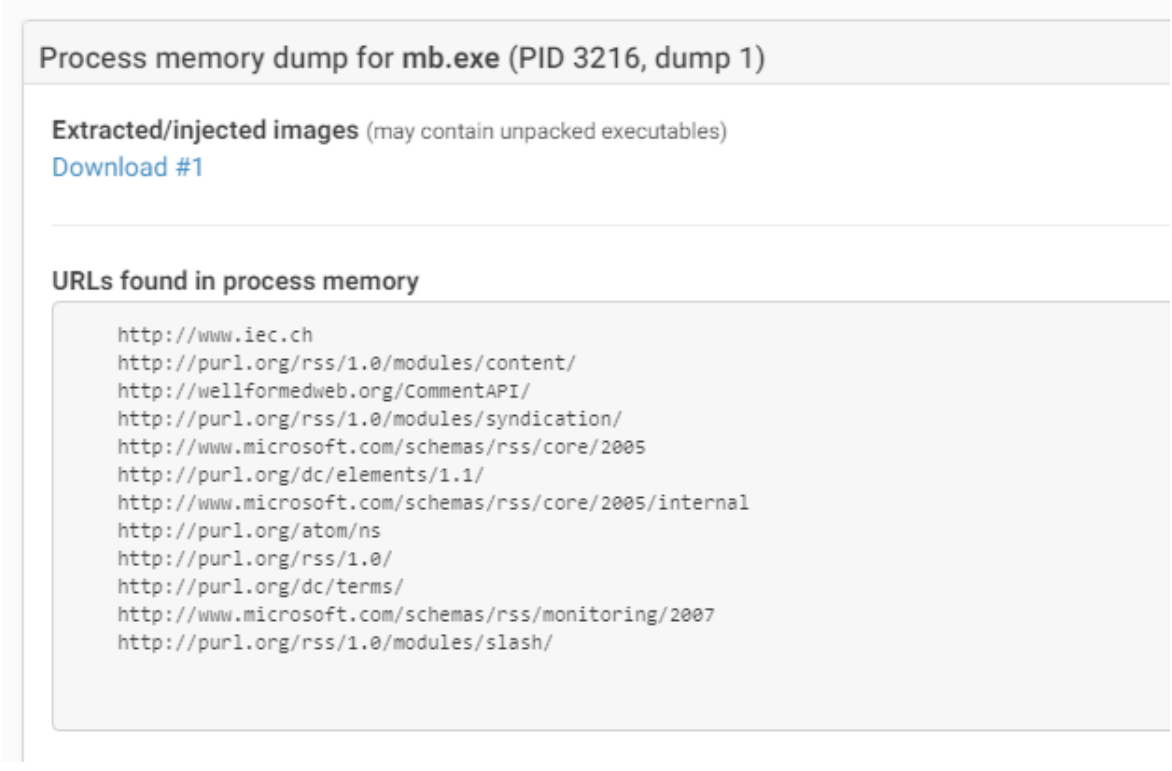
In the image, you can see the stream request and response from the machine to Microsoft's server and vice versa; obviously this network traffic is legit.

- *Dropped Files* contains a list of all files created by malware. They can be downloaded to proceed with manual analysis or directly submitted to Cuckoo Sandbox for a specific automated analysis. In the screen below, you can view the file *"eau.ppt"*, already encountered during behavioral analysis.



-*Dropped Buffers* section includes all suspicious memory buffers allocated during the malware execution.

-*Process Memory* reports the results of the process memory inspection through some tools like STRINGS, YARA and Volatility. In this case, Cuckoo has extracted from process memory some URLs.



-*Compare Analysis* allows you to compare two different analyses of the same sample. It is useful to discover malware behavior variations related to modifications of machine configuration.

-*Export Analysis* is used to export an analysis in Cuckoo format. The exported analysis can be imported into another Cuckoo instance using the button "Import".

-*Reboot Analysis* repeats the analysis of the same sample.

Applications

From the previous section, it is easy to understand the powerful of Cuckoo Sandbox, able to detonate and analyze in-depth even the most dangerous malware, summarizing all the information into a few, simple, sections. Moreover, thanks to its user-friendly interface, Cuckoo Sandbox can be used by a multitude of users, technicians and others. In addition to malware analysts, Red Teaming operators can also exploit the platform to test the effectiveness and stealthiness of malware they craft to conduct their advanced penetration tests. In that way, they can understand if their own malware reaches its goal (i.e. to exfiltrate information or open a backdoor in the system) and what evidence it leaves in the system during its intentionally malicious actions. And again, an enterprise could build up its own sandbox, based on Cuckoo, to analyze suspicious files that are able to exceed the perimetral protection through mail attachments or unaware downloads by users. So, the sandbox can be visible to all of an enterprise's employees in order to allow them to upload any strange files, otherwise, the file submissions can be automated using a probe that intercepts network traffic (including downloads and email attachment) and sends all files belonging to a specific type (i.e. executable files) to Cuckoo, alerting when a dangerous one emerges. Finally, a Cuckoo Sandbox instance can be publicly exposed by a security firm or an independent threat researcher as a free service to the rest of the world, allowing the people to upload any suspicious file. In this case, who offered the service can obtain every submitted sample, using them to study new threats, malware trends or to create his private malware repository.

About the Author

Antonio Farina has master degree in Software Engineering at University of Sannio (Italy). During his thesis, he studied the malware's anti-sandboxing techniques and implemented a new approach to contrast them, through the runtime modification of API calls. Since 2017 he is a member of ISWATLab - University Of Sannio Software Security Lab, a research center focused on software vulnerabilities and new malware identification approaches. Since 2018, Antonio is part of ZLab, the malware analysis team of Yoroi – Cybaze, a leading security company in Italy. In ZLab, he wrote more than twenty malware analysis reports until now, related both Advanced Persistent Threat and CyberCriminal group.

Things to know about Buffer Overflow

by Sibi Chakkaravarthy Sethuraman & Deepsagar Mandal

A buffer overflow occurs when the data that is written into the buffer exceeds the allocated space and results in the overwriting of adjacent memory locations. Most of the present day systems have a very similar memory layout. This causes a serious problem as a very simple piece of vulnerable code, if neglected, can possess a grave problem. Security attacks using buffer overflow are fairly common and most of them seek to modify data in the memory, gain access to confidential data and many more similar exploits.

What is it?

A buffer is a storage space for data that is available temporarily during the execution of a program. A buffer is located in the physical memory (typically the RAM (Random Access Memory)) of the computer. It is a very commonly used data structure.

A buffer overflow occurs when the data that is written into the buffer exceeds the allocated space and results in the overwriting of adjacent memory locations. Most of the present day systems have a very similar memory layout. This causes a serious problem as a very simple piece of vulnerable code, if neglected, can possess a grave problem. Security attacks using buffer overflow are fairly common and most of them seek to modify data in the memory, gain access to confidential data and many more similar exploits.

Overflow

To understand the concept of overflow we can make use of the well-known term, **'Integer Overflow'**. In a given n bit memory space, we can use only numbers in the range of 0 to (2^n)-1. This is universally true for unsigned integers. However, if we were to store signed numbers in the same amount of memory, the range will now become -2^(n-1) to (2^(n-1))-1. For example, in an 8-bit memory space, we can store the numbers 0 to 255. If we want to store signed numbers however, the new range happens to be -128 to 127. This is demonstrated below using a C program.

integer_overflow.c

```c
#include<stdio.h>

int main() {
    char n1 = 100, n2 = 200, sum;
    sum = n1+n2;
    unsigned char un1 = 100, un2 = 200, usum;
    usum = un1+un2;
    signed char sn1 = 100, sn2 = 200, ssum;
    ssum = sn1+sn2;
    printf("n1: %d n2: %d sum: %d \n", n1,n2,sum);
    printf("un1: %d un2: %d usum: %d \n", un1,un2,usum);
    printf("sn1: %d sn2: %d ssum: %d \n", sn1,sn2,ssum);

    return 0;
}

```

Figure 1. Sample code – integer overflow

The program has character variables of the type default, signed and unsigned. A character variable in C language is allotted a space of 1 byte, that is 8 bits. To these 8 bits, we assign the values of 100 and 200 and try to compute their sum in different variables of the character data type.

```
root@itdsm:~# gcc -g integer_overflow.c -o integer_overflow
root@itdsm:~# ./integer_overflow
n1: 100 n2: -56 sum: 44
un1: 100 un2: 200 usum: 44
sn1: 100 sn2: -56 ssum: 44
```

Figure 2. Results – integer overflow

From the above output, we conclude a few things; the default character data type gives the same output as that of signed character data type. This means that by default, in C, character data type is

signed. What is more interesting to see is that the signed variations have the second number stored as -56 instead of 200, which we have assigned in the program. This is not the case, however, in the unsigned variation. This proves the fact that in 8 bits the range of numbers that can be stored is -128 to 127 in signed versions and 0 to 255 in unsigned versions. In fact, what has happened here is that binary representation of 200 happens to be 1100 1000. As it exceeds the range of the number which is taken as 2's complement of the number -56, which also happens to be 1100 1000, thus the number is considered to be -56 and sum is calculated as 100-56 = 44. In the signed variation, although 200 is in range of the numbers allowed, the sum of the numbers 100+200 = 300 exceeds the range. The binary representation of 300 is 0001 0010 1100. However, as char data type only allows 8 bits, only the lowest 8 bits are taken. 0010 1100 happens to be the binary representation of the number 44 which is in range. This is a simple and commonly known example of overflow in the context of binary operations.

Common math operations such as addition, multiplication, division, shifts, etc., can produce a result that is too large to store, resulting in an integer overflow. This vulnerability can be exploited in various ways. For example, if a buffer was being allocated in memory based on an integer variable that happens to suffer from an integer overflow and the value of the integer becomes negative, a negative amount of space cannot be allocated to a buffer and the application, therefore, ends up crashing. This just happens to be one of the many ways how a denial of service can be carried out on an application. If we know the current maximum size of the integer variable, we can even overflow the integer to a specific value that we desire. This new buffer can later be vulnerable to a buffer overflow even if steps are taken by the programmer to prevent data larger than the original integer value to be entered into the buffer. Other exploits of integer overflow even include ways of allowing execution of arbitrary harmful code that might result in a complete takeover of the system. We will now see how buffer overflows are carried out.

A Simple Exploit

Now that we understand the terms 'buffer' and 'overflow'. Let us try to simulate a simple exploit that might lead to harmful consequences in the real world. This time we shall be using a C++ program to tinker around the memory and get some clarity on the concept.

buffer_overflow.cpp

```
1  #include <iostream>
2  #include <string.h>
3  using namespace std;
4
5  int main() {
6      bool auth = false;
7      char username[10];
8      char password[10];
9
10     cout<<"Before accepting values"<<endl;
11     cout<<"username addr: "<<&username<<endl;
12     cout<<"username value: "<<username<<endl;
13     cout<<"password addr: "<<&password<<endl;
14     cout<<"password value: "<<password<<endl;
15     cout<<"auth addr: "<<&auth<<endl;
16     cout<<"auth value: "<<auth<<endl;
17
18     cout<<"Enter Username :";
19     cin>>username;
20     cout<<"\nAfter accepting username"<<endl;
21     cout<<"username value: "<<username<<endl;
22     cout<<"password value: "<<password<<endl;
23     cout<<"auth value: "<<auth<<endl;
24
25     cout<<"Enter Password :";
26     cin>>password;
27
28     cout<<"\nAfter accepting both username and password"<<endl;
29     cout<<"username value: "<<username<<endl;
30     cout<<"password value: "<<password<<endl;
31     cout<<"auth value: "<<auth<<endl;
32
```

Figure 3a. Sample code – variable (Buffer) - initialization

```
32
33     if(strcmp(username,"root")==0 && strcmp(password,"pass")==0) {
34         auth = true;
35     }
36     if(auth) {
37         cout<<"Access Granted"<<endl;
38     }
39     else {
40         cout<<"Wrong username or password"<<endl;
41     }
42     return 0;
43 }
44
```

Figure 3a. Sample code – Buffer overflow (stack)

The above program is a simple authentication tool that grants access to a user if the username and password he or she enters are the correct values for the respective buffers. The 'username' string is stored in a character array of size 10 bytes and 'password' string is also stored in a character array of size 10 bytes. A Boolean variable 'auth', which is by default set to false, is used in the authentication process. As per line 33 of the code segment shown above, it can be seen that if the values entered in

username and password are ‘root’ and ‘pass’, the value of auth variable is changed to true allowing access to the user. Let us look the execution of the program.

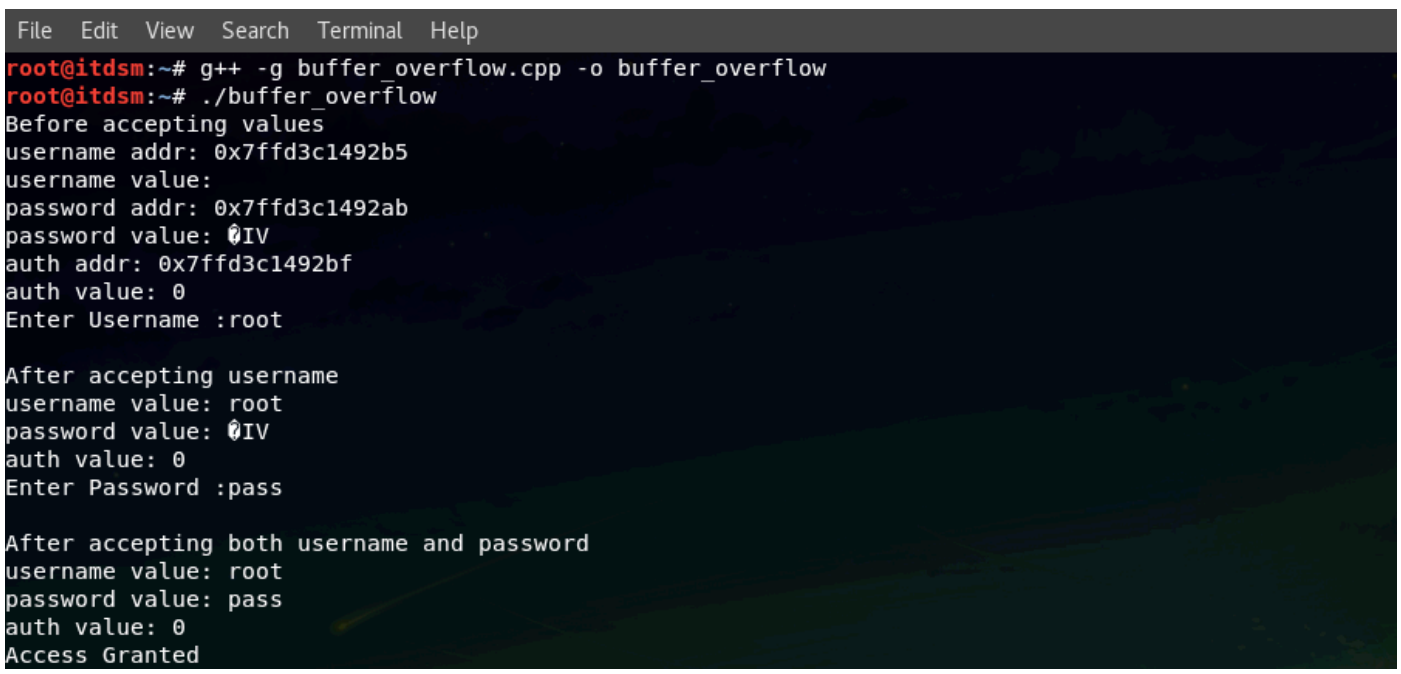


Figure 4. Buffer overflow - output

It can be seen from the above snippet that the value passed to the ‘username’ buffer is ‘root’ and that passed to the ‘password’ buffer is ‘pass’. The program executes properly and grants access to the user. In addition to these, we also have print statements showing the starting addresses of the buffers, username and password. The address at which auth variable is stored is also printed.

Note: The auth value print at the end is 0. This is because the print statement is located before the ‘if’ segment where we validate the buffers. The auth variable is changed to true within this segment, thus granting access to the user.

Now let us execute this program again. This time we shall enter some random values in the buffers.

```
root@itdsm:~# ./buffer_overflow
Before accepting values
username addr: 0x7ffeaf298ec5
username value:
password addr: 0x7ffeaf298ebb
password value: ⌂U
auth addr: 0x7ffeaf298ecf
auth value: 0
Enter Username :random@123

After accepting username
username value: random@123
password value: ⌂U
auth value: 0
Enter Password :morerandom

After accepting both username and password
username value:
password value: morerandom
auth value: 0
Wrong username or password
```

Figure 5a. Buffer overflow - validation

As expected, since the values entered are 'random@123' and 'morerandom' for username and password, respectively, the authentication fails. The message says 'Wrong username or password'. Note, however, that the values entered are within range of 10 bytes which we had declared previously in our code segment. Let us execute the program again, this time we shall enter random values with a twist. The values for both the buffers will exceed their defined range of 10 bytes.

```
root@itdsm:~# ./buffer_overflow
Before accepting values
username addr: 0x7fffd0ee2c85
username value:
password addr: 0x7fffd0ee2c7b
password value: @!V
auth addr: 0x7fffd0ee2c8f
Enter Username :averylongrandomstring

After accepting username
username value: averylongrandomstring
password value: @!V
Enter Password :anotherverylongrandomstring

After accepting both username and password
username value: ylongrandomstring
password value: anotherverylongrandomstring
Access Granted
Segmentation fault
```

Figure 5b. Buffer overflow - validation

Interestingly enough the program grants access to the user. Along with an 'Access Granted' message, the output also has a message that says 'Segmentation Fault'. What does this mean and how did entering some large random values, which are definitely not 'root' and 'pass', grant access to the user? Before trying to find out these answers, let us look at the memory layout of C programs. The process is divided into the sections kernel, stack, heap, global/static data and machine code. These sections are arranged as shown. The stack and heap memory grow opposite to each other. The stack grows towards the lower address space, while the heap grows towards the higher address space. As we know, the stack is used for static memory allocation while the heap is used for dynamic memory allocation. Since we are not dynamically allocating our buffers, both username and password are stored in stack. The auth Boolean variable is also stored in stack. If we look at the output above, we can see that the starting address of username is 0x7fffd0ee2c85 and that of password is 0x7fffd0ee2c7b.

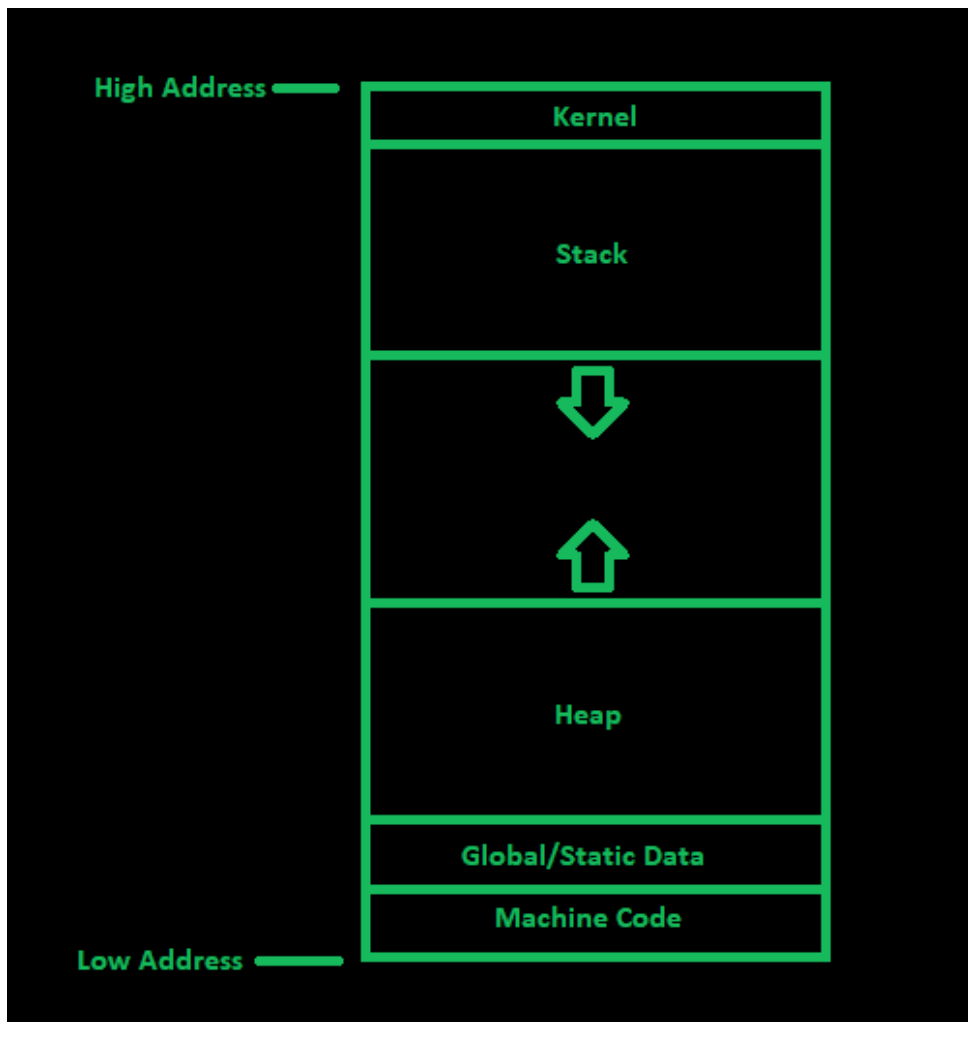


Figure 6. Memory organization

The difference of these two hexadecimal numbers is 0x7fffd0ee2c85-0x7fffd0ee2c7b = 0xa or 10 in decimal form. This means that the two buffers located adjacent to each other as they are each of size 10 each. The auth variable is located 10 spaces above the username as 0x7fffd0ee2c8f-0x7fffd0ee2c85 = 0xa where 0x7fffd0ee2c8f is the address of the auth variable. For better representation and understanding, let us overload the buffer again, filling the character 'u' when prompted for username and 'p' when prompted for password.

```
root@itdsm:~# ./buffer_overflow
Before accepting values
username addr: 0x7fff218852d5
username value:
password addr: 0x7fff218852cb
password value: ??U
auth addr: 0x7fff218852df
auth value: 0
Enter Username :uuuuuuuuuuuuuuuuuuuu

After accepting username
username value: uuuuuuuuuuuuuuuuuuuu
password value: ??U
auth value: 117
Enter Password :pppppppppppppppppppppppppppp

After accepting both username and password
username value: pppppppppppppppppppp
password value: pppppppppppppppppppppppppppp
auth value: 112
Access Granted
Segmentation fault
```

Figure 7. memory during execution

In the execution above, we have entered the character 'u' 20 times when asked for username and 'p' 25 times for password. Here is a visual representation of how these are stored in the buffer. As shown in the figure, filling the username field with 20 u's results in overflow of the characters causing them to write to the adjacent memory location which happens to be the location of the auth variable. From the snippet of the execution above, it can be seen that the value of auth is 117 which is the ASCII value of the character 'u'. As we enter an even greater amount of 'p' characters in the password field, it not only overwrites the username field but also the auth field. Referring to the output shown above, it can be seen that the value if auth variable, after accepting the password, is 112, which is the ASCII value of character 'p'. This means that although the if block at line 33 is not entered, the if statement at line 36 still passes as C++ takes any value greater than 0 as 'true'. This results in the program ultimately granting access to the user. With that, we have now understood the workings of a simple buffer

overflow exploit in a simulated environment. It is very important to avoid such programming errors. Some programming languages are more secure than others. Java, for example, runs everything in a Java Virtual Machine, which handles such exceptions by itself. However, if performance is a concern, it is important to pay attention and handle these easily avoided vulnerabilities yourself.

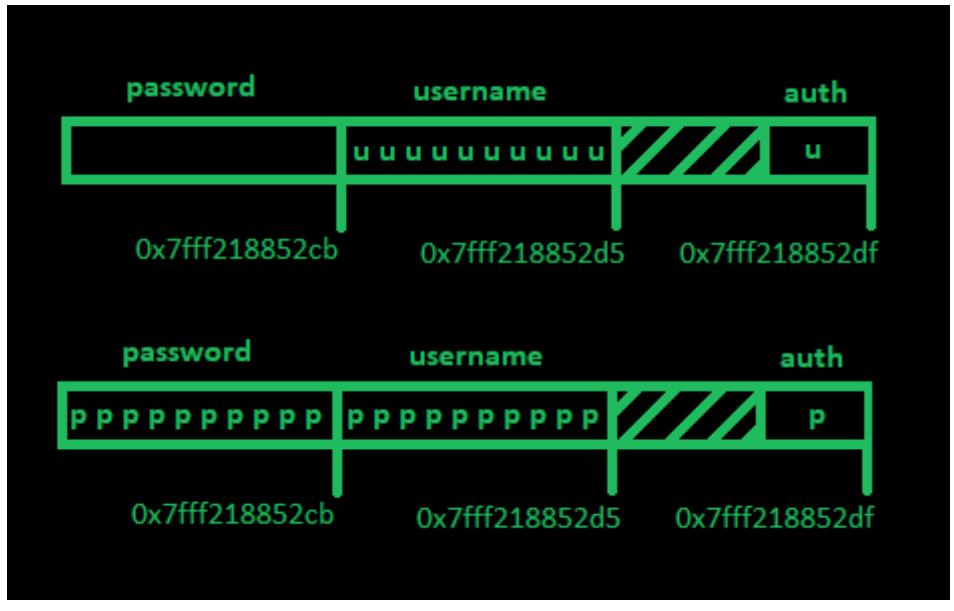


Figure 8. Memory organization during overflow

About Deepsagar

Deepsagar Mandal is a sophomore student of School of Computer Science and Engineering, Vellore Institute of Technology – Andhra Pradesh (VIT-AP). His research interest includes Network security, malware analysis etc.

About Sibi

Sibi Chakkaravarthy Sethuraman is currently working with School of Computer Science and Engineering, Vellore Institute of Technology – Andhra Pradesh (VIT-AP). He is also the coordinator for Artificial Intelligence and Robotics (AIR) Centre at VIT-AP. The author holds Ph.D. from Anna University and he is the recipient of DST Fellowship. His research interests include drone security, network security, malware analysis, AI based adversarial modelling, etc.

Cuckoo-SandBox Detection & Bypassing

by Mohammed Ali

Today, a lot of malware contains anti-SandBox, anti-virtualization with a lot of techniques that allow the malware to have an initial background of what type of system that's running on it, and take an action. This is one of the critical disadvantages of SandBox, but it's just come in the second rate. The first disadvantage is "Time", that's what makes SandBox not useful for most companies, i.e., "Who will wait for 30~120 seconds for each file who download, or access it on the internet?" Unfortunately, we have nothing to do with this issue "resources & quality have a positive relationship", but we can deal with the second one with some tricks.

Cuckoo-SandBox

Is an open source automated malware analysis system, allowing you to submit the files into a preconfigured virtual machine (Windows, Linux or Darwin) and check, and note the changes that occur on the system for a predefined time period, after executing the file.

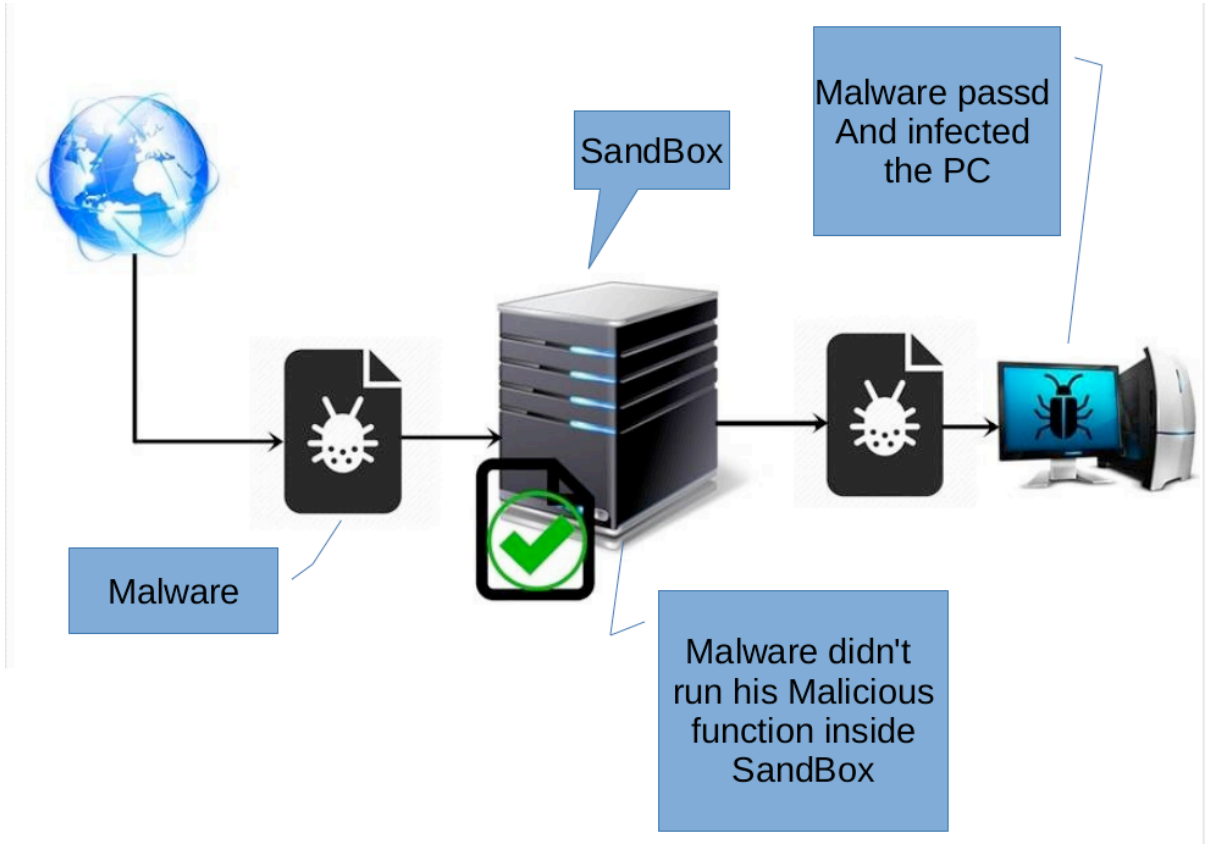
BACKGROUND

Despite how great Cuckoo-SandBox is, it has a lot of weaknesses that can be easily exploited by any programmer.

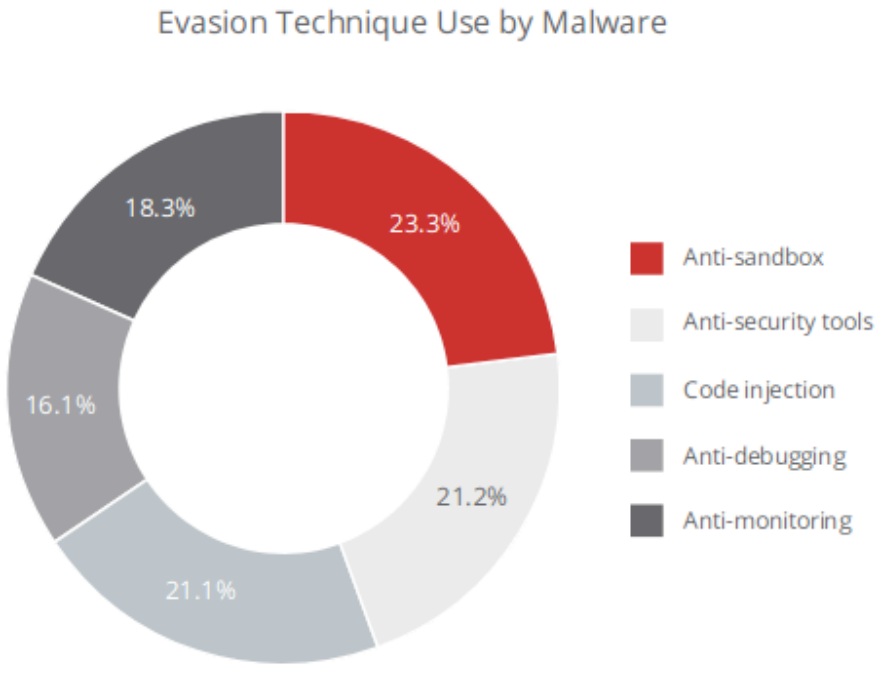
Today, a lot of malware contains anti-SandBox, anti-virtualization with a lot of techniques that allow the malware to have an initial background of what type of system that's running on it, and take an action. This is one of the critical disadvantages of SandBox, but it's just come in the second rate.

The first disadvantage is "Time", that's what makes SandBox not useful for most companies, i.e., "Who will wait for 30~120 seconds for each file who download, or access it on the internet?" Unfortunately, we have nothing to do with this issue "resources & quality have a positive relationship", but we can deal with the second one with some tricks.

The following image shows us how the bypass is done,



It is more complicated when we come to the statistics of malware that contain "anti-SandBox". Here is a graph published by McAfee, showing us the rise in SandBox evasion, against other evasion techniques.



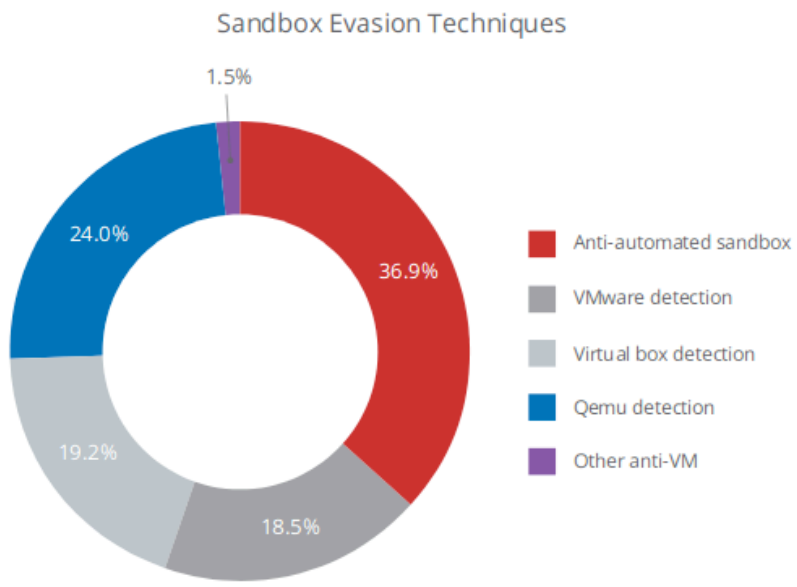
Detection

TYPES OF EVASION:

1. Time Manner. 2. Windows Registry. 3. MAC Address. 4. System Username. 5. Number Of Process. 6. User Files.	1. Processes Names. 2. Windows Files. 3. Startup Time. 4. Disk Size & Ram Size. 5. RedPill & BluePill. 6. BackDoor I/O

And a lot of other techniques that can be used to bypass SandBox, whatever the bypass type used, as long as it’s doing what it's targeted for.

The following statistics, from VirusTotal and McAfee, are derived from samples known to contain sandbox evasion techniques.



Let's take a short look about each of the techniques above.

1. Time manner.	This is the most common way, and the easiest, to bypass SandBox, i.e. "make the app delay for a time before running malicious function", the most time it delays is for 60 seconds.
2. Windows Registry	The Virtual-Box Windows registry will find: "HKLM\SYSTEM\ControlSet001\Services\VBox " and the same for VMware: "HKLM\SYSTEM\CurrentControlSet\Enum\IDE\DiskVMware_Virtual_IDE_Hard_ Drive_____00000001\30303030303030303030303030303030303030303130\FriendlyName VMware Virtual IDE Hard Drive" "HKLM\System\CurrentControlSet\Control\Class\{4D36E968-E325-11CE-BFC1-08002BE10318}\0000\DriverDesc VMware SVGA II" and a lot of other registry keys. It's a simple and most common way for detecting virtualization.
3. MAC Address	Is the easiest way to the detect a virtual machine by checking the MAC address of this machine it will always start with in VBox: "08:00:27" in VMware "00:05:69", "00:0C:29", "00:1C:14", "00:50:56".
4. System UserName	Malware also checks if UserName is "cuckoo", "malware", "SandBox", "sample".
5. Number of process	Malware counts the running processes to check if it is the only running process, and point out if it's a SandBox machine.
6. User Files	Malware checks inside files under user "Documents", "Downloads" ... etc., for files to check if it's an active user or fake.
7. Processes Names	Check on the running processes names, for example, process "vmtoolsd.exe", etc.

8. Windows Files	Checking under C:\\windows\\system32 for executable files that start with string "vbox". Checking under C:\\windows\\system32\\drivers for files that start with string "VBox" The same for VMware, just change the string to "vmware".
9. Startup Time	Some malware use the machine time to check if this machine has been working for a while or if it just started.
10. Disk Size & RAM Size	Some malware check for RAM size and disk size to make sure it's not running under SandBox virtual machine

11. Red Pill & Blue Pill	**Red Pill** is more or less equivalent to the following code (returns nonzero when in Matrix):
	`int swallow_redpill()`
	`{`
	`unsigned char m[2 + 4], rpill[] = "\x0f\x01\x0d\x00\x00\0x00\x00\xc3";`
	`*((unsigned *)&rpill[3]) = (unsigned)m;`
	`((void (*)()) & rpill)();`
	`return (m[5] > 0xd0) ? 1 : 0;`
	`}`
	Blue Pill is the codename for a rootkit based on x86 virtualization. The Blue Pill concept is to trap a running instance of the operating system by starting a thin hypervisor and virtualizing the rest of the machine under it.

12. Backdoor I/O port	VMware uses a virtual I/O port 0x5658 (VX in ASCII) for communication between the guest operating system and host operating system, used for copy-paste, etc., between guest and host machine.

Bypassing

Let's provide some tricks to deal with those issues. There are tricks can solve some of the above techniques, but unfortunately cannot be valid 100% for all cases. "Everything has an exception".

1. **Time manner**	**There is no radical solution to this problem, but we can deal with it by increasing Cuckoo-SandBox work time for the files that contain "sleep" function; here is a simple Python script.**
	`#!/usr/bin/env python`
	`import os`
	`SampleDir = "/home/bow/cuckoo-samples/"`
	`def CheckForDelay(sample):`
	`  sample = "%s" % (SampleDir + sample)`
	`  xfile = open(sample, "rb")`
	`  data = xfile.read()`
	`  if b"sleep" in data:`
	`  os.system(("cuckoo submit --timeout 120 %s" % (sample)))`
	`  else:`
	`  os.system(("cuckoo submit --timeout 30 %s" % (sample)))`
	`if __name__ == "__main__":`
	`  samples = os.listdir(SampleDir)`
	`  for sample in samples:`
	`  CheckForDelay(sample)`
	But unfortunately if the sleep function is packed or decoded it will not work.

2. **MAC Address**	Pretty easy to change it, from virtual machine network settings.
3. **System UserName**	Just do not use any name of the above for your virtual machine.
4. **Number Of Processes**	You can bypass it by just running more processes before taking a snapshot, you can run notepad, Internet-explorer, etc.
5. **User Files**	You can add some documents to your documents, download, desktop folder, also add more than three files to C:\\User\\userAppData\\Local\Temp, and C:\\windows\\temp, one more thing, the recent items must contain at least five files, so make sure you access some files.

6. Process Name	You can bypass it by just terminating this process, "you do not need it".
7. StartUp Time	You can take your snapshot after at least one hour, after starting your machine.
8. Disk & RAM Size	You can easily bypass it by making sure your RAM size is more than 4GB and your disk is more than 60GB.

9. RedPill	There are many ways you can deal with red-pill, but if we come to the easiest way is don't install the VMware tool as it will create a lot of artefacts (e.g. registry key, file, folder, etc.).
	Change the configuration of your virtual machine by adding the following options to your .vmx file:
	`isolation.tools.getPtrLocation.disable = "TRUE"`
	`isolation.tools.setPtrLocation.disable = "TRUE"`
	`isolation.tools.setVersion.disable = "TRUE"`
	`isolation.tools.getVersion.disable = "TRUE"`
	`monitor_control.disable_directexec = "TRUE"`
	`monitor_control.disable_chksimd = "TRUE"`
	`monitor_control.disable_ntreloc = "TRUE"`
	`monitor_control.disable_selfmod = "TRUE"`
	`monitor_control.disable_reloc = "TRUE"`
	`monitor_control.disable_btinout = "TRUE"`
	`monitor_control.disable_btmemspace = "TRUE"`
	`monitor_control.disable_btpriv = "TRUE"`
	`monitor_control.disable_btseg = "TRUE"`

Abstract, use the following steps to handle anti-sandbox,

1. Do not Install tools provided by the virtual machine in your guest OS.
2. Do not use "cuckoo", "sandbox", "malware", "sample" as username.
3. Run some process that will not affect the analysis.
4. Add some files to Desktop, Documents, Downloads, C:\\User\\userAppData\\Local\Temp, and C:\ \windows\\temp.
5. Change MAC Address of your virtual machine, it should not start with any of the above.
6. Increase your virtual machine disk to be more than 60 GB and increase your RAM to be more than 4GB.

7. Take some time working on your virtual machine, act like it's your personal machine before taking the snapshot.
8. Edit your Vmware .vmx file, by appending the configuration lines that I mention in the RedPill bypassing.
9. (OPTIONAL) Use Python script that I mentioned above to validate and submit your malware.

You can also use pafish to test your configuration, it's open source simulation to many techniques of bypassing virtual machine.

Conclusion:

There is a lot of anti-SandBox, "anti-Virtualization", etc., we cannot handle all of them, but we can deal with maybe 80% of malware by the above steps.

Ironically, these "smart ways" can be defeated, but there is little that you can do to prevent virtual machine detecting. Why?

Because a virtual machine that has its own characteristics: VGA adapter, IDE + HDD characteristics, ..., device drivers, for legacy guest operating system support, security mechanisms should not rely on obfuscation. They should rely on the intrinsic strength of the algorithms and architecture. You should assume that knowledgeable guests can detect that they are running in a virtual machine. This issue is whether they can do anything about it.

BIBLIOGRAPHY & CITATIONS:

- How red-pill works: www.ouah.org/Red_%20Pill.html
- Blue-Pill: en.wikipedia.org/wiki/Blue_Pill_(software)
- McAfee Malware Analysis without Sandbox: www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-jun-2017.pdf
- Prevalent Threats Targeting Cuckoo Sandbox Detection and Our Mitigation: www.fortinet.com/blog/threat-research/prevalent-threats-targeting-cuckoo-sandbox-detection-and-our-mitigation.html

- How does malware know the difference between the virtual machine and the real machine?: blog.talosintelligence.com/2009/10/how-does-malware-know-difference.html
- Backdoor I/O Port: sites.google.com/site/chitchatvmback/backdoor
- Avira Cuckoo Sandbox vs. Reality: blog.avira.com/cuckoo-sandbox-vs-reality-2/
- Trend Micro, how can Advanced Sandboxing Techniques Thwart Elusive Malware: www.trendmicro.com/vinfo/us/security/news/security-technology/how-can-advanced-sandboxing-techniques-thwart-elusive-malware
- Avira, VMCloak – Create a Virtual Machine the Easy Way: blog.avira.com/vmcloak-create-virtual-machine-easy-way/
- Pafish: github.com/a0rtega/pafish/tree/master/pafish

About Mohammed

I'm experienced in CyberSecurity and Malware Analysis (started my Penetration testing and programming journey when I was 12 years old). I'm dealing with languages such as Python, C/C++, PHP, Java, JavaScript. Three years ago I started focusing on Malware Analysis For Windows & Linux OS and working as a freelancer at UpWork. I'm doing my bachelor degree in "Information System & Management".