Global Information Assurance Certification Paper

Deployment of a Flexible Malware Sandbox Environment Using Open Source Software

GIAC (GCIH) Gold Certification

Author: Jose Ortiz, siltecon@gmail.com

Advisor: Richard Carbone

Accepted: July 18, 2015

Abstract

The identification and analysis of malware is one of the many tasks performed by incident handlers. Only a small number of commercial entities provide the technology capable of automating this. Most times these offerings are beyond the reach of small organizations due to the high costs associated with licensing and maintenance. One open source alternative is Cuckoo Sandbox. It is a free software project licensed under GNU GPLv3. It allows the user to analyze and collect data against suspected pieces of malware. The framework installation requires careful configuration by an experienced Linux administrator. The accepted method of deployment is to follow the prescribed steps and test the application "until it works." Attempting to scale the sandbox environment beyond a few virtual machines becomes a complicated process due to the maintenance required for multiple Windows configurations. By using techniques borrowed from the DevOps methods, a small team of incident handlers can create a sandbox environment that is not only repeatable and consistent, but also scalable. The user can create multiple template "profiles," which allow for flexible testing.

1. Introduction

Infection of a system with malicious code is part of the Kill Chain (Hutchins, Cloppert, & Amin, 2011) methodology. A downloaded payload is the step following an exploited vulnerable application. The purpose of this code is to give the attacker command and control (C&C) of the compromised system to expand access. This lateral movement allows for privilege escalation; and ultimately, the acquisition of high value information assets. By understanding the effects malware has on systems, defenders can create tactics and procedures to counteract the adversary.

One of the methodologies used to understand malicious code is sandboxing (Greamo & Ghosh, 2011). In simple terms, this process comprises the execution of malicious code in a controlled way that allows for direct observation of the effects. By documenting artifacts, like open ports, registry keys, IP address, dropped files, and domain names, a team can gain intelligence on the adversary's tactics and short circuit the kill chain. The best defender teams are intimately familiar with this concept and are able to cut the adversary at multiple stages of the chain. Doing so creates a resilient defense posture.

The implementation of a commercially supported sandbox comes with a hefty price tag and often times an annual support contract in the six or seven figure range. Multiple sandbox environments exist that are free and open source (Zeltser, 2015). Open source software is not free of cost either. An incident response team willing to implement open source software as an alternative to commercial offerings is left with 'community' and internal support. Deploying a sandbox is a very delicate process with many steps and pitfalls.

Several people have attempted to ease the pain of deploying a sandbox by creating scripts to take care of any manual steps. This takes care of the sandbox itself. Nevertheless, what about the targets? A typical enterprise with an incident response team can have dozens of custom applications, each with its own stack of database, web server, programming language, and web framework. The current methodology for testing malware is either to create a 'generic' computer target (lowest common denominator) or to spend multiple hours trying to replicate production environments in a testing lab.

Many solutions already exist to solve the problem of creating a sandbox using open source software (Zeltser, 2015). The main thesis of this paper focuses on creating the targets necessary for testing. By using configuration management and the concept of 'infrastructure as code' (Sabharwal & Wadhwa , 2014), new possibilities open up. A distributed team can collaborate on the creation of suitable targets. Incident response teams do not have to settle for generic

testing platforms, instead replicating enterprise systems closely. Defining infrastructure as code also opens the possibility of cloud technologies and throwaway instances; these are used only once and then deleted, thus releasing valuable computing and storage resources. It is not easy to distribute virtual machines based on proprietary operating systems on the Internet without violating intellectual property laws, but sharing the 'recipe' on how to build them is completely acceptable. This concept is the standard methodology proven to work in the Developer Operator arena and can definitely be leveraged for incident response teams.

2. Sandboxing

2.1. Malware Analysis

Incident handlers responding to an intrusion will at times use malware analysis in the course of incident response. Effective response requires proper implementation and execution of the identification, containment, eradication, and recovery phases (Pepe, Luttgend & Mandia, 2014). A product of the lessons learned during incident handling is the creation of signatures for host or network based intrusion detection and prevention systems. The proper signature creation will improve detection in case of future compromises.

One of these indicators is the file hash (Sikorski & Honig, 2012). Of all the indicators of compromise (IOCs), this is the weakest. It is very easy to alter a piece of code to get past a detection rule based on a hash. The complete understanding of malicious code requires disassembly and analysis of every function and system call executed. This process is extremely tedious, requires a high degree of expertise, and sometimes takes years to complete. According to the AV-TEST institute (AV-TEST, n.d.), in 2014 alone the total number of malware specimens released was over 140,000,000. Attempting to reverse engineer every single piece of malware is simply an impossible task.

A faster approach is to focus on the effects the piece of malware has on an affected system. Indicators like registry keys, dropped files, connections, DNS queries, and strings that point to a certain actor or persona. These indicators help in the creation of signatures. There is no need to open the binary for analysis.

Testing malware this way requires the code to run on a system to observe its behavior. This action will infect a system and could potentially make it unsafe. This is the reason malware is normally tested using an isolated system. However, isolating a machine could have unintended consequences for our analysis. What happens if the malware tries to create a DNS query and

fails? One solution is to create a 'black hole Internet.' The black hole will act as a sink for all activities the infected system takes. If the malware makes a DNS request for 'evil.com,' the black hole could respond with an IP address within our control. The black hole also hosts the most common services requested by malware, like HTTP, FTP, and SSL. One example of such a system is FakeNet (FakeNet, 2012).

2.1.1. Static Analysis

In static malware analysis, an incident response team analyzes the code or structure of a program to determine its function without running the program (Sikorski & Honing, 2012). The first steps include the use of any available anti-virus suite or service. This could give a clue for known malware for which a signature is available, saving time in the process. A major flaw of this technique is the reliance on anti-virus detection that relies mostly on file signatures. A team of malicious code authors can easily change the code to avoid detection (Dalziel, 2014). Ideally, one should attempt detection using different anti-virus programs. A good resource for achieving this without any major investment is VirusTotal (VirusTotal, n.d.). The web service is capable of scanning Windows binaries, Android APKs, PDFs, images, and JavaScript among other file formats. As of the current date, VirusTotal offers a free service to the community that uses over 50 different scanning engines.

File hashing is another method to identify malware. A good database for this type of information is located at the (Malware Hash Registry (MHR, n.d). This service is free for non-commercial use only. The MHR complements the anti-virus program by helping to identify known pieces of malware.

Other forms of identification using static methods include string analysis, DLL and function enumeration, and packed binary examination. Many times a piece of malware will contain strings that indicate an IP address or hostname. Malicious binaries often contain packing and encryption with the intent of obfuscating the content (especially strings).

2.1.2. Dynamic Analysis

Basic dynamic analysis is the follow on step to static analysis. In this phase, the malware executes and its behavior observed. The most popular method of running malware in a safe way is the use of sandbox technology. A sandbox runs as a separate system, contains the untrusted program, and prevents any action from reaching the real network. Sandboxes often provide network services to the malware in a 'black hole' fashion. If the untrusted program makes a DNS request for example, the sandbox will reply the answer, normally from 127.0.0.1 (loopback).

Jose Ortiz, siltecon@gmail.com

A number of sandbox environments exist already set up and ready to use. Some of these offerings are Anubis, BitBlaze, EUREKA, ViCheck (Zeltser, 2012). Most of these services are available free of cost; however, some organizations prefer hosting their own sandbox internally due to policy or other corporate requirement. Creating one's own sandbox environment could be an expensive proposition due to the many working hours required to install and configure many pieces of software.

One of the free software packages that allow an organization to create its own internal malware testing environment is Cuckoo Sandbox (Oktavianto & Muhardianto, 2013). Python scripting and libraries are its major components (Hosmer, 2014). The system consists of a host with the Cuckoo software installed, a virtualization provider such as Oracle VirtualBox (Dash, 2013), and the Cuckoo agent running inside a virtual machine guest. The guest will be a virtual machine running Windows XP or Windows 7. This virtual machine will have User Account Control (UAC), automatic updates, and firewall disabled..

3. Sandbox Automation

3.1. Configuration Management

A computer system with even a minimal amount of software is highly dynamic. The creation and effective management of computing resources need tools that go beyond in-house developed scripts and ad-hoc processes. A number of configuration management systems have emerged to solve this problem. The first scientist to create such a system was Mark Burgess while working as a post-doctoral fellow at the University of Oslo in Norway (Burgess & Zamboni, 2013). From his first research and the creation of CFEngine, others have followed on a pursuit to solve the problem of maintaining consistency and configuration state. Nowadays, system administrators have tools like Ansible, Salt, Puppet, and Chef (Chef, n.d).

One of the barriers of entry to testing malware with a sandbox is the creation of suitable targets. The accepted method of achieving this goal is to manually create a virtual machine, configure it appropriately 'until it works' and take a snapshot. The machine can then be reverted to a good known state after completion of the test. Medium to large organizations have many combinations of software. A complete testing of the organizational exposure of malware means many hours spent in creating virtual machine targets.

An idea that will ease the manual process of target creation is using a configuration management framework to define and set the critical attributes required. Configuration management systems

use a Domain Specific Language (DSL) to define a system end state. Once the list of required attributes is complete and executed, the system will reach the desired state (converge). A file server can then serve as the main repository for malware testing targets.

The method explained in this paper uses Chef-Solo (Rahman, 2014), but any system capable of providing configuration management as source code is a suitable alternative. Chef (the commercial company that created the tool) provides native packages for Windows and Linux to install Chef. Another possible method is to install the required software as a Ruby gem. One of the advantages of using the Ruby gem method is the ability to add gems that support our workflows.

Chef supports the configuration of Linux and Windows hosts. This system is capable of configuring any attribute on the target including software packages, files and directories, users and permissions, and running services. An incident response team can leverage this to automate the creation of virtual instances suitable for malware testing. The deployment of virtual machine instances using a scripted configuration saves time and minimizes human error.

In a large enterprise, a centralized server hosts this configuration management system. For small deployments, a reduced scale version of the software is available. Both deployment strategies are capable of producing a fully configured server. The difference between both is subtle and not relevant to the business case of automated malware target creation.

3.2. Virtualization

The technology now called virtualization started in the 60's as a method of partitioning computing resources inside a mainframe to increase efficiency and optimize resource use. Thanks to the decrease in the price of commodity hardware, this technology is now available to people for little to no cost.

Both static and dynamic methodologies complement the analysis of malicious code. Static code analysis requires the determination of many internal aspects of an operating system. Some aspects include Dynamic Link Libraries (DLL's), operating system calls, strings that point to malicious internet hostnames or C&C servers, or whether the code is packed and obfuscated to avoid anti-virus detection. Dynamic analysis requires the execution of the malicious program. The execution activity needs observation and recording. Examples of dynamic analysis artifacts include registry keys created, open communication ports, and attempts to reach a C&C server.

Malicious code installed on the target system begins the phase of dynamic analysis. Restoring the target system after analysis prepares for the next test.

Using virtualization greatly improves the ability to restore a system to a pristine condition by using a snapshot. A snapshot is a saved system state that includes both non-volatile and volatile data at a given time. All running programs and processes are "frozen" and restored at will. Most virtualization suites support the idea of multiple snapshots for a single system.

Another characteristic of virtualization is the different networking modes. A virtual machine can be "bridged" to a physical interface. This means the virtual host has physical (Layer 2) access to any other host on the network. Network Address Translation (NAT) is the second mode of networking. The virtual machine is sharing the host's IP address in NAT mode. All connections on the physical network seem to come from the physical host. The virtual machine has no Layer 2 access on the network. Another scheme for virtual network connectivity is the "host-only." Using this method the virtual machine has access only to other virtual machines on the same virtual network, and optionally the physical host. Traffic generated by virtual machines on this type of network is isolated from any physical host other than the one containing the hypervisor.

Cuckoo Sandbox takes advantage of the characteristics given by "host-only" virtual networking and the ability to take snapshots. The virtual machine can be reverted to a good known state once the malware test is complete. Incident responders can have a good level of assurance that traffic is not leaving the physical host and infecting other parts of the network. The sandbox intercepts any C&C request or other attempts to gain Internet access by handling them appropriately. The creators of Cuckoo have enabled the ability to choose from several virtualization technologies. The currently supported hypervisors are VMware, VirtualBox, and KVM.

Jose Ortiz, siltecon@gmail.com

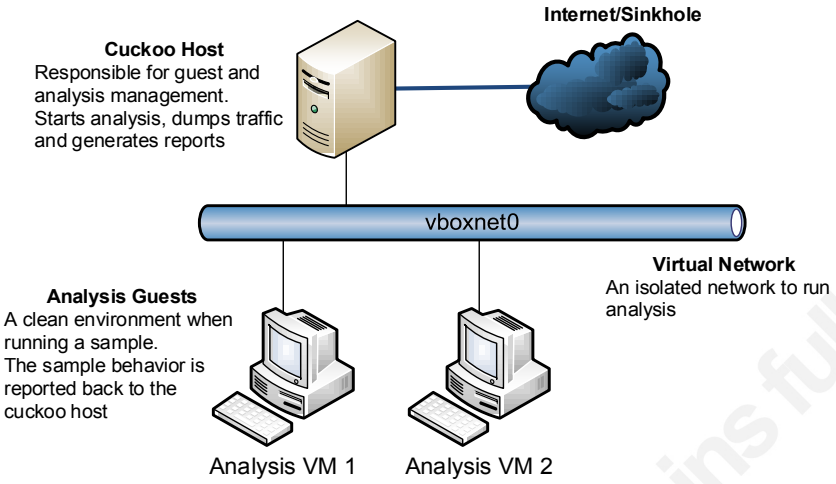

Figure 1. Cuckoo Sandbox virtualization.

(Oktavianto & Muhardianto, 2013, adapted).

3.3. Provisioning

3.3.1. Packer

This is a set of binaries to help in the creation of identical virtual machine images (Packer, n.d.). The resulting artifacts are then exported in a suitable format to support any virtualization workload desired (can include cloud-ready images). Provisioning an instance is a different problem solved by other tools and workflows. Packer supports the creation of many artifacts, like VMware, Docker, OpenStack, Parallels, QEMU, Amazon AMI, and VirtualBox. The platform supports extensions via plugins to allow the creation of custom builders. Windows images can also be 'sysprepped' during this process.

The virtual machine definition is a JSON (Sriparasa, 2013) structure that indicates various Packer component configurations and attributes (see Appendix E). This structure is the template used to configure various aspects of Packer. The structure keys are 'builders,' 'description,' 'min_packer_version,' 'post-processors,' 'variables,' and 'provisioners.' The builder key is the only mandatory component of the template. It is possible to use multiple builders on a single configuration file. The implication of this is that the testing team can create an identical machine for multiple platforms without any extra work.

In order to use this builder, VirtualBox needs to be installed on the development machine. It is possible to export target artifacts in either OVF or OVA format. The VirtualBox builder is capable of using an operating system installation ISO as a starting point to create a virtual machine from scratch. If an existing VirtualBox machine is available, the builder is capable of

creating a second machine using the first box as a starting point. One of the advantages of this builder is its ability to feed the resulting OVA back into the workflow to iterate on a given machine until the operator achieves the desired configuration state.

A provisioner is a component of Packer used to add customization software packages to the base operating system. This section is optional; however, it is uncommon to find a template that does not use this feature. The order found in the template determines when provisioners execute. As of the date of this paper, Packer supports provisioners for Ansible, Chef (solo, client), Puppet (masterless, server), Salt, shell scripts, file uploads, and custom. The custom provisioner allows for the creation of new provisioners without altering the source code of Packer.

Packer also has the ability to post process an image after its creation. One example is the 'Vagrant' post-processor that takes the resulting artifact and creates a Vagrant box. A single configuration file can specify multiple post processors. Thus, it is possible to create a Vagrant box and an artifact from the VMware builder and upload it to VSphere in a single Packer run.

3.3.2. Vagrant

Vagrant (Hashimoto, 2013) allows the use of code to create virtual environments that closely mimic production systems (Peacock, 2013). Since everything is code, development becomes testable and repeatable, and consistent between contributors. Multiple developers can collaborate using SDLC (Software Development Life Cycle) best practices. It is also possible to track changes and recreate the complete history of the project. Anyone with access to the repository can replicate the environment by downloading the code and running Vagrant.

This tool started as a method to automate VirtualBox deployments on a single developer workstation. The developers have extended it to support VMware, Docker, Hyper-V, Amazon Web Services, and custom providers. The default provider is for VirtualBox deployments. The VMware provider is a commercial product that requires a separate license for its use.

A single Vagrant file is capable of deploying multiple virtual machines, configuring networks, and enabling synced folders between the guest and host systems. Another ability of Vagrant is being able to apply configurations using Chef, Puppet, CFEngine, Ansible, or shell scripts. Users can extend, reconfigure, repackage, and deploy a box created with Packer. Another developer can use the resulting box exported as an OVA/OVF and create his/her own extensions.

After a Vagrant run, boxes become the basic virtualization artifact provisioned and configured. The user can halt, reload, provision, and destroy virtual machines at any time. A common test is

to completely destroy the environment and recreate it to ensure consistency and repeatability of the results. A Vagrant box needs importing into the development environment before it is available for use. The usual method of accomplishing this is to specify the URL where the box is located.

There is an incompatibility with the networking requirements when trying to use Vagrant directly to provision virtual machines for Cuckoo. Vagrant requires that the first network device attached to a virtual machine be a NAT device. This device allows for port forwarding into the host. This is required in order to allow Vagrant SSH access into a virtual machine for configuration. Cuckoo requires that machines use 'host-only' networking to prevent potentially malicious traffic from reaching the real network. The remedy to this is to export the Vagrant artifact using the OVA file method and import it into VirtualBox.

4. Sandbox Deployment

Cuckoo Sandbox deployment on a Linux-based environment is preferred. Other operating systems could run Cuckoo successfully, but without any warranty on performance or stability.

4.1. Scripting the installation

Cuckoo Sandbox provides incredible functionality for a piece of software that is free. It is very simple to use and the reports obtained, from it contain a large amount of detailed information. This functionality comes with the price of a complex and delicate installation. A successful deployment requires a combination of native distribution packages, compiled libraries, and other frameworks like Volatility.

An install script provided in the first Appendix helps reduce the time it takes to install Cuckoo. The BASH script installs Cuckoo Sandbox and all its dependencies including VirtualBox. This script created the development environment used for this research paper. Using this in production may require additional steps not documented here including securing the sandbox or complying with corporate policies. The reference system is a minimal installation of Debian Linux 7.6 64-bit. A different Linux distribution may need other configurations or additional software packages. To complete the installation requires the configuration of Cuckoo. The recommended source is always the official documentation found on the Cuckoo website. Refer to Appendix I for a checklist to aid with the deployment.

Jose Ortiz, siltecon@gmail.com

4.2. Creating testing targets

The next step after the installation and configuration of the Cuckoo host is the creation of guest virtual machines that will serve as testing targets. A base box is created using Packer. Vagrant will configure this intermediate VM. An OVA file containing the complete and configured VM is the product of Vagrant. The importing of this OVA file into VirtualBox completes the sandbox target.

The prerequisites for creating a base box are: a) the Packer binaries; b) Windows operating system media in the form of an ISO file; c) a virtualization builder, and d) a JSON template. Debian Linux 7.6 (64-bit) is a reference test system for the script provided in Appendix A. It installs all the required packages for a development workstation, VirtualBox, and Cuckoo.

Instead of creating templates and provisioning scripts from scratch, users can leverage the community-driven QWindows templates created by Moujan Anna and Mischa Taylor and hosted on GitHub. The only missing piece is the installation media to create our target. It is required to download the templates' repository and all scripts using Git from https://github.com/boxcutter/windows.git. The installation script currently obtains this repository and copies it to "/home/cuckoo/boxcutter."

The Cuckoo documentation uses Windows XP to create the sandbox environment, but support for this operating system has ended. Windows 7 is a suitable alternative for creating the sandbox, provided the disabling of UAC (User Account Control). Some organizations are still using XP, but the trend is to migrate to something more modern and supported. For this reason, Windows 7 is the current Operating System used. A file share will host the resulting boxes from Packer. This library serves as the main repository for reference boxes, consumed as necessary by the malware research team.

Jose Ortiz, siltecon@gmail.com

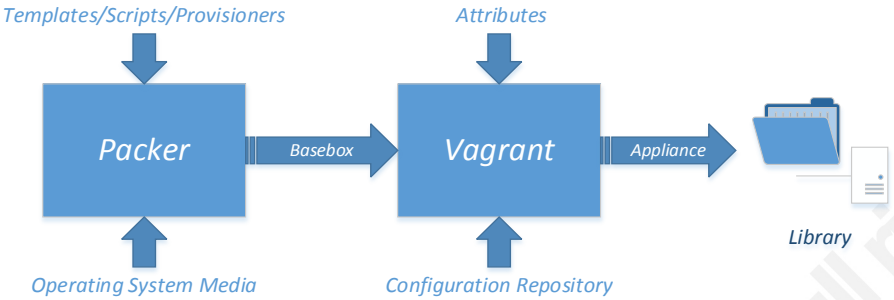

Figure 2 Integration Workflow. Author retains all copyrights.

The definition file used to create the artifact is 'win7x64-pro.json,' located in the base directory of our templates repository. This file is a JSON structure and includes builders for VMware, Parallels, and VirtualBox. When making any changes it is extremely important to make sure that the section that corresponds to the builder intended for use is the one edited (refer to Appendix E).

Cuckoo requires us to disable UAC if Windows 7 is the sandbox. The community templates already include a script that performs this step for us. The script default behavior is disabled and not included in the run. The addition of such functionality requires an addition to the list of scripts executed during artifact creation. Inside the JSON template, find the array called "floppy_files" and add a new string element called "floppy/uac-disable.bat" (refer to Appendix E). The configuration file "floppy/win7x64-pro/Autounattend.xml" contains any customizations to the installation that are specific to an organization such as registration keys.

Before building the box, the process needs a Makefile that overrides some default configuration values. Create a file called "Makefile.local" inside the boxcutter repository ("/home/cuckoo/boxcutter," if the install script is used) with the following contents:

```
CM := chef
WIN7_X64_PRO := <absolute_path_to_the_installation_iso>
```

These variables will signal Packer to install Chef and use the provided ISO. This demo used Windows 7 Pro 64 bit with Service Pack 1 Volume License (en_windows_7_professional_with_sp1_vl_build_x64_dvd_u_677791.iso). To create the box, run the 'make' command from the /home/cuckoo/boxcutter directory, passing a parameter that indicates the virtualization platform and JSON template combination to use. The following is an example of the make command:

```
# make virtualbox/win7x64-pro
```

The resulting box is located inside the *box/virtualbox* directory. The local virtual machine catalog contains this artifact after importing. Vagrant already includes that functionality. To import the box, enter this command at the terminal prompt from the *boxcutter* base directory:

```
# vagrant box add cuckoo/win7x64-pro box/virtualbox/<name>
```

Substitute the tag <name> with the box name (e.g win7x64-pro-cheflatest-1.0.4.box)

4.3. Target Customization

4.3.1. Repository Configuration

The virtual machine imported by Vagrant needs further customization before exporting it into VirtualBox. Customization may include the installation of third party applications, user accounts, and directory permissions. The purpose of this step is to mirror the production systems as closely as possible. By using Chef, the user is able to set almost any configuration item inside a target. The decision of which configuration management system to use is mostly one of corporate policy. Vagrant supports Ansible, CFEngine, Chef, Puppet, Salt, and shell scripts.

In this demonstration, the scope will be limited to the installation of packages, specifically MSI and EXE files. The preferred method is to use packages that have a silent install option that need no user interaction. If the silent option is not available and the user cannot modify the MSI, the only other option is to have the installation proceed manually. A single Vagrantfile can create multiple targets. Each target can have its own separate list of packages and configurations. Appendix D demonstrates a Vagrantfile with a single profile. This sample Vagrantfile is a starting point. Use it to extend and create a personal configuration.

To create the base repository the user will need to initialize it, download a single cookbook (along with its required dependency), and create a custom cookbook to hold our configuration. The installation script in Appendix A already installs Chef as a Ruby gem and creates the Chef repository. This repository is located inside the home directory of the Cuckoo user ("/home/cuckoo/chef-repo"). The next step is to download the 'windows' cookbook and create the 'cuckoosandbox' cookbook. Execute the commands below in the following order. The repository is now ready to accept changes.

```
# cd /home/cuckoo/chef-repo
# knife cookbook site install windows
# knife cookbook create cuckoosandbox
```

The key to installing packages in a generic way and with a minimal amount of code is to name them consistently. The scheme to installing packages uses a string with the form of '<name>_<version>.<extension>.' The 'name' label describes the package file name inside the repository, like 'adobereader,' 'java,' or 'flashplayer.' The 'version' tag is a numeric indicator specific to a release, like '9.2.0,' or '9.5.' The 'extension' is either 'msi' or 'exe.' The 'cuckoo' cookbook stores all MSI and EXE files inside the "cuckoo/files/default" directory. This directory holds all packages. Remember to name packages appropriately in a manner consistent with the description hash (e.g. adobereader_9.2.0.msi, java_1.5.3.exe).

Figure 3 describes the data passed to Chef as a JSON structure. The identifier 'cuckoosandbox' corresponds to a cookbook name. The array 'packages' contains all the hashes (dictionary) that describes a file name. Curly braces enclose each dictionary entry, separated by a comma. The default recipe on the 'cuckoosandbox' cookbook reads this data structure and parses it appropriately.

The Cuckoo cookbook has no default attributes. These attributes will come from a JSON structure defined inside the Vagrantfile and passed to the node at runtime. A single recipe (refer to Appendix C) in this cookbook copies the Cuckoo agent and creates a registry key that will make such agents start automatically when the system boots. The second part of this recipe copies the packages from the default cookbook files and installs them.

Jose Ortiz, siltecon@gmail.com

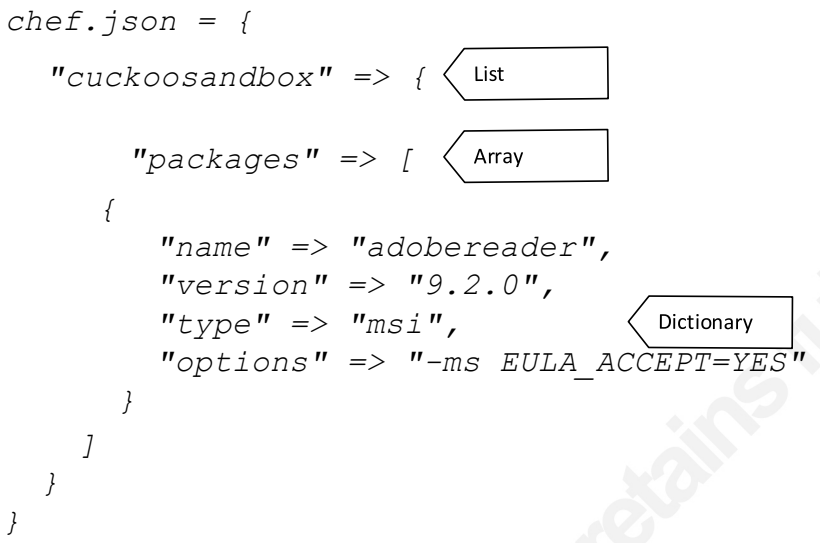

Figure 3. JSON attributes describing a file to install. Author retains all copyrights.

The Vagrantfile is what contains a JSON structure that describes the packages needed to install. The key to Appendix C is an array of hashes with four key and value pairs. Each one of these pairs defines a critical property (name, version, file type, and any install options). It is possible to define multiple profiles in a single Vagrantfile. For example, consider the following:

```
config.vm.define :profile1 do |profile1|
     <options for profile 1>
end

config.vm.define :profile2 do |profile2|
     <options for profile 2>
end

config.vm.define :profile3 do |profile3|
     <options for profile 3>
end
```

Once the Vagrantfile is complete, all targets deploy by invoking the Vagrant binary, as shown below:

```
# vagrant up
```

Vagrant will copy the template specified and customize it one at a time until complete. Do not interrupt this process. Once finished, the VirtualBox manager application will appear similar to

Jose Ortiz, siltecon@gmail.com

Figure 4. The documentation for Cuckoo recommends a soft reboot of the machine to make sure that the Cuckoo agent is running properly. The command used to do this is 'vagrant reload.' Once all the machines reboot, proceed to take a snapshot. The key to success is to make sure that both the virtual machine and snapshot match the directives in "cuckoo.conf." Take the snapshot while the machine is running and the Cuckoo agent is listening on port 8000.

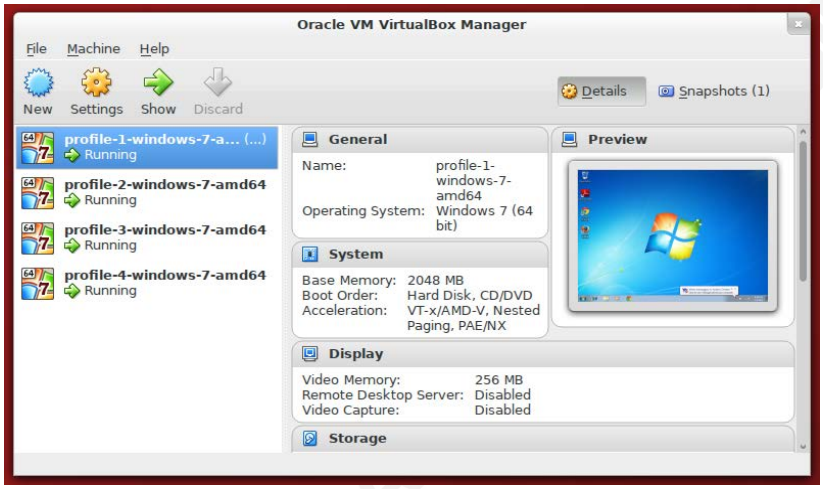

Figure 4. VirtualBox targets created by Vagrant. Author retains all copyrights.

Vagrant forces a virtual machine to have the first interface as 'NAT.' One can add another interface as 'host-only,' but the first interface will always be 'NAT.' This is not compatible with the way Cuckoo Sandbox works. In the sandbox environment, all interfaces need to be 'host-only' to black hole all traffic and keep potentially malicious artifacts from reaching the network. To get around this incompatibility, manually set the network interface options to mirror Figure 5. It is very important to allow promiscuous mode in order to allow the Cuckoo agent to collect Tcpdump (Pcap) data.

Jose Ortiz, siltecon@gmail.com

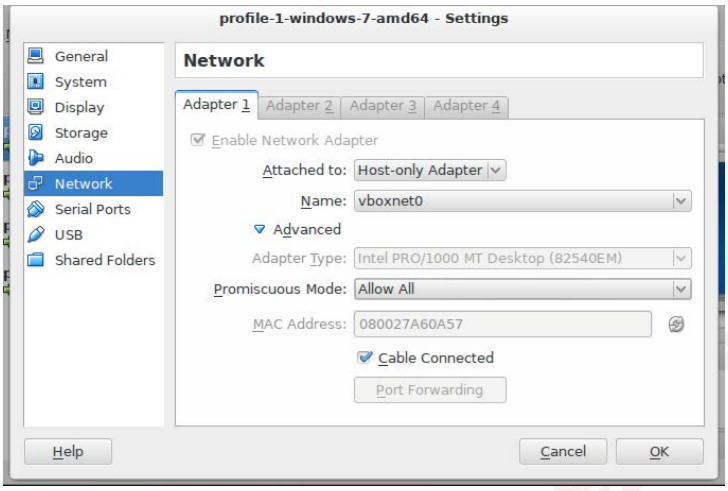

Figure 5. VirtualBox network settings. Author retains all copyrights.

5. Sandbox Use

5.1. Submitting samples

Cuckoo has a flexible mechanism to submit samples that allow the end user to interact with the suite by using a web browser, run a submission utility from the command line, or integrate the sandbox with other tools like Bro IDS by using scripting and the REST API. The only prerequisite to submit a sample is that Cuckoo needs to be running. To start Cuckoo, change to the installation's root directory ("/opt/cuckoo" in the provided script) and start the sandbox:

```
$ cd /opt/cuckoo
$ python cuckoo.py
```

By default, Cuckoo is set to check for updates on api.cuckoosandbox.org. If the sandbox is running inside an isolated enclave, accessing the update service is not possible. Setting the parameter *version_check* to *off* inside "cuckoo.conf" disables automatic checking for updates. A properly prepared Windows target has no firewall activated, no automatic updates and no anti-virus software installed. Using a scripted installation of Windows ensures that the template will meet those requirements programmatically.

Cuckoo will do a sanity check during startup and attempt to connect to the instances specified in "cuckoo.conf" and "virtualization.conf." One of the most common mistakes made is the misspelling of a machine name or providing incorrect IP addresses. Make sure to double check the IP address of the Cuckoo server and all hostnames. The "submit.py" utility has many options

that allow for customization of the analysis. Consult the official documentation for a complete list.

It is possible to submit a directory path instead of a file. This will create an analysis job for every one of the files contained inside the directory. This can be useful if the user is trying to automate Cuckoo by using a centralized location.

5.2. Using Cuckoo with Maltego

Maltego is an application that allows for the analysis of relationships between entities with the intent of obtaining actionable intelligence (Maltego, n.d.). It can help an intelligence analyst determine relations and links between people, groups, companies, websites, networks, affiliations, documents, and files. Maltego is also capable of extracting information found on social media sites, like Facebook, and using this information to create relations between entities. Open Source Intelligence (OSINT) is what links these entities via the concept of transforms.

The open source community has developed Canari as a Python framework that allows the development of Maltego transforms. The developers created this framework to aid in the execution of penetration testing and vulnerability assessments. Its use has evolved to include digital forensics analysis. The main Maltego terminology is as follows (adapted from the Canari Framework Documentation, http://www.canariproject.com/canari-a-quick-introduction):

- Entity: Represented as a node in Maltego.
- Transform: A function that takes one entity as input and produces zero or more entities as output.
- Input Entity: Entity passed into the transform to use for data mining purposes.
- Output Entity: Entity returned by the transform.
- Transform Module: A python module to local transform code.
- Transform Package: A python package containing one or more modules.

Cuckooforcanari is a plugin that gives Maltego the ability to submit either a file or URL to Cuckoo directly without the need for opening another application. Using Maltego to analyze malware gives us the ability to create a graphical representation of any relations found existing between the malware and other entities like hosts.

Jose Ortiz, siltecon@gmail.com

The installation of Cuckooforcanari requires the Python Ez_setup tool and the Canari framework. The Ez_setup script is part of the Easy_install Python modules. These tools enable the automatic downloading, building, installation, and management of Python packages. After installing all dependencies, clone the Cuckooforcanari repository and build the plugin as follows:

```
# wget https://bootstrap.pypa.io/ez_setup.py -O - | python
# easy_install canari
# mkdir cuckooforcanari
# cd cuckooforcanari/
# git clone https://github.com/bostonlink/cuckooforcanari.git
# cd cuckooforcanari/
# python setup.py install
# canari install-package cuckooforcanari
```

The above script will create a Maltego archive called "cuckooforcanari.mtz." This package contains all the transforms that support Cuckoo inside of Maltego. The end user will import this package by following these instructions:

1. Open Maltego.
2. Click on the home button (Maltego icon, top-left corner).
3. Click on 'Import'.
4. Click on 'Import Configuration'.
5. Follow the prompts.

The next step is to ensure that "cuckooforcanari.conf" contains the proper configuration by entering the hostname or IP address of the sandbox and the API port (default 8090). Make sure the sandbox is running the API server. This makes it possible to start the service as the Cuckoo user. Run the following commands to start the API service:

```
$ cd /opt/cuckoo
$ ./utils/api.py --host 0.0.0.0 --port 8090
```

Figure 6 shows the new palette that contains all the Cuckoo sandbox transforms provided by this extension. This new palette provides the analyst with the tools required to submit a malware sample to Cuckoo and visualize the analysis. To start using the new palette, drag-and-drop the Cuckoo Malware Sample tool into the Main View (some call this the Canvas), and right click. This action will create a pop-up menu with options. Choose Run Transform -> Cuckoo Sandbox

-> Submit file for analysis. The backend application will make a REST API call in the form of a HTTP POST and submit the file for analysis.

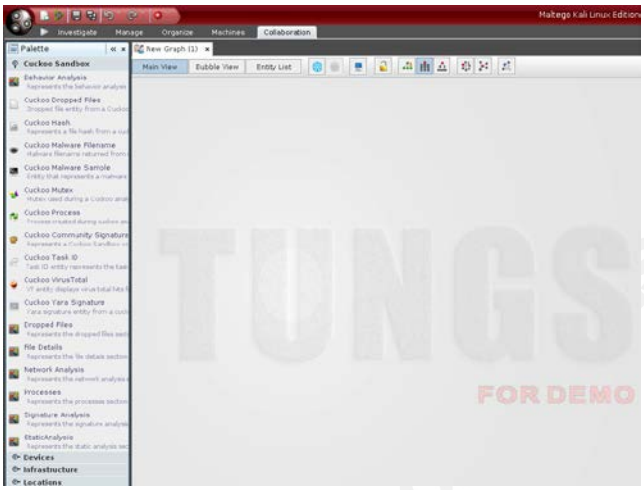

Figure 6. Maltego palette after importing cuckooforcanari. Author retains all copyrights.

5.3. Automating with Security Onion and Bro

The idea of building and configuring an IDS from scratch is never an easy proposition. A properly configured system using open source requires extensive knowledge of not only Snort, but also Linux, scripting, kernel tuning, web and database servers, and IPTables. Many readily configured Linux Live CDs have attempted to fill the gap that exists for an easy to deploy IDS. The most popular Linux distribution that simplifies this process is Security Onion (Bejtlich, 2013). Dough Burks and a group of volunteers support this compilation of software. It can be downloaded free of cost from SourceForge (http://sourceforge.net/projects/security-onion). The included toolset combines either Snort or Suricata to offer signature based intrusion detection, flow analysis tools, and a number of graphical user interfaces to get access to the data.

One of the tools included in this distribution is the Bro Network Security Monitor (Bro, n.d.). Bro is compared to a signature based IDS, but it provides a framework that facilitates the analysis of network protocols. Figure 7 shows a graphical representation of the Bro architecture. The initial step takes packets from the network or a Pcap file and produces a series of events. These events are a description of what activity has occurred, without considering policy or the reason the events occurred. The policy script interpreter receives these events. This component takes scripts written in the Bro language and is capable of creating alerts, or executing external programs and scripts.

Jose Ortiz, siltecon@gmail.com

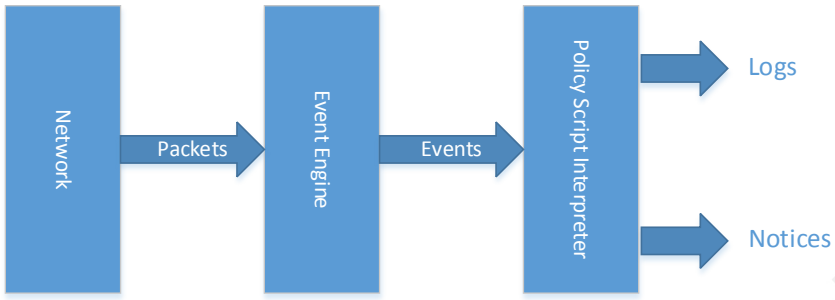

Figure 7. Bro Network Monitor Architecture. Author retains all copyrights.

An interesting ability of Bro is its ability to extract files directly from the wire based on the observed MIME type. Writing file analysis scripts was a cumbersome process before Bro version 2.3 because files can be present on the network using various application protocols. The new file analysis framework generalizes the presentation of data regardless of the application layer involved.

Security Onion already includes the required Bro scripts for extracting binary files. During the initial setup of Security Onion, enable Bro and file extraction when asked. After following these instructions, all executable files seen by Bro are downloaded to the "/nsm/bro/extracted" directory. Extracted files will use filenames of the form PROTOCOL-uid.exe (e.g. 'HTTP-FOpuMT3z0Mv0NEs2d.exe'). The first part of the filename is the protocol, followed by the Bro connection ID of this data stream. The script below will copy all extracted EXE's to the Cuckoo sandbox and rename the binary with a string that represents the MD5 hash, as shown below:

```
#!/bin/bash

#this script will run on the security onion box

#where are the extracted files?
bindir="/nsm/bro/extracted"

#destination directory on the cuckoo box
destdir="/opt/cuckoo/bindir"

#where is the sandbox?
sandbox="192.168.10.10"

#what credentials to scp files to the sandbox?
user="cuckoo"
```

Jose Ortiz, siltecon@gmail.com

```
rsakey="/home/cuckoo/.ssh/id_rsa"

cd $bindir

#copy all exe files to the sandbox
for file in $(ls *.exe)
do
  scp -i ${rsakey} ${file} \
    ${user}@${sandbox}:${destdir}/$(md5sum ${file})
done
```

The above script will not work if the stream has encrypted communication, like HTTPS or SSH. Bro is passively listening and has no way of decrypting this type of conversation, but some data may still be available. In the case of SSL/TLS, Bro will be able to extract any X509 certificates present. Once collected, the "submit.py" utility will submit the samples. The magic script below will only submit samples not yet seen:

```
#!/bin/bash

bookmark="/var/lib/bro/bookmark"
bindir="/var/lib/binaries"

cd $bindir

for i in $(ls)
do
if [ grep -Fx "$i" "$bookmark" ];then
  rm -f ${i}
else
  echo ${i} >> $bookmark
  curl -F file=@${i} http://localhost:8090/tasks/create/file
  rm -f ${i}
fi
done
```

The aforementioned script will submit samples received from Security Onion and send them to the sandbox for analysis. The bookmark keeps track of binaries already sent to the sandbox. The script deletes the files after submission to save hard drive space. This technique can recreate some of the capabilities of commercial sandboxes capable of automatic extraction and analysis of binaries.

Jose Ortiz, siltecon@gmail.com

6. Conclusion

The current threat environment includes an incredible number of new malware variations released daily. Malicious code is the primary enabler for an attacker to gain initial access and maintain foothold on a network. The probability of finding this type of program in the course of an investigation is very high. With so many variations of malware, an incident response team is in need of tools and techniques to accelerate the analysis process. Using automation can potentially increase the effectiveness of team during the course of an investigation in attempting to identify, catalogue, and analyze malware. Multiple commercial vendors have developed a market for appliances and tools that provide automated malware analysis, and in some cases active threat response.

The Open Source Community has responded with alternatives to commercial offerings that promise to deliver a good understanding of the methods and techniques used by attackers, and at the same time speed up the analysis process. A small incident response team can combine hardware and network virtualization, free memory forensic tools, and inexpensive computers to implement an automated malware analysis capability. These alternatives, perceived to have a low cost of entry, require an investment in time and technical expertise. The proper implementation of a sandbox using Open Source requires many delicate steps and is highly prone to error. A good strategy to success is to pin software library dependencies to specific versions known to work and script the installation.

Cuckoo Sandbox is arguably one of the most popular alternatives to analyze malware and keep a detailed record of findings. Many automation scripts exist that help in the implementation of the sandbox, but not many solutions for the consistent development of testing targets. This research paper has explored just one possibility to address this problem, along with some additional integration with other systems.

Jose Ortiz, siltecon@gmail.com

7. References

AV-TEST – The Independent IT-Security Institute. (n.d.). Retrieved August 2, 2015, from https://www.av-test.org/en/

Bejtlich, R. (2013). The practice of network security monitoring: Understanding incident detection and response (1st ed.). Lay Flat.

Brickell, E. F., Hall, C. D., Cihula, J. F., & Uhlig, R. (2011). U.S. Patent No. 7,908,653. Washington, DC: U.S. Patent and Trademark Office.

Bro - The Bro Network Security Monitor. (n.d.). Retrieved August 5, 2015, from https://www.bro.org/

Chef - All about Chef ... (n.d.). Retrieved January 15, 2015, from https://docs.chef.io/

Dalziel, M. (2014). How to defeat advanced malware: New tools for protection and forensics. Amsterdam: Elsevier.

Dash, P. (2013). Getting started with Oracle VM VirtualBox: Build your own virtual environment from scratch using VirtualBox. Birmingham, UK: Packt Pub.

FakeNet. (2012, February 29) Retrieved August 4, 2015, from http://practicalmalwareanalysis.com/fakenet/

Greamo, C., & Ghosh, A. (2011). Sandboxing and virtualization: Modern tools for combating malware. IEEE Security and Privacy, 9, 79–82. doi:10.1109/MSP.2011.36

Hashimoto, M. (2013). Vagrant: Up and running. O'Reilly Media.

Hosmer, C. (2014). Python forensics: A workbench for inventing and sharing digital forensic technology. Waltham, MA: Syngress.

Hutchins, E. M., Cloppert, M. J., & Amin, R. M. (2011). Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains. 6th Annual International Conference on Information Warfare and Security, 1–14. Retrieved from http://papers.rohanamin.com/wp-content/uploads/papers.rohanamin.com/2011/08/iciw2011.pdf

Maltego. (n.d.). Retrieved August 5, 2015, from https://www.paterva.com/web6/products/maltego.php

MHR - Malware Hash Registry. (n.d.). Retrieved August 4, 2015, from http://www.team-cymru.org/MHR.html

Jose Ortiz, siltecon@gmail.com

Oktavianto, D., & Muhardianto, I. (2013). Cuckoo malware analysis: Analyze malware using Cuckoo Sandbox. Packt Publishing.

Packer by HashiCorp. (n.d.). Retrieved August 5, 2015, from https://packer.io/

Peacock, M. (2013). Creating Development Environments with Vagrant. Packt Publishing.

Rahman, N. U. (2014). Configuration management with Chef-Solo: A comprehensive guide to get you up and running with Chef-Solo. Birmingham, UK: Packt Pub.

Sabharwal, N., & Wadhwa, M. (2014). Automation through Chef Opscode: A hands-on approach to Chef. New York: Apress.

Sikorski, M., & Honig, A. (2012). Practical malware analysis: The hands-on guide to dissecting malicious software. San Francisco: No Starch Press.

Sriparasa, S. (2013). JavaScript and JSON Essentials. Birmingham: Packt Publishing.

VirusTotal - Free Online Virus, Malware and URL Scanner. (n.d.). Retrieved August 4, 2015, from https://www.virustotal.com/

Zamboni, D., & Burgess, M. (2012). Learning CFEngine 3. Sebastopol, CA: O'Reilly Media.

Zeltser, L. (2015). Free Automated Malware Analysis Sandboxes and Services. Retrieved July 4, 2015, from https://zeltser.com/automated-malware-analysis/

Jose Ortiz, siltecon@gmail.com

Appendix A - Cuckoo install script


```
#!/bin/bash

CUCKOO_USER=cuckoo
CUCKOO_GROUP=cuckoo
CUCKOO_HOME=/home/cuckoo
CUCKOO_DIR=/opt/cuckoo
VBOXUSER=vboxusers

### DO NOT EDIT BELOW THIS LINE ###
if [ "$(id -u)" != "0" ]; then
  echo "The install script should be run as root" 1>&2
  exit 1
fi

wget -q https://www.virtualbox.org/download/oracle_vbox.asc -O- | apt-key add
-

VIRTUALBOX_REPO_FILE=/etc/apt/sources.list.d/virtualbox.list

cat > $VIRTUALBOX_REPO_FILE << EOF
deb http://download.virtualbox.org/virtualbox/debian wheezy contrib
EOF

apt-get update && apt-get upgrade -y

apt-get install -y python python-dev python-pip python-gridfs git build-
essential \
mongodb libapache2-mod-wsgi python-sqlalchemy python-bson python-jinja2
python-magic \
python-bottle python-pefile python-dpkt libpcre3 libpcre3-dev unzip python-
crypto \
python-imaging python-openpyxl python-tz ipython cmake libdistorm64-1
libxml2-dev \
libxslt-dev python-chardet tcpdump libcap2-bin dkms virtualbox-4.3 pwgen \
libapache2-mod-gnutls curl

vboxmanage extpack install Oracle_VM_VirtualBox_Extension_Pack-4.3.28-
100309.vbox-extpack

groupadd $CUCKOO_GROUP
useradd -G $VBOXUSER -g $CUCKOO_GROUP $CUCKOO_USER || true

pip install Django==1.6.1 flask flask-restful flask-sqlalchemy requests
alembic python-dateutil pymongo

tar -zxvf dpkt-1.8.tar.gz && cd dpkt-1.8 && python setup.py install
```

Jose Ortiz, siltecon@gmail.com

```
cd .. && rm -fr dpkt-1.8

tar zxf v3.3.0.tar.gz && cd yara-3.3.0
./bootstrap.sh && ./configure
make && make install

cd yara-python
python setup.py build && python setup.py install
cd ../../ && rm -fr yara-3.3.0

tar zxf ssdeep-2.10.tar.gz && cd ssdeep-2.10
./configure && make && make check && make install
cd .. && rm -fr ssdeep-2.10

tar zxvf pydeep-0.2.tar.gz && cd pydeep-0.2
python setup.py build && python setup.py install
cd .. && rm -fr pydeep-0.2

unzip distorm3.zip && cd distorm3 && python setup.py install
cd .. && rm -fr distorm3

tar xvfz libforensic1394-0.2.tar.gz && cd libforensic1394-0.2
mkdir build && cd build && cmake -G"Unix Makefiles" ../ && make && make
install
cd ../../ && rm -fr libforensic1394-0.2

tar zxvf volatility-2.4.tar.gz && cd volatility-2.4 && python setup.py
install
cd .. && rm -fr volatility-2.4

tar xvfz cybox-2.0.1.4.tar.gz && cd cybox-2.0.1.4 && python setup.py install
cd .. && rm -fr cybox-2.0.1.4

tar xvfz maec-4.0.1.0.tar.gz && cd maec-4.0.1.0 && python setup.py install
cd .. && rm -fr maec-4.0.1.0

#these instructions come from the cuckoo installation manual
setcap cap_net_raw,cap_net_admin=eip /usr/sbin/tcpdump

#installing cuckoo 1.2
tar zxvf cuckoo_1.2.tar.gz -C /opt/
mkdir /opt/cuckoo/tmp

#modified the default cuckoo local_setting.py
#reference: https://github.com/cuckoobox/cuckoo
cat << 's/.* CUCKOO_PATH .*/CUCKOO_PATH = "/opt/cuckoo"/'
/opt/cuckoo/web/web/local_settings.py

#modified the default cuckoo wsgi.py
#reference: https://github.com/cuckoobox/cuckoo
sed -i 's/import os/&,sys\n sys.path.append('/opt/cuckoo')\n
sys.path.append('/opt/cuckoo/web')\n os.chdir('/opt/cuckoo/web/')
/'  /opt/cuckoo/web/web/wsgi.py
```

Jose Ortiz, siltecon@gmail.com

```
unzip -d /usr/local/bin packer_0.8.1_linux_amd64.zip

[ -d /home/cuckoo/boxcutter ] || git clone \
https://github.com/boxcutter/windows.git /home/cuckoo/boxcutter

#reference: https://rvm.io/rvm/install
gpg --keyserver hkp://keys.gnupg.net --recv-keys
409B6B1796C275462A1703113804BB82D39DC0E3
\curl -sSL https://get.rvm.io | bash -s stable --ruby

source /usr/local/rvm/scripts/rvm
gem install chef --no-document

[ -d /home/cuckoo/chef-repo ] || git clone git://github.com/chef/chef-
repo.git /home/cuckoo/chef-repo

#creating default knife.rb config
cat << '__EOF__' > /home/cuckoo/chef-repo/.chef/knife.rb
log_level                :info
log_location             STDOUT
cookbook_path        ["#{current_dir}/cookbooks"]
__EOF__

#reinitialize git repo
rm -fr /home/cuckoo/chef-repo/.git && git init /home/cuckoo/chef-repo
cd /home/cuckoo/chef-repo && git add . && git commit -m 'initial commit'

#install vagrant
dpkg -i vagrant_1.7.2_x86_64.deb

#begin boxcutter repo setup. the last commit that worked was d0a09ec
cd /home/cuckoo/boxcutter
git checkout -b cuckoo d0a09ec
sed -i 's/"40960",/"20480",/' win7x64-pro.json
sed -i 's/"floppy\/passwordchange.bat",/&\n
"floppy\/disablewinupdate.bat",/' win7x64-pro.json
sed -i 's/"floppy\/passwordchange.bat",/&\n        "floppy\/uac-
disable.bat",/' win7x64-pro.json
sed -i 's/"floppy\/disablewinpudate.bat",/&\n        "floppy\/prepare-
cuckoo.bat",/' win7x64-pro.json

#this script was inspired by openssh.bat and modified to suit my needs
cat << '__EOF__' > /home/cuckoo/boxcutter/floppy/prepare-cuckoo.bat
@setlocal EnableDelayedExpansion EnableExtensions

echo ==^> Disabling Windows Firewall

netsh advfirewall set allprofiles state off

echo ==^> Installing Python. Please wait...

if not defined PYTHON_URL set
PYTHON_URL=https://www.python.org/ftp/python/2.7.9/python-2.7.9.amd64.msi
```

Jose Ortiz, siltecon@gmail.com

```
for %%i in (%PYTHON_URL%) do set PYTHON_EXE=%%~nxi
set PYTHON_DIR=%TEMP%\python
set PYTHON_PATH=%PYTHON_DIR%\%PYTHON_EXE%

echo ==^> Creating "%PYTHON_DIR%"
mkdir "%PYTHON_DIR%"
pushd "%PYTHON_DIR%"

if exist "%SystemRoot%\_download.cmd" (
  call "%SystemRoot%\_download.cmd" "%PYTHON_URL%" "%PYTHON_PATH%"
) else (
  echo ==^> Downloading "%PYTHON_URL%" to "%PYTHON_PATH%"
  powershell -Command "(New-Object
System.Net.WebClient).DownloadFile('%PYTHON_URL%', '%PYTHON_PATH%')" <NUL
)
if not exist "%PYTHON_PATH%" goto exit1

powershell -Command (Start-Process %PYTHON_PATH% /qn -Wait)

echo ==^> Installing PIP. Please wait...

if not defined PIP_URL set PIP_URL=https://bootstrap.pypa.io/get-pip.py

for %%i in (%PIP_URL%) do set PIP_EXE=%%~nxi
set PIP_DIR=%TEMP%\python
set PIP_PATH=%PIP_DIR%\%PIP_EXE%

echo ==^> Creating "%PIP_DIR%"
mkdir "%PIP_DIR%"
pushd "%PIP_DIR%"

if exist "%SystemRoot%\_download.cmd" (
  call "%SystemRoot%\_download.cmd" "%PIP_URL%" "%PIP_PATH%"
) else (
  echo ==^> Downloading "%PIP_URL%" to "%PIP_PATH%"
  powershell -Command "(New-Object
System.Net.WebClient).DownloadFile('%PIP_URL%', '%PIP_PATH%')" <NUL
)
if not exist "%PIP_PATH%" goto exit1

C:\Python27\python %PIP_PATH%

echo ==^> Installing Pillow. Please wait...

C:\Python27\Scripts\pip install pillow
__EOF__

git add .
git commit -m 'updated JSON template'

#reference: http://httpd.apache.org/docs/2.2/
cat << '__EOF__' > /etc/apache2/sites-available/cuckoo_web
<IfModule mod_gnutls.c>
    NameVirtualHost *:8080
```

Jose Ortiz, siltecon@gmail.com

```
    Listen 8080
    <VirtualHost *:8080>
      ServerAdmin webmaster@localhost
      ServerName web.cuckoo.net.local
      WSGIDaemonProcess web.cuckoo.net.local user=cuckoo group=cuckoo
processes=1 threads=5 python-
path=/opt/cuckoo/web:/opt/cuckoo/web/web:/opt/cuckoo/lib
      WSGIProcessGroup web.cuckoo.net.local
      WSGIScriptAlias / /opt/cuckoo/web/web/wsgi.py
      <Directory /opt/cuckoo/web/web>
            WSGIApplicationGroup %{GLOBAL}
            AllowOverride All
            Order deny,allow
            Allow from all
      </Directory>

      Alias /static /opt/cuckoo/web/static
      ErrorLog ${APACHE_LOG_DIR}/cuckoo_web_ssl_error.log

      # Possible values include: debug, info, notice, warn, error, crit,
alert, emerg.
      LogLevel warn
      CustomLog ${APACHE_LOG_DIR}/cuckoo_web_ssl_access.log combined

      GnuTLSEnable on
      GnuTLSPriorities GnuTLSPriorities NONE:!VERS-SSL3.0:+VERS-
TLS1.0:+ARCFOUR-128:+RSA:+SHA1:+COMP-NULL
      GnuTLSCertificateFile    /etc/ssl/certs/ssl-cert-snakeoil.pem
      GnuTLSKeyFile /etc/ssl/private/ssl-cert-snakeoil.key

      BrowserMatch "MSIE [2-6]" \
      nokeepalive ssl-unclean-shutdown \
      downgrade-1.0 force-response-1.0
      # MSIE 7 and newer should be able to use keepalive
      BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown
    </VirtualHost>
</IfModule>
__EOF__

a2enmod gnutls
a2dissite default || true
a2dissite default-ssl || true
a2dissite default-tls || true
a2ensite cuckoo_web && service apache2 restart

chown -R ${CUCKOO_USER}.${CUCKOO_GROUP} ${CUCKOO_HOME}
chown -R ${CUCKOO_USER}.${CUCKOO_GROUP} ${CUCKOO_DIR}
```

Appendix B - Required files

The install script assumes these packages are available inside the working directory:

```
http://downloads.cuckoosandbox.org/1.2/
            cuckoo_1.2.tar.gz

https://pypi.python.org/packages/source/c/cybox/
            cybox-2.0.1.4.tar.gz

https://distorm.googlecode.com/files/
            distorm3.zip

https://dpkt.googlecode.com/files/
            dpkt-1.8.tar.gz

https://freddie.witherden.org/tools/libforensic1394/releases/
            libforensic1394-0.2.tar.gz

https://pypi.python.org/packages/source/m/maec/
            maec-4.0.1.0.tar.gz

https://dl.bintray.com/mitchellh/packer/
            packer_0.8.1_linux_amd64.zip

https://pypi.python.org/packages/source/p/
            pydeep/pydeep-0.2.tar.gz

https://github.com/google/rekall/releases/download/v1.2.1/
            rekall-forensic_1.2.1_amd64.deb

http://downloads.sourceforge.net/project/ssdeep/ssdeep-2.10/
            ssdeep-2.10.tar.gz

https://dl.bintray.com/mitchellh/vagrant/
            vagrant_1.7.2_x86_64.deb

http://downloads.volatilityfoundation.org/releases/2.4/
            volatility-2.4.tar.gz

http://download.virtualbox.org/virtualbox/4.3.18/
            Oracle_VM_VirtualBox_Extension_Pack-4.3.28-100309.vbox-extpack

https://github.com/plusvic/yara/archive/
      v3.3.0.tar.gz
```

Jose Ortiz, siltecon@gmail.com

Appendix C – Cuckoo Sandbox default recipe


```
cookbook_file "agent.pyw" do
  path "C:\\Python27\\Scripts\\agent.pyw"
  action :create_if_missing
end

registry_key
'HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run' do
  values [
    {
      :name => 'Cuckoo',
      :type => :string,
      :data => 'C:\\Python27\\Scripts\\agent.pyw'
    }
  ]
  action :create
end

node["cuckoosandbox"]["packages"].each do |package|
  cookbook_file "#{package["name"]}_#{package["version"]}.#{package["type"]}"
do
    path
"C:\\tmp\\#{package["name"]}_#{package["version"]}.#{package["type"]}"
    action :create_if_missing
  end

  windows_package
"#{package["name"]}_#{package["version"]}.#{package["type"]}" do
    source
"C:\\tmp\\#{package["name"]}_#{package["version"]}.#{package["type"]}"
    installer_type :custom
    options "#{package["options"]}"
    action :install
  end
end
```

Jose Ortiz, siltecon@gmail.com

Appendix D – Vagrantfile

This is an adaptation of a standard Vagrantfile. The chef.json data structure is my original work.

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

Vagrant.configure(2) do |config|

  config.vm.define :profile1 do |profile1|
    profile1.vm.box = "win7x64-pro"
    profile1.ssh.insert_key = false
    profile1.vm.provider "virtualbox" do |vb|
      vb.gui = true
      vb.name = "Windows7-Profile-1"
      vb.memory = "1024"
     end

    config.vm.provision "shell", :path => "files/prepare_cuckoo.bat"

    profile1.vm.provision "chef_solo" do |chef|
      chef.cookbooks_path = "cookbooks"
      chef.add_recipe "cuckoosandbox"
      chef.json = {
        "cuckoosandbox" => {
          "packages" => [
            {
              "name" => "adobereader", "version" => "9.2.0",
              "type" => "msi", "options" => "-ms EULA_ACCEPT=YES"
            }, {
              "name" => "firefox", "version" => "31.5.0",
              "type" => "msi", "options" => "-ms"
            }, {
              "name" => "flashplayer", "version" => "17",
              "type" => "msi", "options" => "-ms"
            }, {
              "name" => "googlechrome", "version" => "latest",
              "type" => "msi", "options" => "-ms"
            }, {
              "name" => "sunjre", "version" => "7u17",
              "type" => "exe", "options" => "/s"
            }
          ]
        }
      }
    end
  end
```

Jose Ortiz, siltecon@gmail.com

```
end
```

Appendix E – JSON template for box creation


```
################################################################
# Modified JSON template. Tested with packer v0.8.1 and commit
# d0a09ec of the boxcutter repo.
# reference: https://www.github.com/boxcutter/windows
################################################################
{
  "builders": [
    {
      "disk_size": 20480,
      "floppy_files": [
        "floppy/win7x64-pro/Autounattend.xml",
        "floppy/00-run-all-scripts.cmd",
        "floppy/install-winrm.cmd",
        "floppy/fixnetwork.ps1",
        "floppy/powerconfig.bat",
        "floppy/01-install-wget.cmd",
        "floppy/_download.cmd",
        "floppy/_packer_config.cmd",
        "floppy/passwordchange.bat",
        "floppy/networkprompt.bat",
        "floppy/disablewinupdate.bat",
        "floppy/uac-disable.bat",
        "floppy/prepare-cuckoo.bat",
        "floppy/openssh.bat",
        "floppy/zz-start-sshd.cmd",
        "floppy/oracle-cert.cer"
      ],
      "guest_os_type": "Windows7_64",
      "iso_checksum": "{{ user `iso_checksum` }}",
      "iso_checksum_type": "sha1",
      "iso_url": "{{ user `iso_url` }}",
      "shutdown_command": "{{ user `shutdown_command`}}",
      "ssh_password": "vagrant",
      "ssh_username": "vagrant",
      "ssh_wait_timeout": "10000s",
      "type": "virtualbox-iso",
      "vboxmanage": [
        [
          "modifyvm",
          "{{.Name}}",
          "--memory",
          "2048"
        ],
        [
          "modifyvm",
          "{{.Name}}",
          "--cpus",
          "1"
```

Jose Ortiz, siltecon@gmail.com

```
        ]
      ],
      "vm_name": "win7x64-pro"
    }
  ],
  "post-processors": [
    {
      "compression_level": 1,
      "keep_input_artifact": false,
      "output": "box/{{.Provider}}/win7x64-pro-{{user `cm`}}{{user
`cm_version`}}-{{user `version`}}.box",
      "type": "vagrant",
      "vagrantfile_template": "tpl/vagrantfile-win7x64-pro.tpl"
    }
  ],
  "provisioners": [
    {
      "environment_vars": [
        "CM={{user `cm`}}",
        "CM_VERSION={{user `cm_version`}}",
        "UPDATE={{user `update`}}"
      ],
      "execute_command": "{{.Vars}} cmd /c C:/Windows/Temp/script.bat",
      "remote_path": "/tmp/script.bat",
      "scripts": [
        "script/vagrant.bat",
        "script/cmtool.bat",
        "script/vmtool.bat",
        "script/clean.bat",
        "script/ultradefrag.bat",
        "script/uninstall-7zip.bat",
        "script/sdelete.bat"
      ],
      "type": "shell"
    },
    {
      "inline": [
        "rm -f /tmp/script.bat"
      ],
      "type": "shell"
    }
  ],
  "variables": {
    "cm": "chef",
    "cm_version": "",
    "iso_checksum": "708e0338d4e2f094dfeb860347c84a6ed9e91d0c",
    "iso_url":
"iso/en_windows_7_professional_with_sp1_vl_build_x64_dvd_u_677791.iso",
    "shutdown_command": "shutdown /s /t 10 /f /d p:4:1 /c 'Packer Shutdown'",
    "update": true,
    "version": "0.1.0"
  }
}
```

Jose Ortiz, siltecon@gmail.com

Appendix F – Sample chef repository layout

```
.
├── cookbooks
│   ├── cuckoosandbox
│   │   ├── attributes
│   │   │   └── default.rb
│   │   ├── CHANGELOG.md
│   │   ├── definitions
│   │   ├── files
│   │   │   └── default
│   │   │       ├── adobereader_9.2.0.msi
│   │   │       ├── adobereader_9.4.0.msi
│   │   │       ├── agent.pyw
│   │   │       ├── firefox_3.0.17.msi
│   │   │       ├── firefox_31.5.0.msi
│   │   │       ├── firefox_3.6.27.msi
│   │   │       ├── flashplayer_17.msi
│   │   │       ├── googlechrome_latest.msi
│   │   │       ├── sunjre_6u24.exe
│   │   │       └── sunjre_7u17.exe
│   │   ├── libraries
│   │   ├── metadata.rb
│   │   ├── providers
│   │   ├── README.md
│   │   ├── recipes
│   │   │   └── default.rb
│   │   ├── resources
│   │   └── templates
│   │       └── default
│   └── README.md
├── files
│   └── prepare_cuckoo.bat
├── LICENSE
├── Rakefile
├── README.md
└── Vagrantfile
```

Jose Ortiz, siltecon@gmail.com

Appendix G – Sample Packer repository layout

```
.
├── AUTHORS
├── bin
│   ├── compress.py
│   ├── ssh-box.sh
│   ├── test-box.sh
│   ├── test-vagrantcloud-box.sh
│   ├── tweak-json.py
│   ├── tweak-jsons.sh
│   └── upload.sh
├── box
│   └── virtualbox
├── CHANGELOG.md
├── floppy
│   ├── 00-run-all-scripts.cmd
│   ├── 01-install-wget.cmd
│   ├── bitvisessh.bat
│   ├── bitvisessh.cfg
│   ├── disablewinupdate.bat
│   ├── _download.cmd
│   ├── fixnetwork.ps1
│   ├── folderoptions.bat
│   ├── install-winrm.cmd
│   ├── networkprompt.bat
│   ├── openssh.bat
│   ├── oracle-cert.cer
│   ├── _packer_config.cmd
│   ├── passwordchange.bat
│   ├── powerconfig.bat
│   ├── time12h.bat
│   ├── time24h.bat
│   ├── uac-disable.bat
│   ├── uac-enable.bat
│   ├── _uninstall-bitvisessh.bat
│   ├── _uninstall-cygwin.bat
│   ├── unzip.vbs
│   ├── update.bat
│   ├── upgrade-wua.bat
│   ├── win7x64-pro
│   │   └── Autounattend.xml
│   ├── zz-start-sshd.cmd
│   └── zzz-debug-log.cmd
├── LICENSE
├── Makefile
├── README.md
├── script
│   ├── 01-install-handle.cmd
│   ├── clean.bat
│   ├── cmtool.bat
│   ├── dump-logs.cmd
│   ├── enable-rdp.cmd
```

Jose Ortiz, siltecon@gmail.com

```
│   ├── reboot.cmd
│   ├── regenerate-dotnet-cache.cmd
│   ├── save-logs.cmd
│   ├── save-temp-dirs.cmd
│   ├── sdelete.bat
│   ├── ultradefrag.bat
│   ├── uninstall-7zip.bat
│   ├── uninstall-openssh.cmd
│   ├── vagrant.bat
│   └── vmtool.bat
├── test
│   ├── spec_helper.rb
│   └── windows_spec.rb
├── tpl
│   └── vagrantfile-win7x64-pro.tpl
├── VERSION
├── win7x64-pro.json
└── wsim
    └── win7
        └── x64
            └── install_Windows 7 PROFESSIONAL.clg
```

Appendix H – Malware Sandbox resources

All links taken from https://zeltser.com/automated-malware-analysis/:

Anubis - http://anubis.iseclab.org/
BitBlaze Malware Analysis Service - https://aerie.cs.berkeley.edu/
Comodo Automated Analysis System – http://camas.comodo.com/
Valkyrie - http://valkyrie.comodo.com/
EUREKA Malware Analysis Internet Service - http://eureka.cyber-ta.org/
Joe Sandbox Document Analyzer - http://www.document-analyzer.net/
Malwr - https://malwr.com/submission/
VxStream Sandbox - https://www.hybrid-analysis.com/
ThreatExpert - http://www.threatexpert.com/submit.aspx
ViCheck - https://www.vicheck.ca/
VisualThreat - http://www.visualthreat.com/
XecScan - http://scan.xecure-lab.com/

Jose Ortiz, siltecon@gmail.com

Appendix I – Installation Checklist

As the root user:

- Clean install of Debian 7.6 amd64
- Preparation script completed without errors

As the Cuckoo user:

- Windows installation media copied to /home/cuckoo/boxcutter/.windows/isos
- Makefile.local created in /home/cuckoo/boxcutter
- Default box script run from /home/cuckoo/boxcutter:
  - `$ make virtualbox/win7x64-pro`
- Box imported into Vagrant library
  - `$ vagrant box add cuckoo/virtualbox box/virtualbox/win7x64-pro-cheflatest-1.0.4.box`
- Created vboxnet0 network:
  - `$ vboxmanage hostonlyif create`
  - `$ vboxmanage hostonlyif ipconfig vboxnet0 –ip 192.168.56.1`
- Prepared ~/chef-repo and created target
  - `$ knife cookbook site install windows –o cookbooks`
  - `$ knife cookbook create cuckoosandbox`
  - Recipes copied to ~/chef-repo/cookbooks/cuckoosandbox
  - Cuckoo agent.py copied to ~/chef-repo/cookbooks/cuckoosandbox/files/default
  - Packages copied to ~/chef-repo/cookbooks/cuckoosandbox/files/default
  - Initialized and edited the Vagrantfile
  - vagrant up; vagrant reload
  - Cuckoo agent is running and listening on port 8000
- Configure networking and take snapshot while the VM is running:
  - Log into Windows machine and change network interface properties:
    - 192.168.56.101/24, gateway 192.168.56.1
  - `$ vboxmanage controlvm <vm_name> acpipowerbutton`

Jose Ortiz, siltecon@gmail.com

- `$ vboxmanage modifyvm <vm_name> --hostonlyadapter vboxnet0`
- `$ vboxmanage modifyvm <vm_name> --nic1 hostonly`
- `$ vboxmanage startvm <vm_name> --type headless`
- `$ vboxmanage snapshot <vm_name> Snapshot1 --pause`
- `$ vboxmanage controlvm <vm_name> poweroff`
- `$ vboxmanage snapshot <vm_name> restorecurrent`

- Allow the guest VMs access to the Internet (optional). As the user root:
  - `# iptables –A FORWARD –o eth0 –i vboxnet0 –s 192.168.56.0/24 –m conntrack --ctstate NEW –j ACCEPT`
  - `# iptables –A FORWARD –m conntrack --ctstate ESTABLISHED,RELATED –j ACCEPT`
  - `# iptables –A POSTROUTING –t nat –j MASQUERADE`
  - `# echo 1 > /proc/sys/net/ipv4/ip_forward`

  Note: Each `iptables` command is a single line

- Configure the sandbox
  - cuckoo.conf:

    memory_dump = on

  - memory.conf:

    guest_profile = Win7SP1x64
    delete_memdump = yes
    [idt]
    enabled = no
    [gdt]
    enabled = no

  - processing.conf:

    [memory]
    enabled = yes

  - reporting.conf:

    [maec40]
    enabled = yes
    [mongodb]
    enabled = yes

Jose Ortiz, siltecon@gmail.com

- virtualbox.conf:

```
mode = headless
label = <vm_name>
ip = 192.168.56.101
resultserver_ip = 192.168.56.1
tags = windows-7-amd64,acrobat_reader_9,java_1.6
```

- Start the sandbox:
  - `$ cd /opt/cuckoo && ./cuckoo.py &`
  - `./utils/api.py --host 0.0.0.0 --port 8090 &`
- Access the sandbox on https://<ip_address>:8080
- Submit a file using the REST API:
  - `curl -F file=@/file/path`

    `http://cuckoo_address:8090/tasks/create/file`