Cuckoo Sandbox Book

Release 1.0

Cuckoo Sandbox

Nov 19, 2017

Contents

Contributed By Check Point Software Technologies LTD.

CuckooDroid is an extension of Cuckoo Sandbox the *Open Source* software for automating analysis of suspicious files. CuckooDroid brigs to cuckoo the capabilities of execution and analysis of android application.

This guide will explain how to configure the android guest machines and the cuckoo host to support android.

CHAPTER 1

Contents

1.1 Introduction

This is an introductory chapter to Cuckoo Sandbox. It explains some basic malware analysis concepts, what's Cuckoo and how it can fit in malware analysis.

1.1.1 Sandboxing

As defined by Wikipedia, "*in computer security, a sandbox is a security mechanism for separating running programs. It is often used to execute untested code, or untrusted programs from unverified third-parties, suppliers, untrusted users and untrusted websites.*".

This concept applies to malware analysis' sandboxing too: our goal is to run an unknown and untrusted application or file inside an isolated environment and get information on what it does.

Malware sandboxing is a practical application of the dynamical analysis approach: instead of statically analyzing the binary file, it gets executed and monitored in real-time.

This approach obviously has pros and cons, but it's a valuable technique to obtain additional details on the malware, such as its network behavior. Therefore it's a good practice to perform both static and dynamic analysis while inspecting a malware, in order to gain a deeper understanding of it.

Simple as it is, Cuckoo is a tool that allows you to perform sandboxed malware analysis.

Using a Sandbox

Before starting to install, configure and use Cuckoo, you should take some time to think on what you want to achieve with it and how.

Some questions you should ask yourself:

- What kind of files do I want to analyze?
- What volume of analyses do I want to be able to handle?

- Which platform do I want to use to run my analysis on?
- What kind of information I want about the file?

The creation of the isolated environment (the virtual machine) is probably the most critical and important part of a sandbox deployment: it should be done carefully and with proper planning.

Before getting hands on the virtualization product of your choice, you should already have a design plan that defines:

- Which operating system, language and patching level to use.
- Which software to install and which versions (particularly important when analyzing exploits).

Consider that automated malware analysis is not deterministic and its success might depend on a trillion of factors: you are trying to make a malware run in a virtualized system as it would do on a native one, which could be tricky to achieve and may not always succeed. Your goal should be both to create a system able to handle all the requirements you need as well as try to make it as realistic as possible.

For example you could consider leaving some intentional traces of normal usage, such as browsing history, cookies, documents, images etc. If a malware is designed to operate, manipulate or steal such files you'll be able to notice it.

Virtualized operating systems usually carry a lot of traces with them that makes them very easily detectable. Even if you shouldn't overestimate this problem, you might want to take care of this and try to hide as many virtualization traces as possible. There is a lot of literature on Internet regarding virtualization detection techniques and countermeasures.

Once you finished designing and preparing the prototype of system you want, you can proceed creating it and deploying it. You will be always in time to change things or slightly fix them, but remember that good planning at the beginning always means less troubles in the long run.

1.1.2 What is Cuckoo?

Cuckoo is an open source automated malware analysis system.

It's used to automatically run and analyze files and collect comprehensive analysis results that outline what the malware does while running inside an isolated Windows operating system.

It can retrieve the following type of results:

- Traces of win32 API calls performed by all processes spawned by the malware.
- Files being created, deleted and downloaded by the malware during its execution.
- Memory dumps of the malware processes.
- Network traffic trace in PCAP format.
- Screenshots of Windows desktop taken during the execution of the malware.
- Full memory dumps of the machines.

Some History

Cuckoo Sandbox started as a Google Summer of Code project in 2010 within The Honeynet Project. It was originally designed and developed by *Claudio "nex" Guarnieri*, who is still the main developer and coordinates all efforts from joined developers and contributors.

After initial work during the summer 2010, the first beta release was published on Feb. 5th 2011, when Cuckoo was publicly announced and distributed for the first time.

In March 2011, Cuckoo has been selected again as a supported project during Google Summer of Code 2011 with The Honeynet Project, during which *Dario Fernandes* joined the project and extended its functionality.

On November 2nd 2011 Cuckoo the release of its 0.2 version to the public as the first real stable release. On late November 2011 *Alessandro "jekil" Tanasi* joined the team expanding Cuckoo's processing and reporting functionality.

On December 2011 Cuckoo v0.3 gets released and quickly hits release 0.3.2 in early February.

In late January 2012 we opened Malwr.com, a free and public running Cuckoo Sandbox instance provided with a full fledged interface through which people can submit files to be analysed and get results back.

In March 2012 Cuckoo Sandbox wins the first round of the Magnificent7 program organized by Rapid7.

During the Summer of 2012 *Jurriaan "skier" Bremer* joined the development team, refactoring the Windows analysis component sensibly improving the analysis' quality.

On 24th July 2012, Cuckoo Sandbox 0.4 is released.

On 20th December 2012, Cuckoo Sandbox 0.5 "To The End Of The World" is released.

On 15th April 2013 we released Cuckoo Sandbox 0.6, shortly after having launched the second version of Malwr.com.

On 1st August 2013 *Claudio "nex" Guarnieri*, *Jurriaan "skier" Bremer* and *Mark "rep" Schloesser* presented Mo' Malware Mo' Problems - Cuckoo Sandbox to the rescue at Black Hat Las Vegas.

On 9th January 2014, Cuckoo Sandbox 1.0 is released.

In March 2014 Cuckoo Foundation born as non-profit organization dedicated to growth of Cuckoo Sandbox and the surrounding projects and initiatives.

On 7th April 2014, Cuckoo Sandbox 1.1 is released.

Obtaining Cuckoo

Cuckoo can be downloaded from the official website, where the stable and packaged releases are distributed, or can be cloned from our official git repository.

> **Warning:** While being more updated, including new features and bugfixes, the version available in the git repository should be considered an *under development* stage. Therefore its stability is not guaranteed and it most likely lacks updated documentation.

1.2 Installation

This chapter explains how to install Cuckoo.

Note: This documentation refers to *Host* as the underlying operating systems on which you are running Cuckoo (generally being a GNU/Linux distribution) and to *Guest* as the Windows virtual machine used to run the isolated analysis.

1.2.1 Preparing the Guest (Android on Linux Machine)

At this point, you should have configured the Cuckoo host component, as well as designed and defined the number and the names of the virtual machines you will use for malware execution.

Now it's time to create Linux machines and to configure them properly.

Guest Machine Architecture

Cuckoo Sandbox is a central management software for sample execution and analysis.

Each analysis is launched in a new and isolated virtual machine. Cuckoo's infrastructure is composed of a host machine (the management software) and a number of guest machines (virtual machines that perform the analysis).

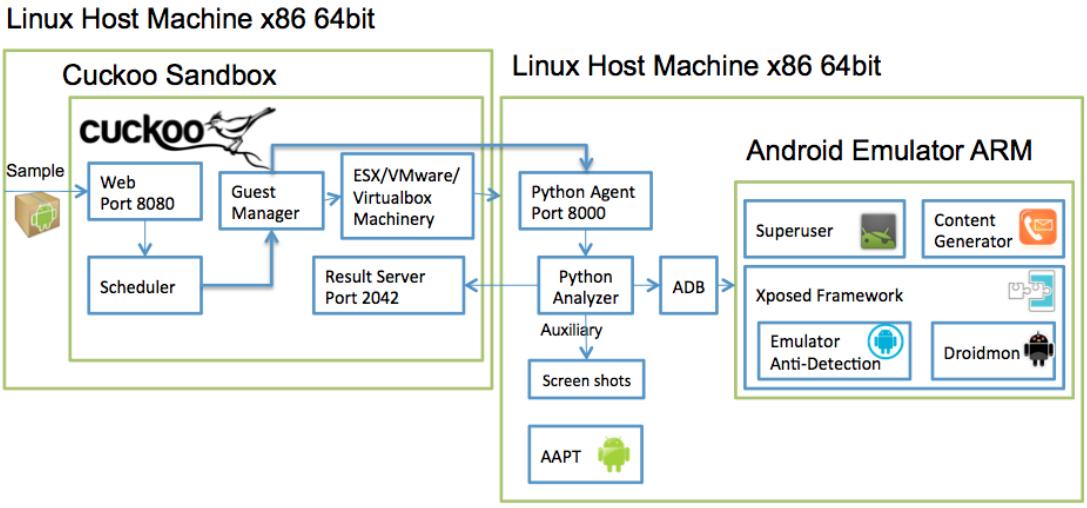
The host runs the core component of the sandbox that manages the entire analysis process, while the guests are the isolated environments where the malware samples are executed safely and then analyzed.

Each guest comprised of linux virtual machine that run android emulator, which is controlled by the machinery module. Additional components installed inside the linux machine to support the analysis process are:

- Python 2.7.
- Agent.py python script for communicating with the machine.
- linux analyzer component that is sent to the guest machine at the beginning of the analysis and controls the emulator.
- AAPT Arm - Android asset packaging tool compiled to arm for extracting the main activity and package name from the APK.
- ADB - android debug bridge binary for communicating with the emulator.

Additional components installed inside the android emulator to support the analysis process are:

- Xposed - a framework for modules that can change the behavior of the system and apps without affecting any APKs. We created 2 additional modules with this framework:
- Droidmon - Dalvik API call monitoring module.
- Emulator anti-detection - a collection of known anti-detection techniques for hiding the android emulator.
- Superuser app - grants and manages Superuser rights for your phone.
- Content Generator - generates a random contact list for a more realistic appearance.



Host Configuration

`conf/cuckoo.conf` configuration:

```
# Specify the name of the machinery module to use, this module will
# define the interaction between Cuckoo and your virtualization software
# of choice.
machinery = virtualbox

[resultserver]
# The Result Server is used to receive in real time the behavioral logs
# produced by the analyzer.
# Specify the IP address of the host. The analysis machines should be able
# to contact the host through such address, so make sure it's valid.
# NOTE: if you set resultserver IP to 0.0.0.0 you have to set the option
# `resultserver_ip` for all your virtual machines in machinery configuration.
ip = 192.168.56.1
```

`conf/virtualbox.conf` configuration:

```
[android_on_linx]
label = android_on_linx
platform = android_on_linux
ip = 192.168.56.201
snapshot = clean_snapshot
interface = vboxnet0
resultserver_ip = 192.168.56.1
resultserver_port = 2042
```

`conf/auxiliary.conf` configuration:

```
[sniffer]
# Enable or disable the use of an external sniffer (tcpdump) [yes/no].
enabled = yes
```

`conf/processing.conf` configuration:

```
[droidmon]
enabled = yes

[googleplay]
enabled = yes
android_id = <add android_id>
google_login = <add google_login>
google_password = <add google_password>

[apkinfo]
enabled = yes
#Decompiling dex with androguard in a heavy operation and for a big dex's
#he can really consume performance from the cuckoo host ,so it's recommended to limit␣
↪the size of dex that you will decompile
#decompilation_threshold=2000000
```

`conf/reporting.conf` configuration:

```
[reporthtml]
enabled = no

[reportandroidhtml]
enabled = yes
```

Creation of the Linux Virtual Machine

Once you have properly installed your virtualization software, you can create all the virtual machines you need.

Please refer to the official documentation, as the use and configuration of virtualization software is beyond the scope of this guide.

Note: Hints and considerations on how to design and create your virtualized environment can be found in the *Sandboxing* chapter.

Note: Cuckoo Sandbox can work with any Linux distribution. For analysis purposes, use Ubuntu Linux 12.04.05.

Cuckoo doesn't require any specific configuration to create the virtual machine. Choose the options that best fit your needs.

Requirements

To make Cuckoo run properly in your virtualized Linux system, you must install some required software and libraries.

Additional Software

Linux dependencies are required:

```
$ sudo apt-get install openjdk-7-jre libstdc++6:i386 libgcc1:i386 zlib1g:i386␣
↪libncurses5:i386
```

Install Android SDK

Android SDK is a strict requirement for the Cuckoo android_on_linux guest component (*analyzer*) to run properly.

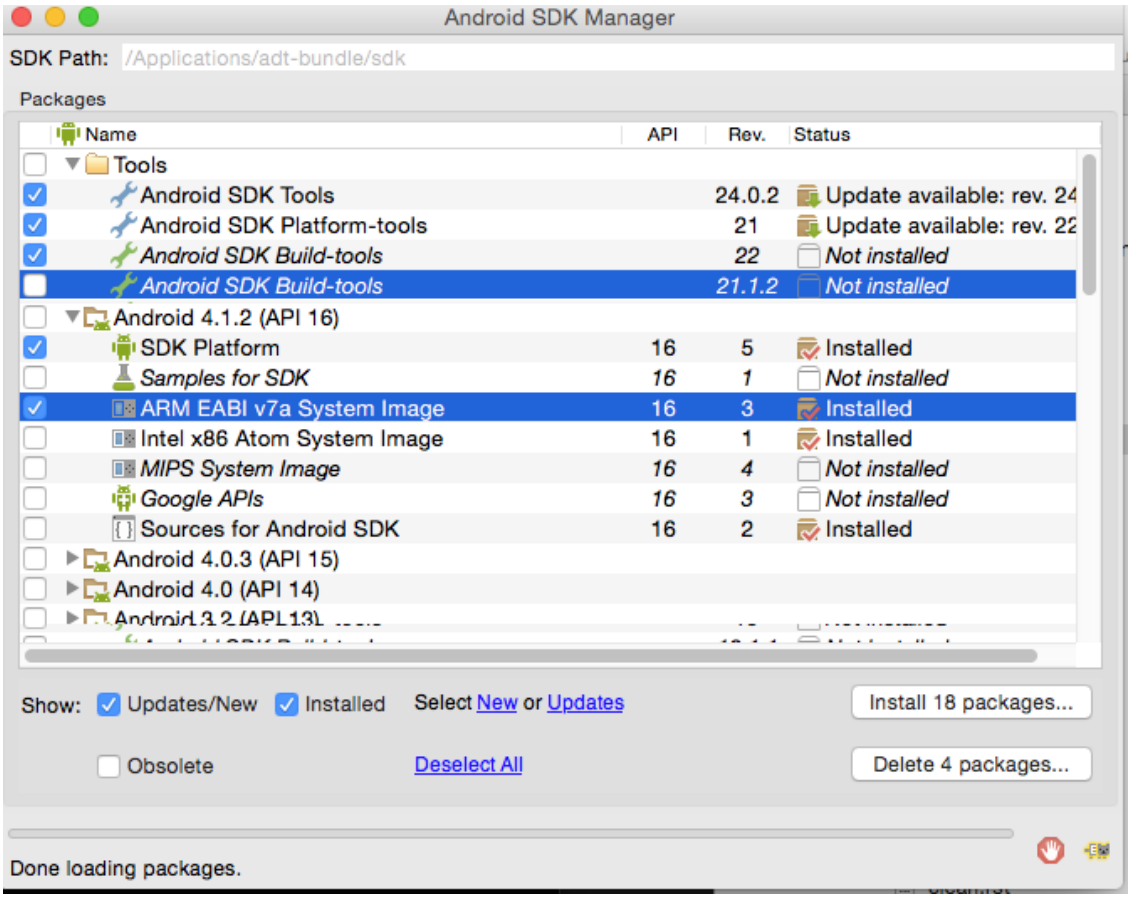
Download the latest SDK from the official website.

After downloading the sdk, go to the folder containing the .tgz file:

```
$ tar -xvf android-sdk_r24.0.2-linux.tgz
$ cd android-sdk
$ tools/android
```

In the Android SDK Manager, install the following components:

- **Tools**
  - Android SDK Tools
  - Android Platform-tools Tools
  - newest Android SDK Tools
- **Android 4.1.2 (API 16)**
  - SDK Platform
  - ARM EABI v7a System Image



Add the android SDK tool to `$PATH` variable:

```
$ export PATH=$PATH:sdk_path/tool:sdk_path/build-tools/x.x.x.x/:sdk_path/platform-
↪tools
```

Create Android Virtual Device

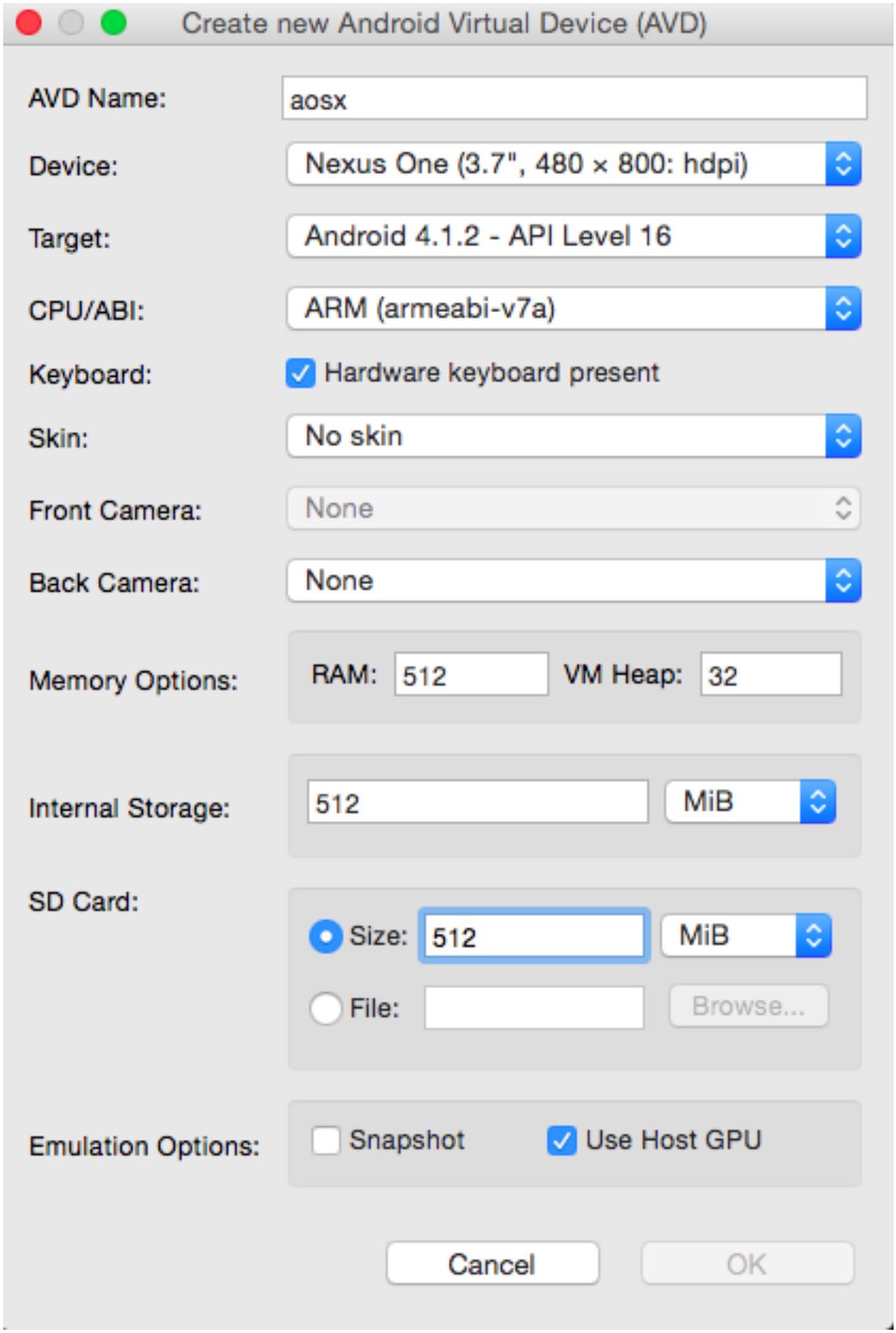
Start the Android Virtual Device Manager:

```
$ android avd
```

Press `Create..` and add the following configurations:

```
AVD Name - aosx
Device - Nexus One
Target - android 4.1.2
Cpu/Abi - arm
Ram - 512mb
Vm Heap - 32
Internal Storage - 512mb
Sdcard size - 512 mib
Emulation options - use host GPU
```

and click OK.


Create new Android Virtual Device (AVD)
AVD Name:
aosx
Device:
Nexus One (3.7", 480 × 800: hdpi)
Target:
Android 4.1.2 - API Level 16
CPU/ABI:
ARM (armeabi-v7a)
Keyboard:
Hardware keyboard present
Skin:
No skin
Front Camera:
None
Back Camera:
None
Memory Options:
RAM:
512
VM Heap:
32
Internal Storage:
512
MiB
SD Card:
Size:
512
MiB
File:
Browse...
Emulation Options:
Snapshot
Use Host GPU
Cancel
OK

Prepare the Android Virtual Device for Analysis

Copy the folder `utils/android_emulator_creator` to the Linux guest machine:

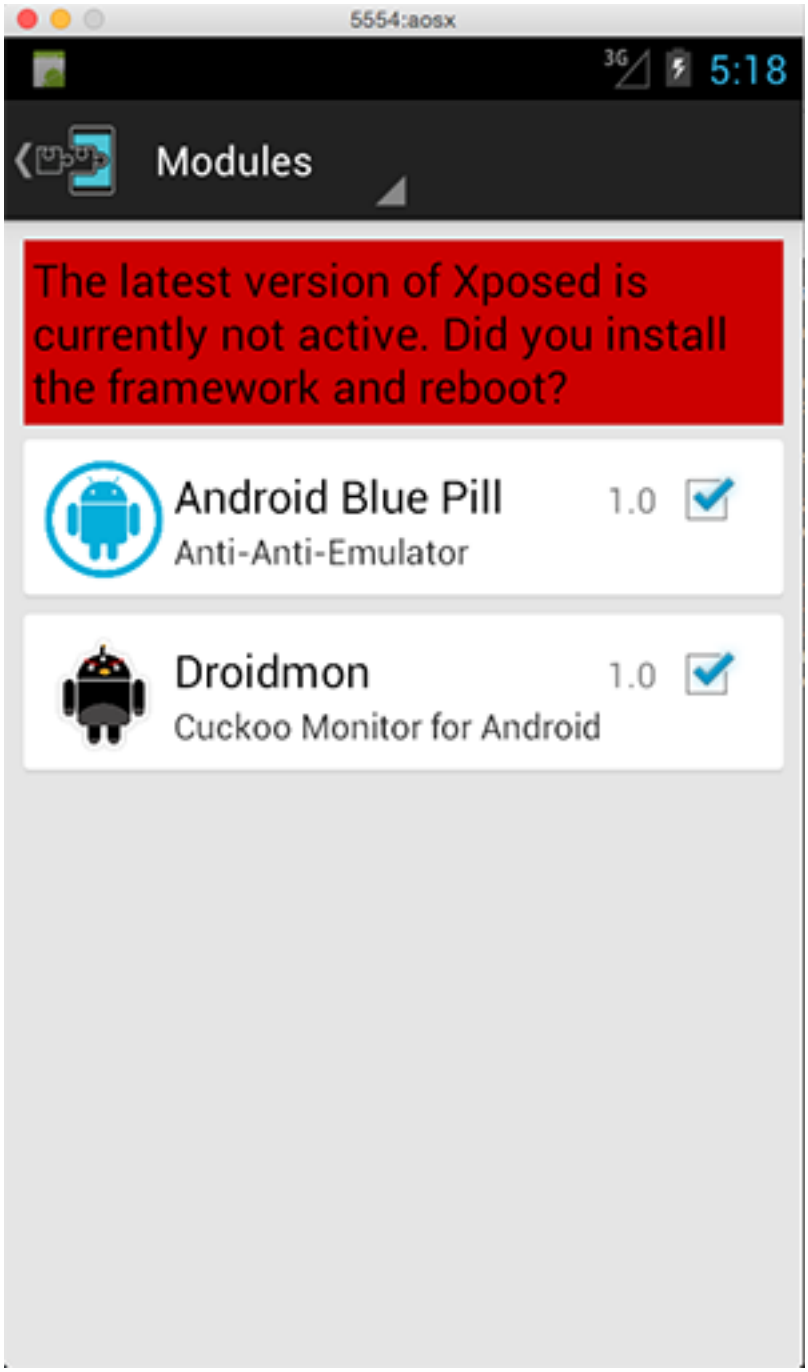
```
$ cd android_emulator_creator_path/
```

Start the emulator with `/system` in read-write mode:

```
$ emulator -avd aosx -qemu -nand -system,size=0x1f400000,file=<sdk_path>/system-
↪images/android-16/default/armeabi-v7a/system.img&
```

Run the script in `utils/android_emulator_creator/create_guest_android_on_linux.sh`

- Press settings->security->screenlock->none
- Press settings->Display->sleep->30 minutes
- Start Generate contacts app
- Start Supersuser app
- Start xposedinstaller app
- In Modules, check both packages `Droidmon` , `Android Blue Pill`



- Press framework -> install -> cancel-> soft reboot

Leave the emulator on for the snapshot.

Network Configuration

It's time to set up the network for your virtual machine.

Linux Settings

Before configuring the underlying networking of the virtual machine, you may want to tweak some settings inside Linux.

In the file `/etc/network/interfaces`, edit `eth0` settings. For example, the virtualbox network:

```
auto eth0
iface eth0 inet static
address 192.168.56.101
netmask 255.255.255.0
network 192.168.56.0
broadcast 192.168.56.255
gateway 192.168.56.1
```

Virtual Networking

You need to decide how to make your virtual machine access the Internet or your local network.

In previous releases, Cuckoo used shared folders to exchange data between the host and guests. From release 0.4 on, it uses a custom agent that communicates over the network with a simple XMLRPC protocol.

To make it work properly, configure your machine's network so that the host and the guest can communicate. To ensure the virtual network was set up correctly, test the network access by pinging a guest. Use only static IP addresses for your guest, as Cuckoo doesn't support DHCP; using it will break your setup.

This stage depends heavily on your requirements and the characteristics of your virtualization software.

> **Warning:** Virtual networking errors! Virtual networking is a vital component for Cuckoo; be sure there is connectivity between host and guest. Most of the issues reported by users are related to an incorrect networking setup. Check your virtualization software documentation and test connectivity with ping and telnet.

The recommended setup uses a host-only networking layout with proper forwarding and filtering configuration done with `iptables` on the host.

For example, using VirtualBox, you can enable Internet access to the virtual machines using the following `iptables` rules: (Assuming that eth0 is your outgoing interface, vboxnet0 is your virtual interface and 192.168.56.0/24 is your subnet address).

> 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

Add IP forwarding:

```
sysctl -w net.ipv4.ip_forward=1
```

Installing the Agent

From release 0.4 on, Cuckoo has a custom agent that runs inside the guest and handles communication and the exchange of data with the host. This agent is designed to be cross-platform, and can therefore be used on Windows as well as on Linux and OS X. To make Cuckoo work properly, install and start this agent.

It's very simple.

In the *agent/* directory you will find an *agent.py* file. Copy it to the guest operating system (in any way you want, perhaps a temporary shared folder or by downloading it from a host webserver) and run it. This launches the XMLRPC server which will be listening for connections.

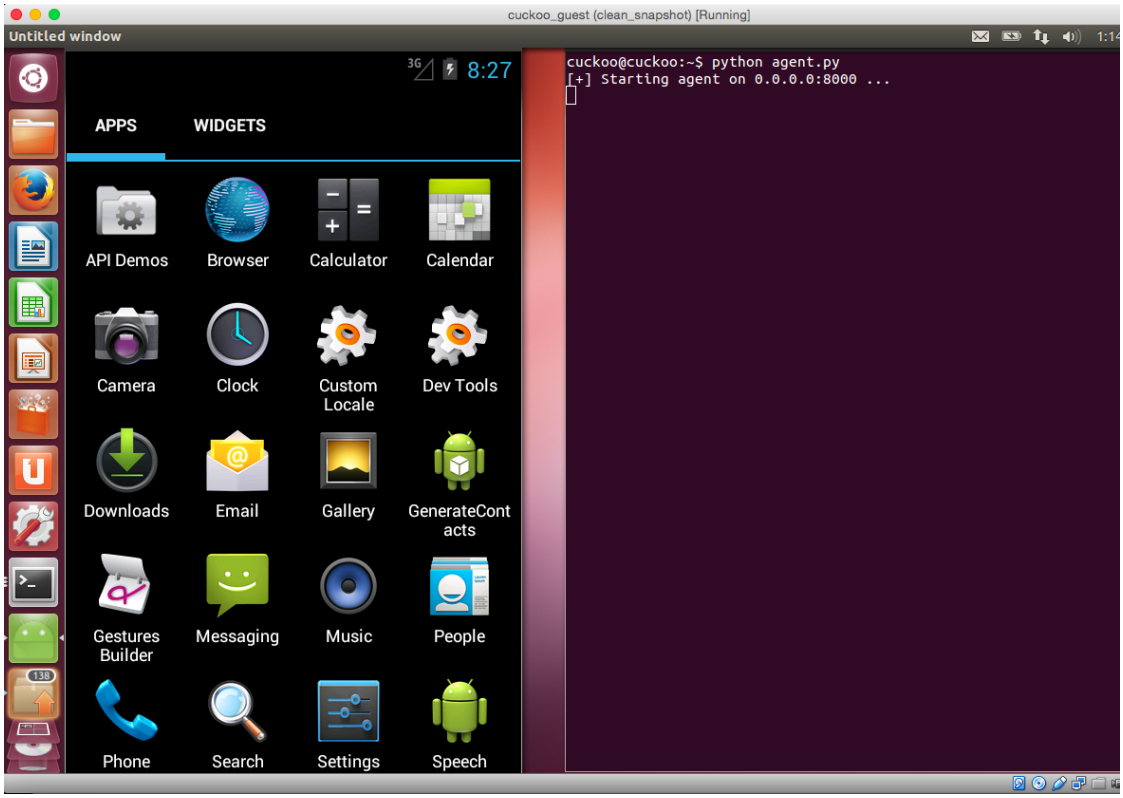
On Linux, launch the script from terminal window,:

```
$ python agent.py
```

Saving the Virtual Machine

You are now ready to save the virtual machine to a snapshot state.

The `agent.py` script must be running and the `android emulator` fully configured.



Proceed to save the machine. The exact method depends on your virtualization software.

If you follow all of the following steps properly, your virtual machine is ready for use.

VirtualBox

With VirtualBox, you can take the snapshot from the graphical user interface or from the command line:

```
$ VBoxManage snapshot "<Name of VM>" take "<Name of snapshot>" --pause
```

After the snapshot creation is completed, you can power off the machine and restore it:

```
$ VBoxManage controlvm "<Name of VM>" poweroff
$ VBoxManage snapshot "<Name of VM>" restorecurrent
```

VMware Workstation

With VMware Workstation, you can take the snapshot from the graphical user interface or from the command line:

```
$ vmrun snapshot "/your/disk/image/path/wmware_image_name.vmx" your_snapshot_name
```

Your_snapshot_name is the name you choose for the snapshot. Power off the machine from the GUI or from the command line:

```
$ vmrun stop "/your/disk/image/path/wmware_image_name.vmx" hard
```

Cloning the Virtual Machine

If you plan to use more than one virtual machine, there's no need to repeat all the steps done so far: you can clone your machine. This way, you'll have a copy of the original virtualized Windows with all the requirements already installed.

However, the clone will also contain all the settings of the original virtual machine. Repeat the steps explained in *Network Configuration*, *Installing the Agent* and *Saving the Virtual Machine* for this new machine.

1.2.2 Preparing the Guest (Android Emulator)

Now it's time to create the Android Emulator and to configure it properly.

> **Warning:** Only one emulator machine is supported! Due to the networking configuration, we are currently only able to support one machine at a time. This issue will be dealt with in the future.

Guest Machine Architecture

Cuckoo Sandbox is a central management software for sample execution and analysis.

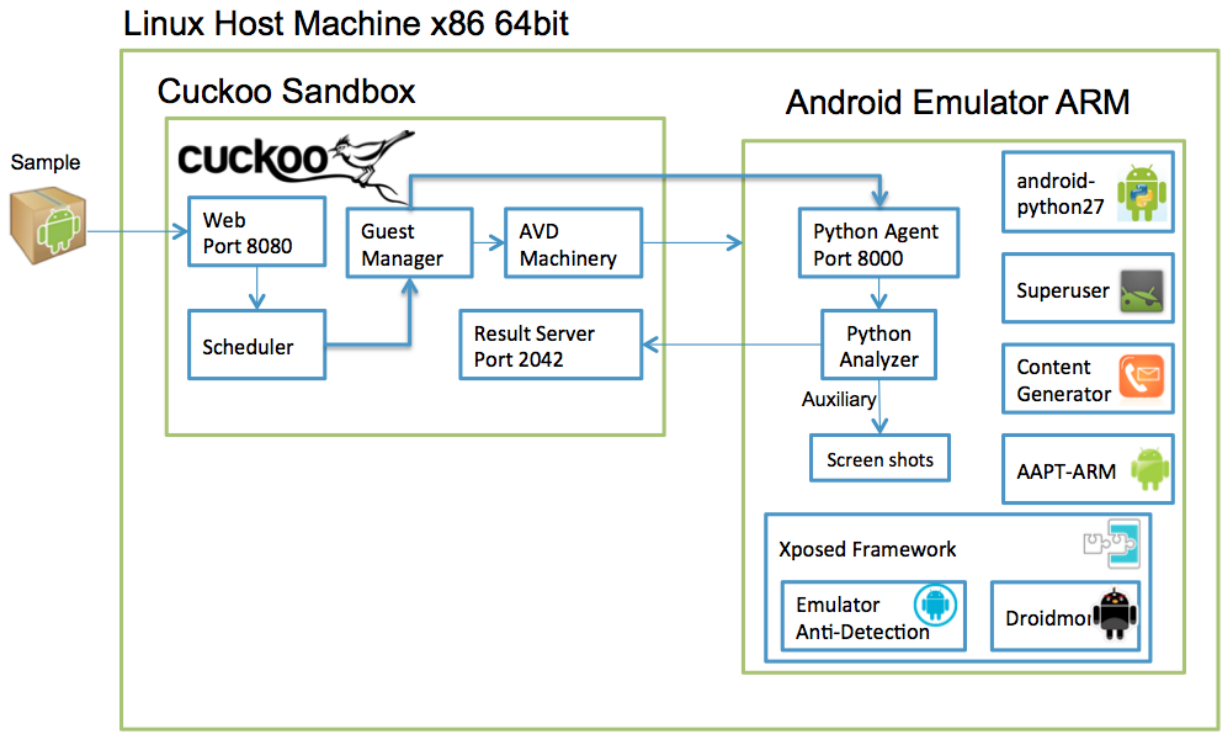
Each analysis is launched in a new and isolated virtual machine. Cuckoo's infrastructure is composed of a host machine (the management software) and a number of guest machines (virtual machines that perform the analysis).

The host runs the core component of the sandbox that manages the entire analysis process, while the guests are the isolated environments where the malware samples are executed safely and then analyzed.

Each guest comprised of android emulator which is controlled by the machinery module avd.py. Additional components installe

- Python 2.7 compiled to arm.
- Agent.py python script modified for the android emulator.
- Android analyzer component that is sent to the guest machine at the beginning of the analysis.
- Xposed - a framework for modules that can change the behavior of the system and apps without affecting any APKs. We created 2 additional modules with this framework:
- Droidmon - Dalvik API call monitoring module.
- Emulator anti-detection - a collection of known anti-detection techniques for hiding the android emulator.
- Superuser app - grants and manages Superuser rights for your phone.
- Content Generator - generates a random contact list for a more realistic appearance.

- AAPT Arm - Android asset packaging tool compiled to arm for extracting the main activity and package name from the APK.



Host Configuration

`conf/cuckoo.conf` configuration:

```
# Specify the name of the machinery module to use, this module will
# define the interaction between Cuckoo and your virtualization software
# of choice.
machinery = avd

[resultserver]
# The Result Server is used to receive in real time the behavioral logs
# produced by the analyzer.
# Specify the IP address of the host. The analysis machines should be able
# to contact the host through such address, so make sure it's valid.
# NOTE: if you set resultserver IP to 0.0.0.0 you have to set the option
# `resultserver_ip` for all your virtual machines in machinery configuration.
ip = 127.0.0.1
```

`conf/avd.conf` configuration:

```
[avd]
#Path to the local installation of the android emulator
emulator_path = <add>

#Path to the local installation of the adb - android debug bridge utility.
adb_path = <add>

#Path to the emulator machine files is located
```

```
avd_path = <add home_path>/.android/avd

#name of the reference machine that is used to duplicate
reference_machine = aosx

# Specify a comma-separated list of available machines to be used. For each
# specified ID you have to define a dedicated section containing the details
# on the respective machine. (E.g. aosx_1,aosx_2,aosx_3)
#currently supports only 1 machine for network limitations
machines =aosx_1

[aosx_1]
# Specify the label name of the current machine as specified in your
# aosx_1 configuration.
label = aosx_1

# Specify the operating system platform used by current machine
platform = android

# Specify the IP address of the current virtual machine. Make sure that the
# IP address is valid and that the host machine is able to reach it. If not,
# the analysis will fail.
# its always 127.0.0.1 because android emulator networking configurations this the␣
↪loopback of the host machine
ip = 127.0.0.1

#Specify the port for the emulator as your adb sees it.
emulator_port=5554

#10.0.2.2 is the loopback of the host machine very importent!!!
resultserver_ip = 10.0.2.2

resultserver_port = 2042
```

Warning: result server ip is always 10.0.2.2! (android emulator network configuration)

`conf/auxiliary.conf` configuration:

```
[sniffer]
# Enable or disable the use of an external sniffer (tcpdump) [yes/no].
enabled = no
```

`conf/processing.conf` configuration:

```
[droidmon]
enabled = yes

[googleplay]
enabled = yes
android_id = <add android_id>
google_login = <add google_login>
google_password = <add google_password>

[apkinfo]
enabled = yes
#Decompiling dex with androguard in a heavy operation and for a big dex's
```

```
#he can really consume performance from the cuckoo host ,so it's recommended to limit␣
↪the size of dex that you will decompile
#decompilation_threshold=2000000
```

`conf/reporting.conf` configuration:

```
[reporthtml]
enabled = no

[reportandroidhtml]
enabled = yes
```

Requirements

To make Cuckoo run properly with the Android Emulator, install these required software and libraries on the Cuckoo host.

Additional Software

Linux dependencies are required:

```
$ sudo apt-get install openjdk-7-jre libstdc++6:i386 libgcc1:i386 zlib1g:i386␣
↪libncurses5:i386
```

Install Android SDK

Android SDK is a strict requirement for the Cuckoo android_on_linux guest component (*analyzer*) to run properly.

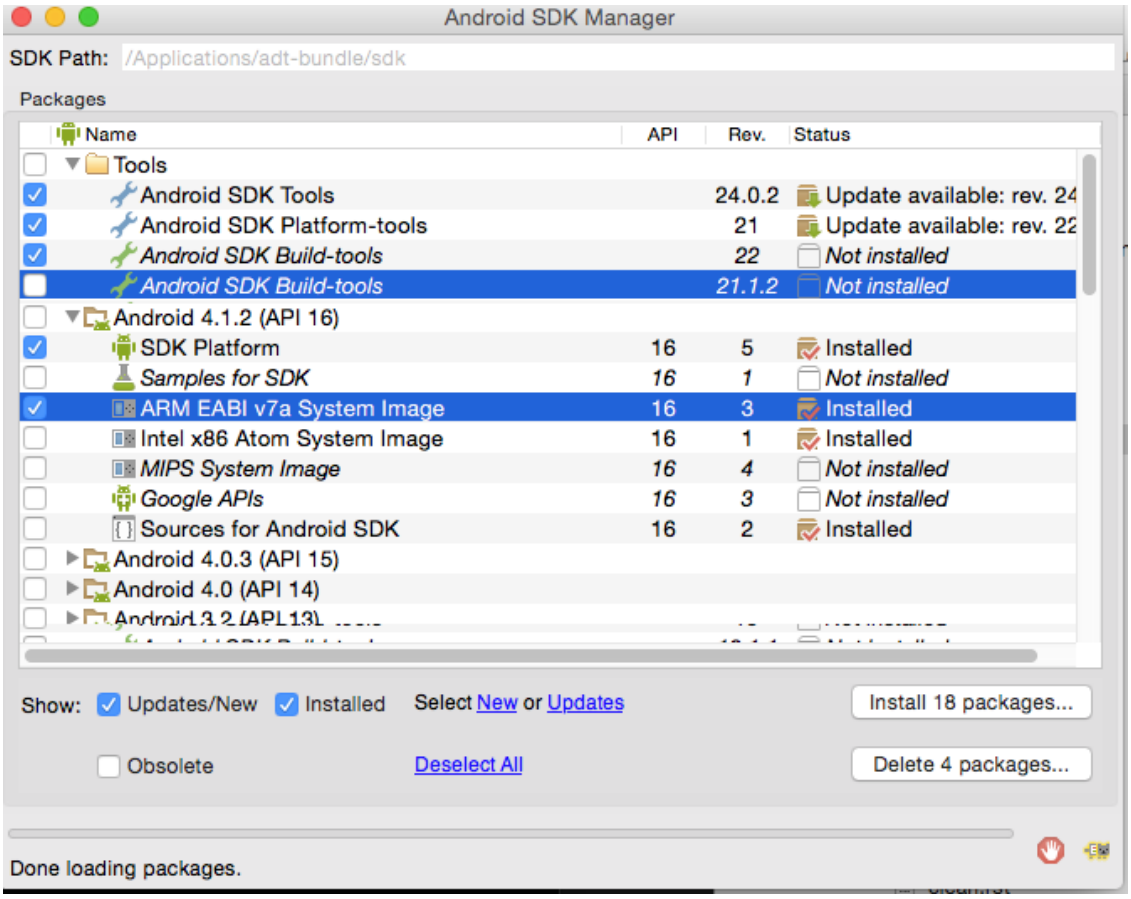
Download the latest SDK from the official website.

After downloading the SDK, go to the folder containing the .tgz file:

```
$ tar -xvf android-sdk_r24.0.2-linux.tgz
$ cd android-sdk
$ tools/android
```

In The Android SDK Manager, check to install the following components:

- **Tools**
  - Android SDK Tools
  - Android Platform-tools Tools
  - newest Android SDK Tools
- **Android 4.1.2 (API 16)**
  - SDK Platform
  - ARM EABI v7a System Image



The android SDK tool needs to be added to the `$PATH` variable:

```
$ export PATH=$PATH:sdk_path/tool:sdk_path/build-tools/x.x.x.x/:sdk_path/platform-
↪tools
```

Create Android Virtual Device

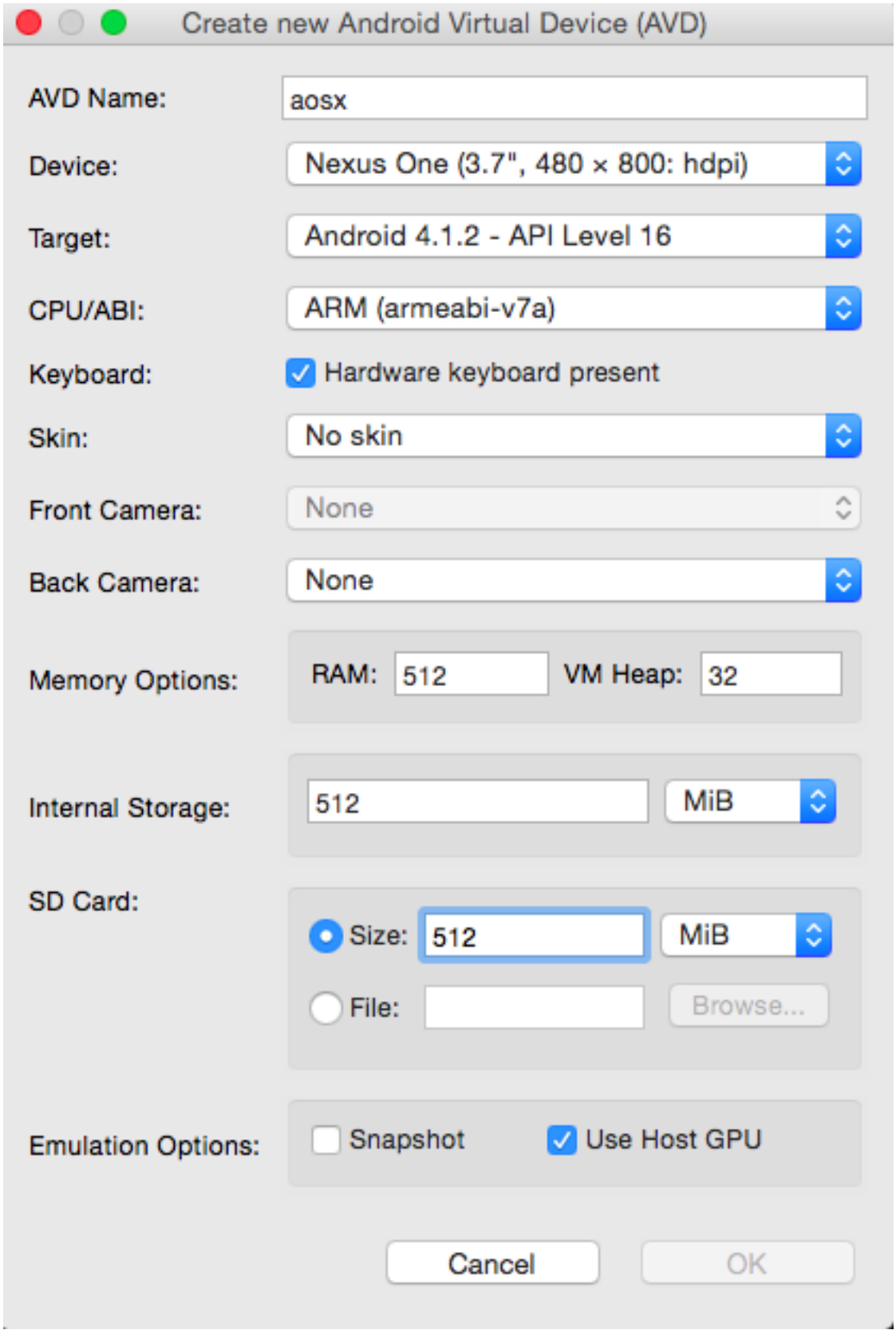
Start the Android Virtual Device Manager:

```
$ android avd
```

Press `Create..` and add the following configurations:

```
AVD Name - aosx
Device - Nexus One
Target - android 4.1.2
Cpu/Abi - arm
Ram - 512mb
Vm Heap - 32
Internal Storage - 512mb
Sdcard size - 512 mib
Emulation options - use host GPU
```

and click OK.


Create new Android Virtual Device (AVD)
AVD Name:
aosx
Device:
Nexus One (3.7", 480 × 800: hdpi)
Target:
Android 4.1.2 - API Level 16
CPU/ABI:
ARM (armeabi-v7a)
Keyboard:
Hardware keyboard present
Skin:
No skin
Front Camera:
None
Back Camera:
None
Memory Options:
RAM:
512
VM Heap:
32
Internal Storage:
512
MiB
SD Card:
Size:
512
MiB
File:
Browse...
Emulation Options:
Snapshot
Use Host GPU
Cancel
OK

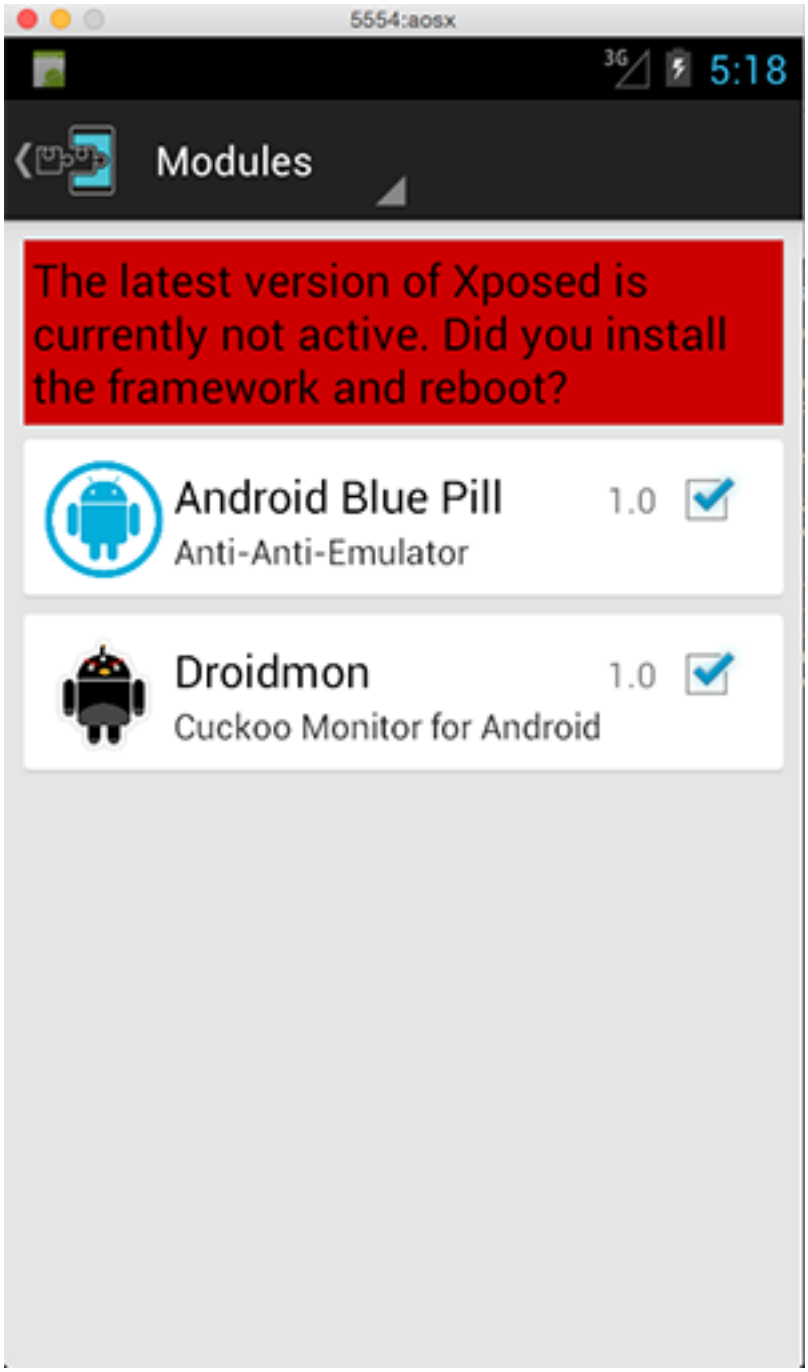
Prepare the Android Virtual Device Reference Machine for Analysis

Start the emulator with `/system` in read-write mode:

```
$ emulator -avd aosx -qemu -nand -system,size=0x1f400000,file=<sdk_path>/system-
↪images/android-16/default/armeabi-v7a/system.img&
```

Run the script in `utils/android_emulator_creator/create_guest_avd.sh`

- Press settings->security->screenlock->none
- Press settings->Display->sleep->30 minutes
- Start Generate contacts app
- Start Supersuser app
- Start xposedinstaller app
- In Modules, check both packages `Droidmon` , `Android Blue Pill`



- Press framework -> install -> cancel-> soft reboot

After the reboot, close the machine.

You have now created a reference machine to duplicate each analysis instead of reverting to snapshot.

1.2.3 Preparing the Guest (Android Device Cross-platform)

At this point, you should have configured the Cuckoo host component, as well as designed and defined the number and the names of the virtual machines you will use for malware execution.

Now it's time to create Android machines and to configure them properly.

Guest Machine Architecture

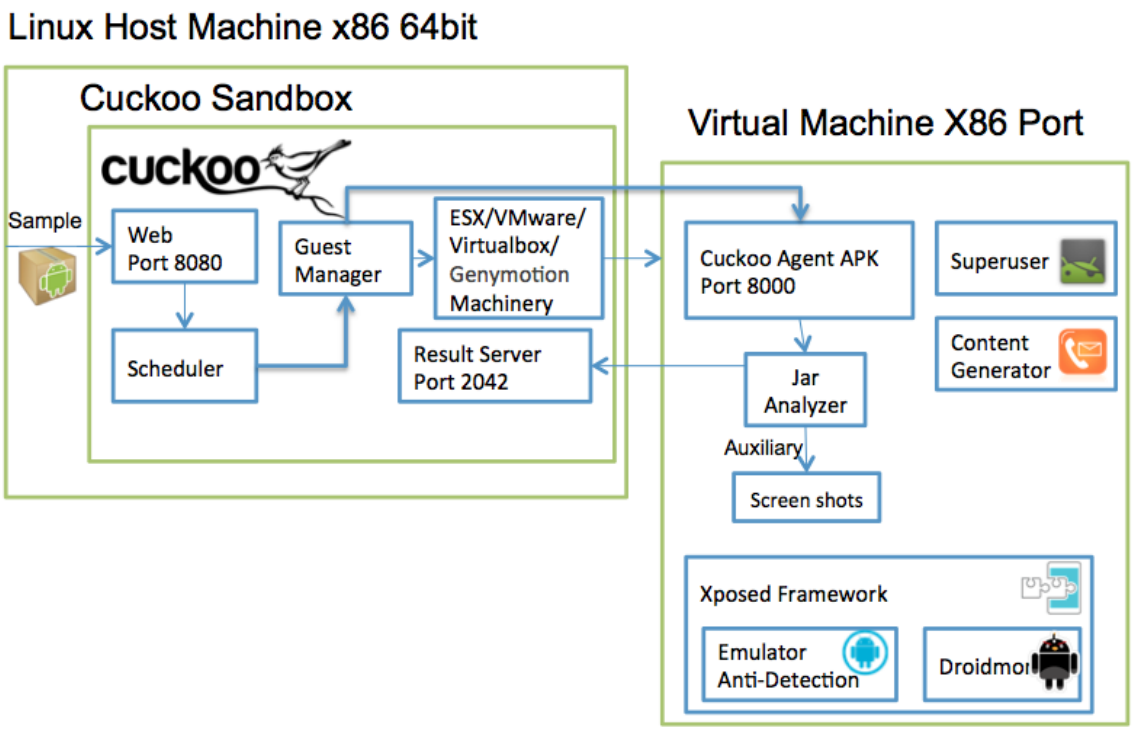
Cuckoo Sandbox is a central management software for sample execution and analysis.

Each analysis is launched in a new and isolated virtual machine. Cuckoo's infrastructure is composed of a host machine (the management software) and a number of guest machines (virtual machines that perform the analysis).

The host runs the core component of the sandbox that manages the entire analysis process, while the guests are the isolated environments where the malware samples are executed safely and then analyzed.

Each guest comprised of any android virtual machine (example: android x86 port) which is controlled by the machinery modul

- CuckooAgent.apk java android application that support the cuckoo protocol.
- analyzer.jar java package analyzer component that is sent to the guest machine at the beginning of the analysis.
- Xposed - a framework for modules that can change the behavior of the system and apps without affecting any APKs. We created 2 additional modules with this framework:
- Droidmon - Dalvik API call monitoring module.
- Emulator anti-detection - a collection of known anti-detection techniques for hiding the android emulator.
- Superuser app (already installed)- grants and manages Superuser rights for your phone.
- Content Generator - generates a random contact list for a more realistic appearance.



Host Configuration

`conf/cuckoo.conf` configuration:

```
# Specify the name of the machinery module to use, this module will
# define the interaction between Cuckoo and your virtualization software
# of choice.
machinery = virtualbox

[resultserver]
# The Result Server is used to receive in real time the behavioral logs
# produced by the analyzer.
# Specify the IP address of the host. The analysis machines should be able
# to contact the host through such address, so make sure it's valid.
# NOTE: if you set resultserver IP to 0.0.0.0 you have to set the option
# `resultserver_ip` for all your virtual machines in machinery configuration.
ip = 192.168.56.1
```

`conf/virtualbox.conf` configuration:

```
[cuckoo_android_x86]
label = cuckoo_android_x86
platform = android_device
ip = 192.168.56.201
snapshot = clean_snapshot
interface = vboxnet0
resultserver_ip = 192.168.56.1
resultserver_port = 2042
```

`conf/auxiliary.conf` configuration:

```
[sniffer]
# Enable or disable the use of an external sniffer (tcpdump) [yes/no].
enabled = yes
```

`conf/processing.conf` configuration:

```
[droidmon]
enabled = yes

[googleplay]
enabled = yes
android_id = <add android_id>
google_login = <add google_login>
google_password = <add google_password>

[apkinfo]
enabled = yes
#Decompiling dex with androguard in a heavy operation and for a big dex's
#he can really consume performance from the cuckoo host ,so it's recommended to limit␣
↪the size of dex that you will decompile
#decompilation_threshold=2000000
```

`conf/reporting.conf` configuration:

```
[reporthtml]
enabled = no
```

```
[reportandroidhtml]
enabled = yes
```

Creation of the Linux Virtual Machine

Once you have properly installed your virtualization software, you can create all the virtual machines you need.

Using and configuring your virtualization software is beyond the scope of this guide, so please refer to the official documentation.

Note: Hints and considerations on how to design and create your virtualized environment can be found in the *Sandboxing* chapter.

Note: Cuckoo Sandbox works with any Android x86 port versions. For analysis purposes, use Android x86 4.4 RC2. Download the `.iso` from the official website.

Network Configuration

Now it's time to set up the network for your virtual machine.

Android Settings

Before configuring the underlying networking of the virtual machine, you may wish to tweak some settings inside Linux.

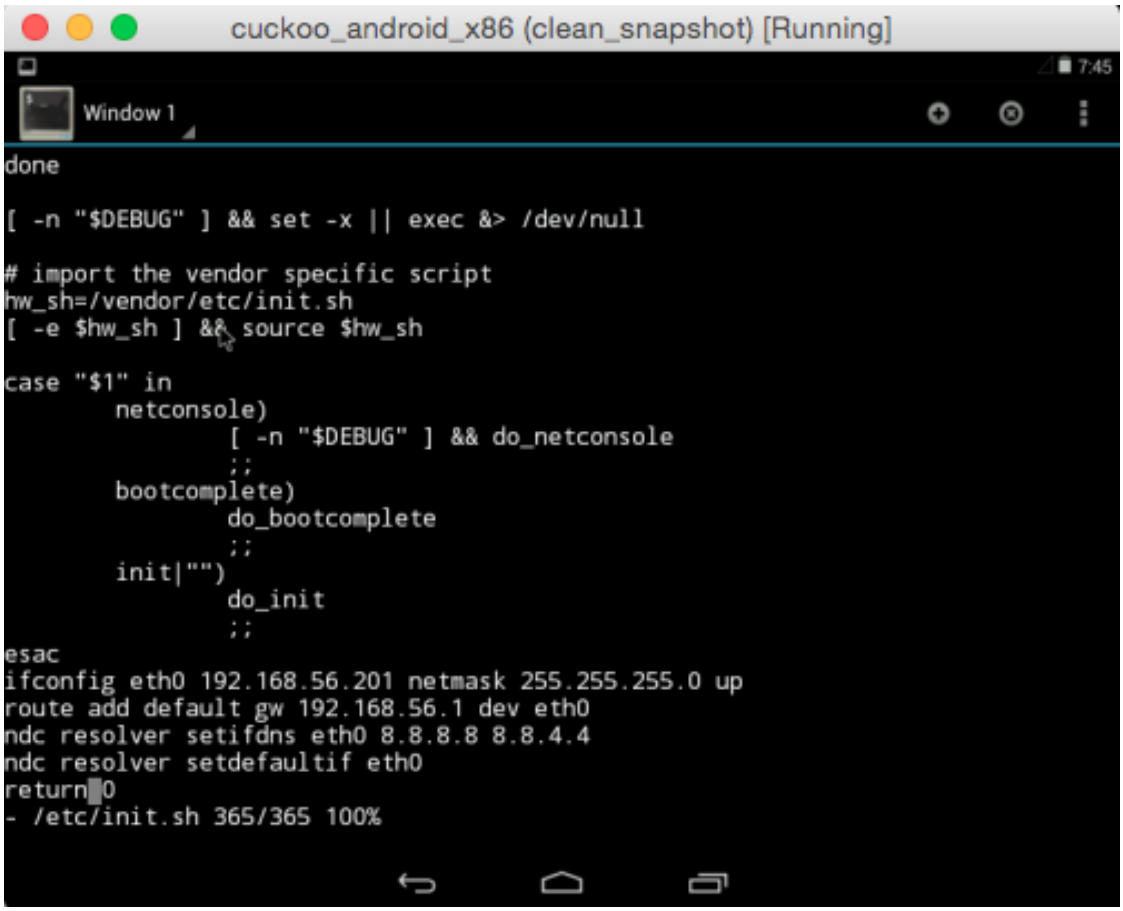
Open the Terminal app inside the machine and change to `su`. In the file `/etc/init.sh`, edit `eth0` settings. For example, in the virtualbox network:

```
$ su
$ vi /etc/init.sh

    #add inside the file
    ifconfig eth0 192.168.56.10 netmask 255.255.255.0 up
    route add default gw 192.168.56.1 dev eth0
    ndc resolver setifdns eth0 8.8.8.8 8.8.4.4
    ndc resolver setdefaultif eth0

$ reboot
```

Android x86 on Virtualbox:



Virtual Networking

You need to decide how to make your virtual machine access the Internet or your local network.

In previous releases, Cuckoo used shared folders to exchange data between the host and guests. From release 0.4 on, it uses a custom agent that communicates over the network with a simple XMLRPC protocol.

To make it work properly, configure your machine's network to allow the host and the guest to communicate. To make sue the virtual netsork was set up properly, test the network access by pinging a guest. Use only static IP addresses for your guest, as Cuckoo no longer supports DHCP. Using it will break your setup.

This stage depends heavily on your requirements and the characteristics of your virtualization software.

> **Warning:** Virtual networking errors! Connectivity between host and guest must be ensured, as virtual networking is a vital component for Cuckoo. sure to get connectivity between host and guest. Most of the issues reported by users are related to an incorrect networking setup. If you aren't sure, check your virtualization software documentation and test connectivity with ping and telnet.

The recommended setup uses a host-only networking layout with proper forwarding and filtering configuration done with `iptables` on the host.

For example, using VirtualBox, you can enable Internet access to the virtual machines using the following `iptables` rules (assuming that eth0 is your outgoing interface, vboxnet0 is your virtual interface and 192.168.56.0/24 is your subnet address):

```
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
```

Add IP forwarding:

```
sysctl -w net.ipv4.ip_forward=1
```

Requirements

To make Cuckoo run properly in your virtualized Linux system, you must install some required software and libraries.

Install Android Debug Bridge

There is no need to download all the SDK to the cuckoo host. Only adb is required.

> $ sudo add-apt-repository ppa:nilarimogard/webupd8 $ sudo apt-get update $ sudo apt-get install android-tools-adb

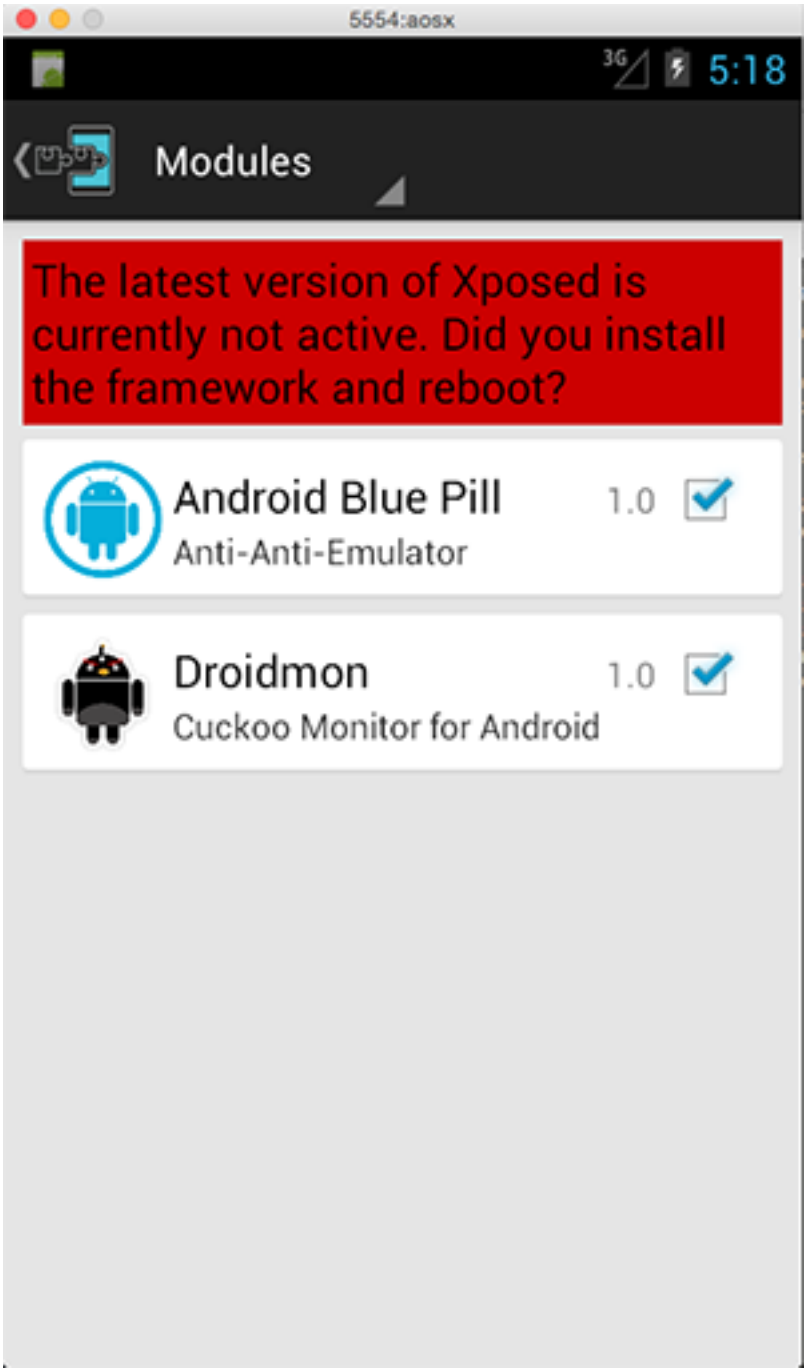
Prepare Android Virtual Machine for Analysis

Start the Android virtual machine after *configuring the network.*:

```
$ adb root
$ adb connect 192.168.56.10
```

Run the script in `utils/android_emulator_creator/create_guest_device.sh`

- Press settings->security->screenlock->none
- Press settings->Display->sleep->Never Timeout
- Press settings->security->checkout verify apps
- Press settings->security->check Unknown sources
- Start Generate contacts app
- **Start Supersuser app -> automatic response -> allow** -> notification None
- Start xposedinstaller app
- In Modules check both packeages `Droidmon` , `Android Blue Pill`



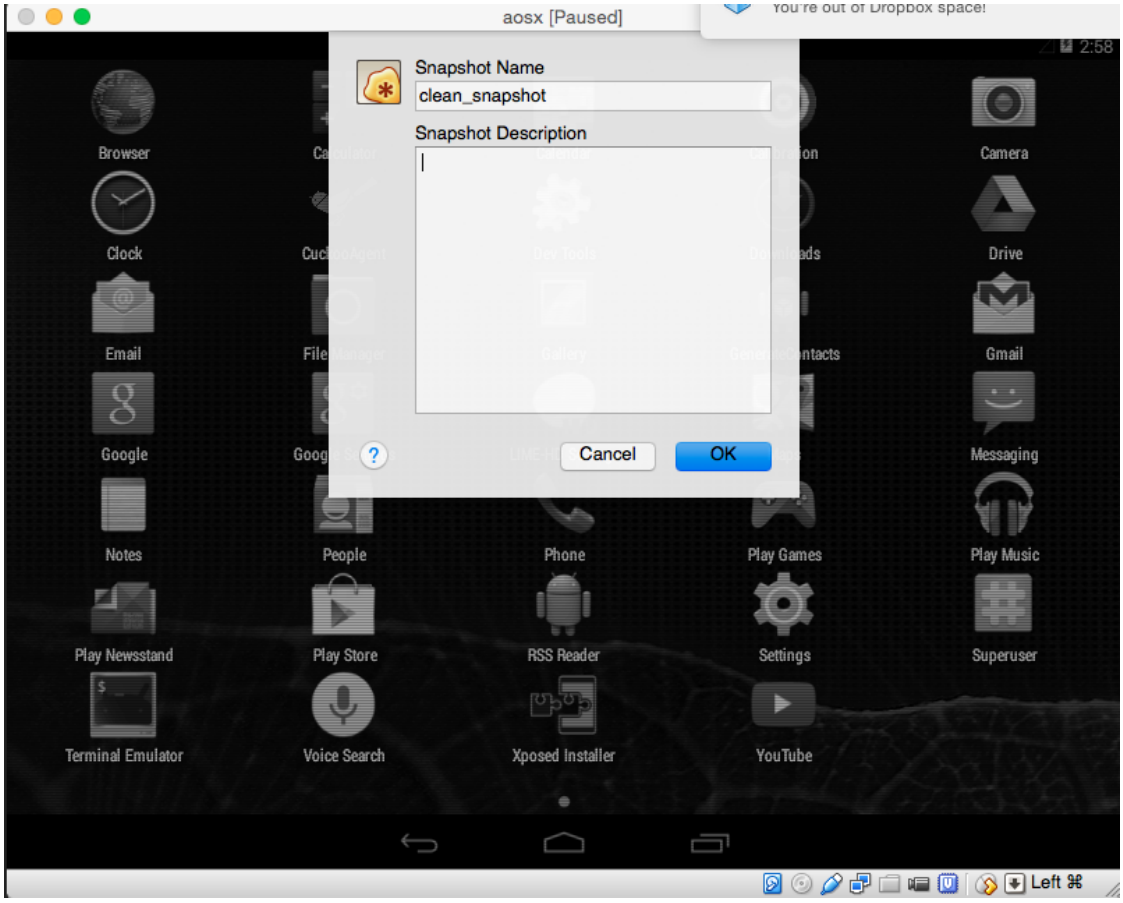
- Press framework -> install -> cancel-> soft reboot

Start cuckooAgent app. Press esc and take snapshot.

Saving the Virtual Machine

You are now ready to save the virtual machine to a snapshot state.

The `CuckooAgent.apk` app must be running in the background and the `Android Virtual Machine` is fully configured.



You can now proceed saving the machine. The method obviously depends on your virtualization software.

If you follow all of the following steps properly, your virtual machine will be ready for Cuckoo use.

VirtualBox

In VirtualBox, you can take the snapshot from the graphical user interface or from the command line:

```
$ VBoxManage snapshot "<Name of VM>" take "<Name of snapshot>" --pause
```

After the snapshot creation is completed, you can power off the machine and restore it:

```
$ VBoxManage controlvm "<Name of VM>" poweroff
$ VBoxManage snapshot "<Name of VM>" restorecurrent
```

VMware Workstation

In VMware Workstation, you can take the snapshot from the graphical user interface or from the command line:

```
$ vmrun snapshot "/your/disk/image/path/wmware_image_name.vmx" your_snapshot_name
```

Your_snapshot_name is the name you choose for the snapshot. Power off the machine from the GUI or from the command line:

```
$ vmrun stop "/your/disk/image/path/wmware_image_name.vmx" hard
```

Cloning the Virtual Machine

If you want to use more than one virtual machine, there's no need to repeat all the steps done so far: you can clone it. This way, you'll have a copy of the original virtualized Windows with all of the requirements already installed.

However, the clone will also contain all the settings of the original virtual machine. Repeat the steps explained in *Network Configuration*, and *Saving the Virtual Machine* for this new machine.