Android Malware Analysis Lab

Dilver Huertas Guerrero

***Abstract*—In this report is show how to configure a malware analysis lab for Android using a virtual machine, that could be deployed over QEMU or VirtualBox. The tool Burp Suite to establish a proxy as a certified authority and Frida to do pentesting. The steps to made the set up of the lab are explained and the code used is presented.**

***Index Terms*—Android, malware, pentesting.**

I. Introduction

THE testing of applications for mobile devices is an area in development. Each day new ways to fight against malware are needed because the attackers are always improving their techniques. In order to do that testing different approaches can be applied, one of the most cost effective is to set up a virtual lab that allows us to test the applications in a controlled environment, without the problem of suffering damages in devices or send important information to the attackers.

In the following sections is show how to set up the lab, including links to the sites to download the tools required, the explanation about the steps realized and the details that have to be take into account to succeed in the process.

II. Setting up the lab

A. Setting up QEMU and the network TAP

In order to install android in a virtual machine it's needed a software of virtualization. In this case QEMU is choose, according to their site: "QEMU is a generic and open source machine emulator and virtualizer", it's available at https://www.qemu.org/. To install QEMU you have to clone the repository from https://github.com/qemu/qemu.git and build it from source code. In this case the host system it's Lubuntu 16.04, but the same process could be done for other linux distributions, windows or macOS.

A script to bring a network terminal access point (TAP) device for QEMU up is needed. Create the file at */etc/qemu-ifup* and add the code shown in figure 1, you must give executable permission to the file. A network TAP: "Denotes a system that monitors events on a local network in order to aid administrators (or attackers) in analyzing the network"[1]. In order to set up the network TAP the packages uml-utilities and bridge-utils are needed to get tunctl and brctl commands. The tunctl command: "allows the host sysadmin to preconfigure a TUN/TAP network interface for use by a particular user. That user may open and use the network/write side of the interface, but may not change any aspects of the host side of the interface"[1]. The brctl command: "is used to set up, maintain, and inspect the ethernet bridge configuration in the linux kernel"[2]. After that, the instructions shown in figure 2 must be executed.

```
#! /bin/sh
# Script to bring a network (tap) device for qemu up. The idea
# in order to be able to find brctl
PATH=$PATH:/sbin:/usr/sbin
ip=$(which ip)

if [ -n "$ip" ]; then
ip link set "$1" up
else
brctl=$(which brctl)
if [ ! "$ip" -o ! "$brctl" ]; then
echo "W: $0: not doing any bridge processing: neither ip nor b
exit 0
fi
ifconfig "$1" 0.0.0.0 up
fi

switch=$(ip route ls |
awk '/^default / {
for(i=0;i<NF;i++) { if ($i == "dev") { print $(i+1); next; } }
}'
)

switch=br0

# only add the interface to default-route bridge if we have su
for br in $switch; do
if [ -d /sys/class/net/$br/bridge/. ]; then
if [ -n "$ip" ]; then
ip link set "$1" master "$br"
else
brctl addif $br "$1"
fi
exit # exit with status of the previous command
fi
done

echo "W: $0: no bridge for guest interface found" >&2
```

Fig. 1. Script for the network TAP

```
:~$ sudo tunctl -t tap0 -u dilver

:~$ sudo ifconfig tap0 up
:~$ sudo brctl addbr br0
:~$ sudo brctl setfd br0 0
:~$ sudo ifconfig br0 10.0.2.2 netmask 255.255.255.0 broadcast 10.0.2.255 up
:~$ sudo brctl addif br0 tap0
:~$ sudo ifconfig tap0 0.0.0.0
:~$ sudo sysctl net.ipv4.ip_forward=1

:~$ sudo iptables --table nat -A POSTROUTING --out-interface wlan0 -j MASQUERADE
```

Fig. 2. Instructions to set up the network TAP

B. Installation of Android in QEMU

Download the file *android-x86_64-8.1-r2.iso* from https://www.android-x86.org/. According to their site: "Android-x86 is a project to port Android to x86 platform, formerly known as "patch hosting for android x86 support". We create our code base to provide support on different x86 platforms, and set up a git server to host it".

After that, in a folder create the disk image for the Android

Dilver Huertas Guerrero, Department of Systems and Industrial Engineering, Universidad Nacional de Colombia, Bogotá, Colombia e-mail: djhuertasg@unal.edu.co

[1] https://linux.die.net/man/8/tunctl
[2] https://linux.die.net/man/8/brctl

storage with the code *qemu-img create -f qcow2 disk.img 10G*. In the same folder must be the android iso to proceed with the installation.In order to run the installer is use the following script *qemu-system-x86_64 -enable-kvm -boot d -cpu host -m 2048 -hda disk.img -cdrom android-x86_64-8.1-r2.iso -net nic -net tap*.
When installing use the next configuration to set up the network:

- SSID: VirtWifi
- Static IP: 10.0.2.12
- Gateway: 10.0.2.2
- DNS: 8.8.8.8

C. Installation of Android with VirtualBox

Some problems could be encountered using QEMU, the alternative is VirtualBox. To install Android you create a virtual machine and select type as Linux and version as Linux 2.6 / 3.x / 4.x 64-bit, a RAM of 8 Gb and a new virtual hard disk of 8 GB. It's important to install android manually and create two partitions, one for the boot and the other for the system. Is a must to install the /system directory as read-write to have the possibility to use ADB (Android Debug Bridge). After installation is needed to enable developer options in Android and enable USB Debugging. In VirtualBox is needed to go to network settings and enable port forwarding to localhost:5555 in order to connect through ADB using *# adb connect localhost:5555*.

D. Installing a custom CA into the Android system

To install a custom CA (Certificate Authority) into the Android system cacert directory, in order to intercept the outgoing/incoming HTTPS traffic with Burp Suite is needed to follow instructions show in figure 3. After that the Android system can be loaded as sudo with the script *qemu-system-x86_64 -enable-kvm -boot c -cpu host -m 2048 -hda disk.img -cdrom android-x86_64-8.1-r2.iso -net nic -net tap*.
If the Android system was installed with VirtualBox you should create the CA as shown in figure 3 and install it through ADB using the code shown in figure 4.

E. Burp Suite SSL interception

Download from https://portswigger.net/burp/communitydownload the software Burp Suite Community Edition v2.1. It's offer free for researchers and hobbyists with a series of essential manual tools. After installing is needed to import the custom generated SSL certificates into the Burp Suite as shown in figure 5. After that is needed to setup Burp Suite to listen on the br0 interface @ 10.0.2.2, as shown in figure 6.

F. Setting up SSL pinning with FRIDA

According to their site available at https://www.frida.re/, FRIDA is: " A dynamic code instrumentation toolkit. It lets you inject snippets of JavaScript or your own library into native apps on Windows, macOS, GNU/Linux, iOS, Android, and QNX. Also provides you with some simple tools built on top of the Frida API".
In order to install Frida pip can be use with the code pipinstallfrida-tools. After that is needed to install the frida server in the virtualized system, the version of frida should be the same in the host and in the virtual machine. From https://github.com/frida/frida/releases get the package *frida-server-12.6.11-android-x86_64.xz*, extract and copy to the virtual machine using ADB with the code *adb push ./frida-server-12.6.11-android-x86_64 /data/local/tmp/*. In the ADB shell as

```
#Mounting the QEMU qcow2 image from the host Linux
#run as sudo
apt-get install libguestfs-tools
cd $ANDROID-QEMU
mkdir img
guestmount -a disk.img -m /dev/sda1 img/
#Setting up the system.sfs file
cp system.sfs ../..   #copying the system.sfs away
cd ../..
umount img
mkdir SYS
mv system.sfs SYS
cd SYS
unsquashfs system.sfs #extracting the system.sfs
#Creating a new mount directory
#run without sudo
mkdir img
mount -o loop system.img img
#Creating a custom certificate via OpenSSL
cd $CERTIFICATE-LOCATION
openssl req -x509 -days 730 -nodes -newkey rsa:2048 -outform der -keyout
openssl rsa -in server.key -inform pem -out server.key.der -outform der
openssl pkcs8 -topk8 -in server.key.der -inform der -out server.key.pkcs
openssl x509 -inform der -in ca.der -out ca.pem
openssl x509 -inform PEM -subject_hash_old -in ca.pem | head -1
cp ca.pem abcdefg1.0
openssl x509 -inform PEM -text -in ca.pem -out /dev/null>> abcdefg1.0
#Move to the extracted image on Android
#run as sudo
mv abcdefg1.0 $ANDROID-QEMU/SYS/squashfs-root/img/etc/security/cacerts
#Pack all these things back together
cd $ANDROID-QEMU/SYS/squashfs.root/
umount img
rm -rf img
cd ..
mkdir BACKUP
mv system.sfs BACKUP/
mksquashfs squashfs-root system.sfs -b 131072
#Mount the qemu2 image again
cd $ANDROID-QEMU/
guestmount -a disk.img -m /dev/sda1 img/
cd img/android-8.1-r2
rm system.sfs              #remove the original system.sfs
cp ../../SYS/system.sfs .  #copy over the new modified one
cd ../..
umount img
```

Fig. 3. Instructions to install the custom CA

```
#Convert the public key
cp ca.pem a58355c2.0
openssl x509 -inform PEM -text -in ca.pem -out /dev/null>> a58355c2.0
#Copy the cert to the phone
adb push a58355c2.0 /data/local/tmp
adb shell
#In the adb shell
su
mount -o rw,remount /system
mv /data/local/tmp/a58355c2.0 /system/etc/security/cacerts/
chown root:root /system/etc/security/cacerts/a58355c2.0
chmod 644 /system/etc/security/cacerts/a58355c2.0
reboot
```

Fig. 4. Instructions to install the custom CA through ADB

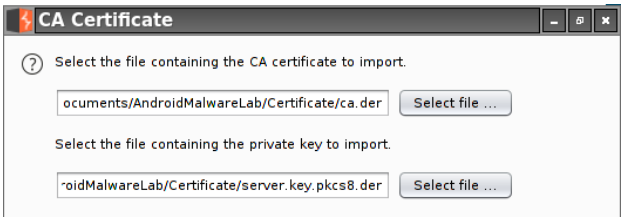

Fig. 5. Importing the custom SSL certificates

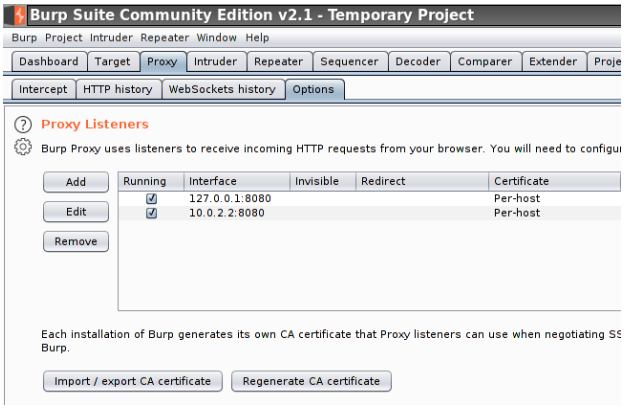

Fig. 6. Setting up the listener

root execute the code *su 0 setenforce 0* to disable SELinux and *./frida-server-12.6.11-android-x86_64 &*. In the localhost using the code *frida-ps -U* the different applications that are running in the virtual machine are shown.
In order to start sniffing the application connections, the script propose by Mattia Vinci[3] shown in figure 7 could be use.

```
/*
   Universal Android SSL Pinning Bypass
   by Mattia Vinci and Maurizio Agazzini
   $ frida -U -f org.package.name -l universal-ssl-check-bypass.js --no-pause
    https://techblog.mediaservice.net/2018/11/universal-android-ssl-check-bypass-2/
*/
Java.perform(function() {
    var array_list = Java.use("java.util.ArrayList");
    var ApiClient = Java.use('com.android.org.conscrypt.TrustManagerImpl');
    ApiClient.checkTrustedRecursive.implementation = function(a1, a2, a3, a4, a5, a6) {
        // console.log('Bypassing SSL Pinning');
        var k = array_list.$new();
        return k;
    }
}, 0);
```

Fig. 7. Script for SSL pinning bypass

III. Testing the lab

After all the tools are set up, it's needed to run the script shown in figure 2 before to start the virtual machine. After that it's needed to establish the proxy using Burp Suite and identify and application that is going to be tested with Frida. Using the code *frida -U -l frida-ssl-2.js –no-paus -f com.example.application* is possible to test any app in the virtual machine. In the figure 8 is shown the Burp Suite working as a proxy for the application Google Chrome and in the figure 9 Frida is set up to make pentesting to the same app.

IV. Conclusion and Further Work

Between the two different tools used to virtualized android, QEMU could be more resource friendly and VirtualBox is more stable. In this case 2Gb of RAM are enough to run the virtual machine, so it's recommended to use VirtualBox because it's easier to set up and no errors were encountered after the installation process was finalized.
Burp Suite is a powerful tool, in this case was use only to set up a proxy network, other tools like Squid[4] could be use in

[3]https://techblog.mediaservice.net/2018/11/universal-android-ssl-pinning-bypass-2/
[4]http://www.squid-cache.org/

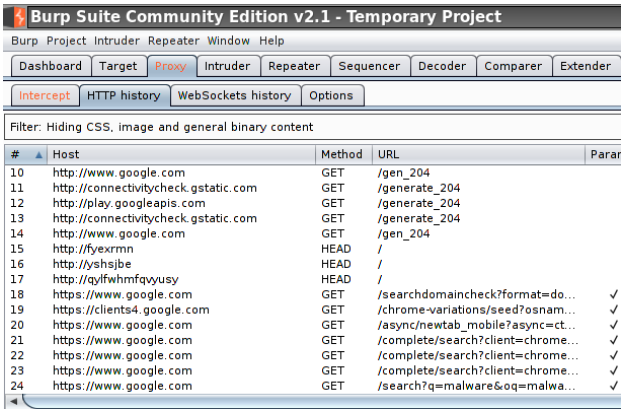

Fig. 8. Burp Suite as proxy

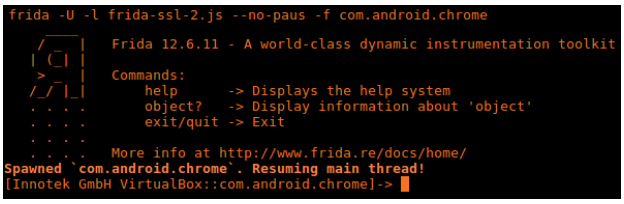

Fig. 9. Frida set up for pentesting

order to save resources and save the state of the configuration, because the community version don't allow to save it.
The further work is to test the lab with actual malware and see how it interacts. Also how Frida could be use to explore that type of software.

References

[1] Abdullah A. Mohammed, Dia M. Ali, *Network Tapping System Based on Customized Embedded Linux: Design and Implementation*, International Journal of Networks and Communications, Scientific & Academic Publishing SAP, accepted Vol 5 (5), 2015.