UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Práctica 3

Análisis estático de apps Android

Tabla de contenido

1. Objetivos

El análisis de aplicaciones móviles resulta fundamental para determinar la existencia de vulnerabilidades en las mismas. Dicho análisis se puede realizar tanto de manera estática como dinámica.

En general, el análisis estático presenta frente al dinámico la ventaja de que se hace sin ejecutar el código. Como no necesita de esa ejecución, el análisis estático permite detectar errores en una fase muy temprana de la escritura de aplicaciones. Así se ahorra mucho tiempo en fases posteriores del desarrollo. Como ya hemos visto, es posible conocer muchas cosas de una app Android simplemente analizando su Manifiesto y su código, tanto de alto (Java) como de bajo (Smali) nivel. Dicho conocimiento puede ser utilizado para profundizar en el comportamiento de la aplicación más tarde cuando esta se ejecute o simplemente para cambiar/ajustar la implementación existente y reensamblar la aplicación para su ejecución. En cambio, el problema más grave que presenta el análisis estático es que puede conllevar la detección de vulnerabilidades que en la práctica no lo sean (falsos positivos), y cuya falsedad solo se pueda confirmar mediante la ejecución del código, lo que nos lleva a la problemática del análisis dinámico de aplicaciones Android.

El análisis dinámico de código se realiza mientras el código se está ejecutando. Es más lento y necesita un proceso completo y algo más complejo de testeo. Sin embargo, permite encontrar muchos errores que quedan ocultos al análisis estático, así como confirmar o desmentir los ya detectados en las revisiones previas del código de la aplicación.

Para cubrir las necesidades de ambos tipos de análisis, el conjunto de herramientas a utilizar es amplio y, en ocasiones, tenerlas todas disponibles y funcionando sobre un mismo sistema operativo resulta imposible. Las máquinas virtuales ofrecen una solución muy interesante a esta situación. De hecho, son varias las máquinas virtuales específicamente desarrolladas con el fin de ofrecer a sus usuarios plataformas de ejecución ya preparadas y con todas las herramientas necesarias ya instaladas para ser utilizadas. No es raro, por tanto, encontrar máquinas virtuales específicamente pensadas para dar soporte al test de penetración de sistemas, al análisis forense de sistemas y aplicaciones, o al hacking ético. Dada la particularidad e importancia de ciertas plataformas, también existen máquinas virtuales específicamente desarrolladas para el análisis de las aplicaciones ejecutables sobre las plataformas. Esto es lo que ocurre con las aplicaciones Android, para cuyo análisis existen máquinas virtuales como Santoku o AndroL4b, que además del utillaje proveen a sus usuarios de varios casos de uso para experimentar y aprender sobre la ciberseguridad móvil.

A todas estas ventajas hay que añadir la "protección" que ofrecen los entornos virtualizados frente a los ataques que pueden llegar a perpetrar las aplicaciones y sistemas bajo análisis. En efecto, las máquinas virtuales definen regiones de confinamiento que impiden la propagación de problemas y minimizan los riesgos inherentes al trabajo con aplicaciones malintencionadas, como el malware o el ransomware.

En esta práctica vamos a poner a punto un laboratorio para poder trabajar en el análisis estático y dinámico de aplicaciones Android. El laboratorio incluirá una máquina virtual Kali, y dos tipos diferentes de emuladores Android, los AVDs (acrónimo de *Android Virtual Device*), con los que ya hemos trabajado, y los que proporciona Genymotion. El objetivo es el de comenzar a diversificar con nuestro entorno de trabajo y crear un laboratorio en el que distintas máquinas virtuales puedan coexistir e interactuar.

2. Componentes con los que trabajaremos

Para comenzar vamos a descargar todo lo necesario para crear nuestro laboratorio, a continuación, procederemos a configurarlo convenientemente. Para las pruebas, utilizaremos un caso de uso muy conocido, una aplicación bancaria vulnerable, denominada *InsecureBankv2*.

Comenzaremos por instalar la herramienta de virtualización que vamos a utilizar, VirtualBox. A continuación, nos descargaremos las máquinas virtuales que incluiremos en el laboratorio. Seguiremos detallando los emuladores Android de los que nos vamos a servir y finalizaremos configurándolo todo y asegurarnos que funciona correctamente gracias al caso de estudio InsecureBankv2.

2.1. Máquinas virtuales

Nuestro laboratorio trabajará con VirtualBox como herramienta de virtualización que nos servirá para ejecutar una máquina virtual Kali Linux, la máquina virtual AndroL4b y el emulador de Android de Genymotion.

VirtualBox es una herramienta de virtualización de Oracle que puede descargarse gratuitamente desde https://www.virtualbox.org/wiki/Downloads. La herramienta está disponible tanto para Windows, como para OS X, Linux y Solaris y se distribuye con un pack de extensión que se recomienda instalar también. La información para instalar VirtualBox estás disponible en https://www.virtualbox.org/manual/ch01.html#intro-installing.

La **máquina virtual Kali Linux** es una máquina virtual Debian diseñada específicamente para test de penetración, análisis forense, hacking ético e ingeniería inversa. Aunque no está específicamente diseñada para el trabajo con dispositivos móviles, tiene detrás una gran comunidad que la mantiene y mejora continuamente, lo cual la hace que su distribución esté al día e integre las últimas versiones de las herramientas más utilizadas, aunque nos obligará a instalar muchas de las aplicaciones que necesitaremos para trabajar específicamente con dispositivos móviles. La máquina virtual Kali Linux está disponible para descarga desde https://www.kali.org/downloads, aunque a nosotros nos interesa la imagen de la máquina virtual para VirtualBox, que se ofrece Offesive Security, una empresa americana centrada en la formación en ciberseguridad, a través de su web https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-image-download. Una vez descargada, hay que importar la imagen descargada desde VirtualBox siguiendo los pasos descritos en https://blogs.oracle.com/oswald/importing-a-vdi-in-virtualbox. Las credenciales (usuario y contraseña) de la máquina virtual con "*kali*" y "*kali*", respectivamente.

Cabe señalar que recientemente se ha puesto a disposición de la comunidad Kali la nueva distribución Kali Nethunter (https://www.kali.org/kali-nethunter), un sistema operativo basado en Android que integra un conjunto de herramientas especialmente pensadas para para comprobar la seguridad de los dispositivos móviles, ya que integra paquetes para estudiar las comunicaciones 802.11, Bluetooth, NFC de los dispositivos, así como sus conexiones a puntos de acceso remotos, y permite estudiar ataques relativos a la introducción de información por pantalla táctil o la conexión de los dispositivos vía USB a otros equipos informáticos. Lo interesante es que la distribución ha sido diseñada con su propio store de apps (https://store.nethunter.com) lo que facilita su uso en los dispositivos móviles bajo estudio al poder instalar nuevas apps, así como actualizar las

apps de seguridad ya existentes. A pesar de su interés el proyecto está aún en su más tierna infancia y las versiones disponibles de la distribución se ejecutan en pocos dispositivos y emuladores, con lo que de momento no la integraremos en nuestro laboratorio.

Existen máquinas virtuales Linux especialmente diseñadas como DOJOs para gente interesada en aprender ciberseguridad para dispositivos móviles. Aunque hay muchas, y no vamos a nombrarlas todas, las más relevantes actualmente son Santoku (https://santoku-linux.com), aunque la máquina virtual hace tiempo que no se mantiene ni actualiza, y **AndroL4b** (https://github.com/sh4hin/Androl4b), de más reciente creación y, por tanto, algo más actual. Esta última integra principalmente herramientas para ingeniería inversa, análisis de vulnerabilidades y análisis de malware, así como casos de uso interesantes, como InsecureBankv2 (el referente a la hora de abordar el análisis estático y dinámico de apps Android, y el que vamos a considerar nosotros), Damn Insecure and vulnerable App for Android (DIVA), GoatDroid (diseñada por OWASP) y Sieve (un app de gestión de contraseñas). La máquina virtual está actualizada a Ubuntu Mate 17.04 y está disponible mediante descarga desde Google Drive y Mega. Su interés viene motivado por la cantidad de herramientas para el análisis de aplicaciones móviles que ya trae instaladas, y que en determinadas circunstancias nos pueden resultar de gran utilidad en las prácticas, tanto para análisis estático, como QARK, Drozer y MobSF, como para análisis dinámico, como BurpSuite, Frida y MobSF. Aunque la descarga de esta máquina virtual no es obligatoria, es recomendable, porque nos proporciona un entorno de trabajo en el que las herramientas más relevantes que vamos a utilizar ya están instaladas y preparadas para ser utilizadas.

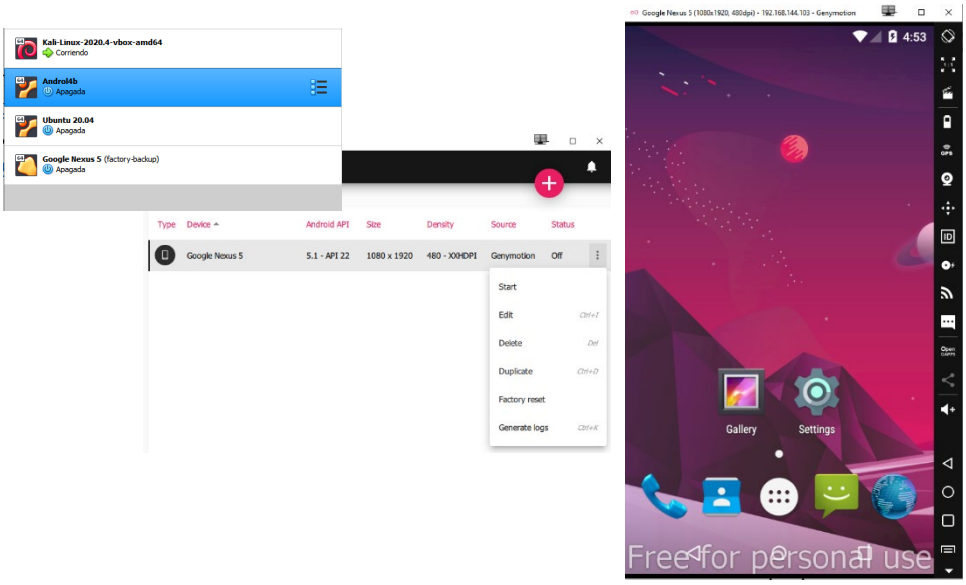
Descarga e instala VirtualBox y su extensión Pack. A continuación, descarga la(s) máquina(s) virtuales que te interesen. Para el análisis estático de apps Android, sólo utilizaremos la máquina virtual Kali.

2.2. Emuladores Android

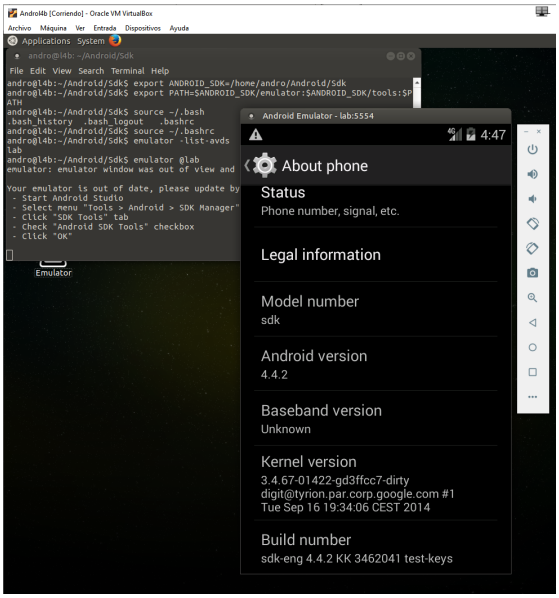
Hasta ahora todo lo que hemos hecho con Android lo hemos hecho sirviéndonos de los AVDs o *Android Virtual Devices*, es decir, los emuladores Android que se pueden crear utilizando el *Android Virtual Device Manager*, que se integra en Android Studio.

Sin embargo, existen alternativas comerciales, como la propuesta por Genymotion. La visión de Genymotion va más allá de la emulación local de dispositivos Android y su modelo de negocio se basa en ofrecer la posibilidad de utilizar dispositivos virtuales Android bajo demanda, monetizando su uso a 0.05$ el minuto, o de poder disponer de máquinas virtualizadas Android en la nube, disponibles a través de los marketplaces de AWS, Azure, GCP y Aliyun, y facturadas a 0.5$ la hora y dispositivo. En su propuesta de escritorio, ofrecen la posibilidad de utilizar emuladores locales cuyo uso comercial exige del pago de una cuota anual, aunque se ofrece la posibilidad de utilizar estos emuladores de manera gratuita para uso personal. Estos emuladores de escritorio utilizan VirtualBox como herramienta de virtualización y la aplicación, llamada *Genymotion Desktop*, para su configuración, creación y lanzamiento a ejecución está disponible en https://www.genymotion.com/desktop. Genymotion Desktop está disponible tanto para Windows, como para Linux y Mac. La información para su instalación está disponible en https://docs.genymotion.com/desktop/latest. Cabe señalar que el uso de los emuladores de Genymotion desde Android Studio exige la instalación de un plugin como se describe en https://docs.genymotion.com/desktop/latest/07_Plugins.html#genymotion-plugin-for-

android-studio. Aunque ya se ha mencionado, para la creación de un emulador es necesario utilizar la aplicación *Genymotion Desktop*. Cuando el emulador se crea, podremos ver que en VirtualBox se ha creado una máquina virtual con el nombre del dispositivo que hayamos creado. Sin embargo, para arrancar dicha máquina virtual se recomienda utilizar Genymotion Desktop tanto para arrancar como para parar la ejecución del emulador. Las siguientes imágenes muestran la entrada creada en VirtualBox para un emulador Google Nexus 5, las opciones que para la gestión de dicho emulador ofrece la aplicación de Genymotion y el emulador en funcionamiento.



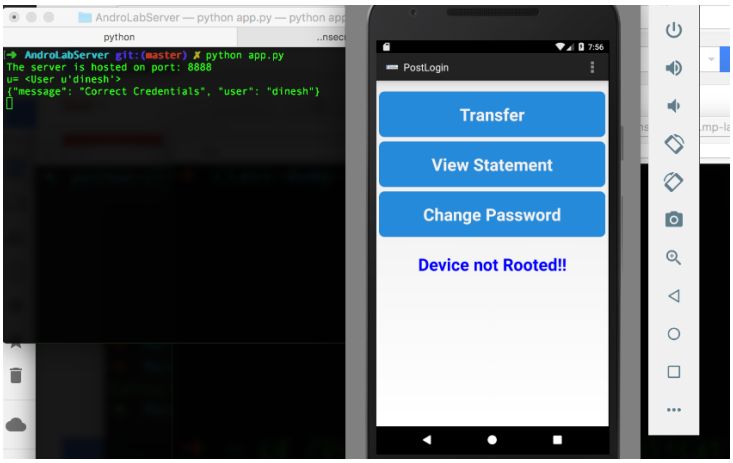
Finalmente, indicar que AndroL4b integra también un emulador que podemos utilizar para nuestras pruebas. La siguiente imagen muestra que el emulador soporta Android 4.4.2 (KitKat API 19), así como las variables de entorno que deberemos inicializar adecuadamente antes de poder lanzar el emulador a ejecución.



3. Caso de estudio: InsecureBank v2

La aplicación *InsecureBankv2* ha sido desarrollada por un grupo de entusiastas de la ciberseguridad con el objetivo de ofrecer un caso de estudio propicio para el análisis de aplicaciones móviles Android y el descubrimiento, explotación y corrección de vulnerabilidades en las mismas. La aplicación consta de 2 partes, una app que se ejecuta en el dispositivo móvil y un servidor (backend) con el que la app realiza transacciones encaminadas a la autenticación de los clientes y la realización de transferencias bancarias.

Tanto el código del cliente como del servidor están disponibles para descarga desde https://github.com/dineshshetty/Android-InsecureBankv2. En esta URL encontraremos tanto el apk del cliente, *InsecureBankv2.apk*, como el código en Python del servidor, en el directorio *AndroLabServer*. Para la ejecución del servidor hay que tener mucho cuidado porque el código está desarrollado con Python 2, y por tanto, no funcionará si utilizamos phyton 3 para su ejecución. En el documento https://github.com/dineshshetty/Android-InsecureBankv2/blob/master/Usage%20Guide.pdf encontraremos una guía para la puesta en marcha del caso de uso utilizando el emulador de Genymotion y ejecutando el backend en el host que ejecuta dicho emulador. La siguiente imagen muestra el laboratorio en funcionamiento.



Cabe señalar que el uso del emulador de Genymotion no es obligatorio para ejecutar este caso de uso, puesto que podríamos emplear los AVDs que ya tenemos y que funcionan teniendo simplemente instalado la SDK de Android que viene con Android Studio. Sin embargo, el objetivo de esta práctica es, como ya hemos mencionado en la introducción, crear un laboratorio con el que cubramos las distintas configuraciones que pueden ser necesarias para el análisis de una aplicación Android y que tienen cuenta de dónde se ejecuta el cliente y el servidor, que pueden hacerlo localmente o en máquinas virtuales.

Volviendo a nuestro caso de estudio, y para concluir, señalaremos que el interés por este ejemplo en concreto, de los muchos disponibles, viene motivado por la gran lista de vulnerabilidades que permite cubrir. Más detalles a este respecto pueden encontrarse en el repositorio https://github.com/dineshshetty/Android-InsecureBankv2.

4. Configuraciones posibles del entorno de trabajo

El gran problema cuando trabajamos con máquinas virtuales es la creación de una red de comunicación entre ellas que permita que las aplicaciones que éstas ejecutan interactúen

entre sí. Los casos a considerar son básicamente dos. En primer lugar, que dos emuladores Android interactúen entre sí y, en segundo lugar, que un cliente que se ejecuta en un emulador Android interactúe con un servidor que corre en una máquina virtual. Veamos cómo gestionar cada caso por separado.

Para comprender como interactúan entre sí dos emuladores hay que entender que, en el caso de los emuladores de Genymotion, que corren sobre VirtualBox, cada emulador dispondrá de una IP propia, con lo que los servicios que ejecuten serán fácilmente direccionables a través de la IP de su emulador y el puerto por el que den servicio.

Esto no pasa en el caso de los AVDs que instanciamos a través de las herramientas que nos proporciona Android Studio. La siguiente tabla ha sido extraída de la web para el desarrollo con Android: https://developer.android.com/studio/run/emulator-networking. En ella se detalla que cada AVD se ejecuta sobre su propia red privada virtual en un rango de direccionamiento 10.0.2.xx tal y como se muestra a continuación:

Network Address	Description
10.0.2.1	Router/gateway address
10.0.2.2	Special alias to your host loopback interface (i.e., 127.0.0.1 on your development machine)
10.0.2.3	First DNS server
10.0.2.4 / 10.0.2.5 / 10.0.2.6	Optional second, third and fourth DNS server (if any)
10.0.2.15	The emulated device network/ethernet interface
127.0.0.1	The emulated device loopback interface

Esta información debe ser tenida en cuenta a la hora de comunicar dos AVDs entre sí o bien un emulador AVD con otro externo, por ejemplo el de Genymotion.

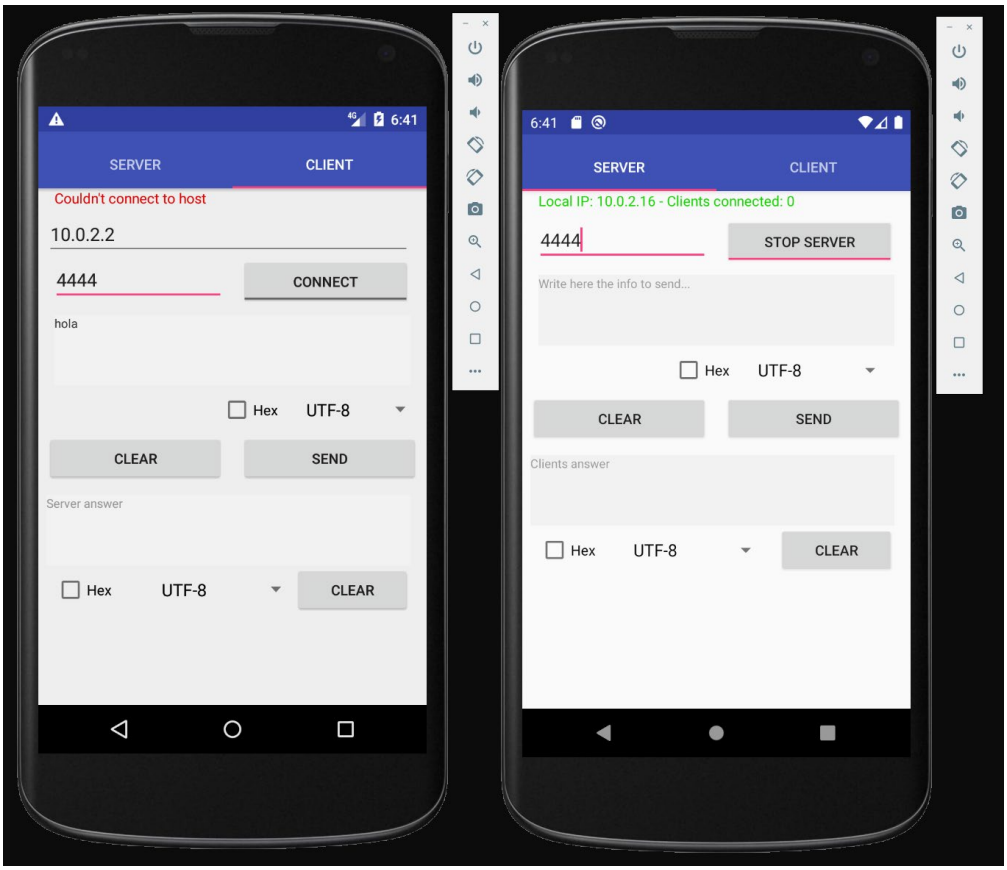
4.1. Comunicación entre 2 AVDs

Consideremos primero que tenemos la necesidad de que 2 AVDs comuniquen entre sí. Tomemos como ejemplo la aplicación *Simple TCP Socket Tester* que nos va a permitir probar si la comunicación entre los emuladores bajo estudio funciona o no. Esta aplicación está disponible para descarga tanto en *APK pure* como en *Google Play* a través de los enlaces: https://apkpure.com/simple-tcp-socket-tester/com.simplesockettester y https://play.google.com/store/apps/details?id=com.simplesockettester, respectivamente.

Arranquemos os dos AVDs distintos y ejecutemos en ellos la aplicación considerada, configurándola en uno como cliente y en otro como servidor en el puerto 4444. A la hora de realizar las pruebas nos aseguramos de que ambos emuladores no tienen configurada ningún reenvío de puerto, así sabemos que partimos de una configuración limpia. Para ello ejecutamos las siguientes órdenes mediante las cuales determinamos los emuladores en ejecución, limpiamos para cada uno todos los reenvíos de puerto que tengan, luego solicitamos que nos los liste para verificar que no hay ninguno, y finalmente solicitamos la eliminación de todo reenvío al puerto que vamos a utilizar, el puerto 4444. Obviamente, esta última orden fallará, puesto que todos los reenvíos han sido ya eliminados. La siguiente captura muestra la secuencia de órdenes a desplegar (que será la misma trabajéis con el operativo que trabajéis) para llevar a cabo las comprobaciones descritas:

```
Símbolo del sistema

C:\Users\jucar\AppData\Local\Android\Sdk\emulator>adb devices
List of devices attached
emulator-5554   device
emulator-5556   device


C:\Users\jucar\AppData\Local\Android\Sdk\emulator>adb -s emulator-5554 forward --remove-all

C:\Users\jucar\AppData\Local\Android\Sdk\emulator>adb -s emulator-5556 forward --remove-all

C:\Users\jucar\AppData\Local\Android\Sdk\emulator>adb -s emulator-5556 forward --list


C:\Users\jucar\AppData\Local\Android\Sdk\emulator>adb -s emulator-5554 forward --list


C:\Users\jucar\AppData\Local\Android\Sdk\emulator>adb -s emulator-5556 forward --remove tcp:4444
adb.exe: error: listener 'tcp:4444' not found

C:\Users\jucar\AppData\Local\Android\Sdk\emulator>adb -s emulator-5554 forward --remove tcp:4444
adb.exe: error: listener 'tcp:4444' not found

C:\Users\jucar\AppData\Local\Android\Sdk\emulator>
```

A continuación, si intentamos que ambos emuladores comuniquen veremos que la comunicación no funciona. Tal y como muestra la imagen inferior, el emulador de la izquierda actúa como cliente, mientras que el de la derecha lo hace de servidor. El servidor la escucha en el puerto 4444. El cliente envía un mensaje "*hola*" a dicho puerto utilizando como dirección IP la dirección 10.0.2.2, que recordemos es un alias de 127.0.0.1. Desgraciadamente, el mensaje no llega.



Alguien puede pensar que puesto que la IP que el servidor declara es la 10.0.2.16 entonces es esa la IP que debería ser utilizada por el cliente, pero si lo probamos, veremos que la comunicación tampoco funciona. Hay que entender que cada emulador está en una red privada virtual diferente, y que su única manera de enviar información a otro equipo es a

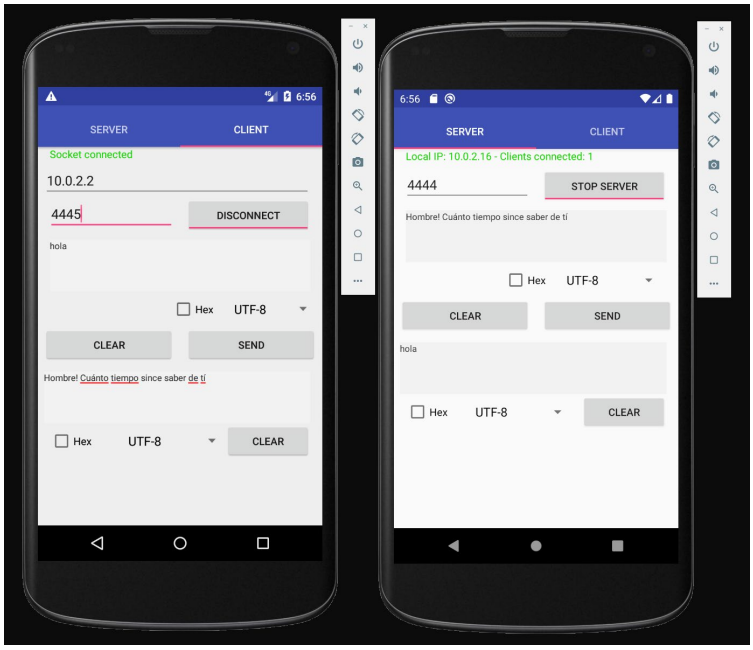
través del equipo que hospeda al emulador, y que se encuentra accesible en la dirección 10.0.2.2 que es un alias de *localhost*, es decir, del local loop-back de la máquina (127.0.0.1).

Por tanto, para solucionar esta situación, es necesario realizar un reenvío del puerto de escucha que utiliza el servidor. Asumamos que el cliente envía la información al puerto 4445 de su localhost y que éste reenvía dicha información al puerto 4444 del emulador del servidor (emulator-5556 en nuestro caso). Esto lo hacemos de la siguiente forma:

```
C:\Users\jucar\AppData\Local\Android\Sdk\emulator>adb -s emulator-5556 forward tcp:4445 tcp:4444

C:\Users\jucar\AppData\Local\Android\Sdk\emulator>adb forward --list
emulator-5556 tcp:4445 tcp:4444
```

Hay que darse cuenta de que estamos asumiendo que la comunicación es por TCP y que el servidor se ejecuta en un emulador con identificador *emulator-5556*. Si se intenta reproducir este escenario, hay que adaptar, obviamente dicho identificador al que tenga el emulador que estemos utilizando, y también, al tipo de comunicación, TCP o UDP, que imponga la aplicación que usemos.

Y el resultado es este:



Obviamente, este escenario funciona porque utilizamos 2 AVDs. Sin embargo, la cosa cambia cuando usamos 1 AVD y 1 emulador Genymotion. El siguiente punto aborda este otro escenario.

4.2. Comunicación entre 1 AVDs y 1 emulador Genymotion

En este caso, el emulador AVD debe hablar con una máquina remota, el emulador Genymotion, que se ejecuta sobre VirtualBox, y que tiene una IP propia.

En el ejemplo que vamos a describir, vamos a considerar el emulador de Genymotion es el que ejecuta el servidor. Por tanto, la máquina sobre la que se ejecuta el cliente (emulador AVD) debe redirigir la peticiones que éste emita hacia el servidor (emulador Genymotion). Esta redirección de puertos depende del sistema operativo de la máquina, ya que recordemos que el emulador Genymotion es visto por la máquina en la que

trabajamos como una máquina remota con IP propia. Así que, el problema que abordamos es un problema de redirección de puertos que se resolverá de manera distinta en función de si trabajamos sobre Linux o lo hacemos sobre Windows.

En el caso de utilizar Linux, si queremos que toda petición de una máquina al puerto 8888 ser remita al puerto 888 de otra con IP 192.168.144.104, procederemos de la manera siguiente utilizando las iptables de la máquina origen de los mensajes:

```
andro@l4b: ~
File  Edit  View  Search  Terminal  Tabs  Help
andro@l4b: ~                                    andro@l4b: ~
andro@l4b:~$ sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
andro@l4b:~$ sudo iptables -t nat -A PREROUTING -p tcp --dport 8888 -j DNAT --to-destination 192.168.144.104:8888
andro@l4b:~$ sudo iptables -t nat -A POSTROUTING -j MASQUERADE
andro@l4b:~$
```

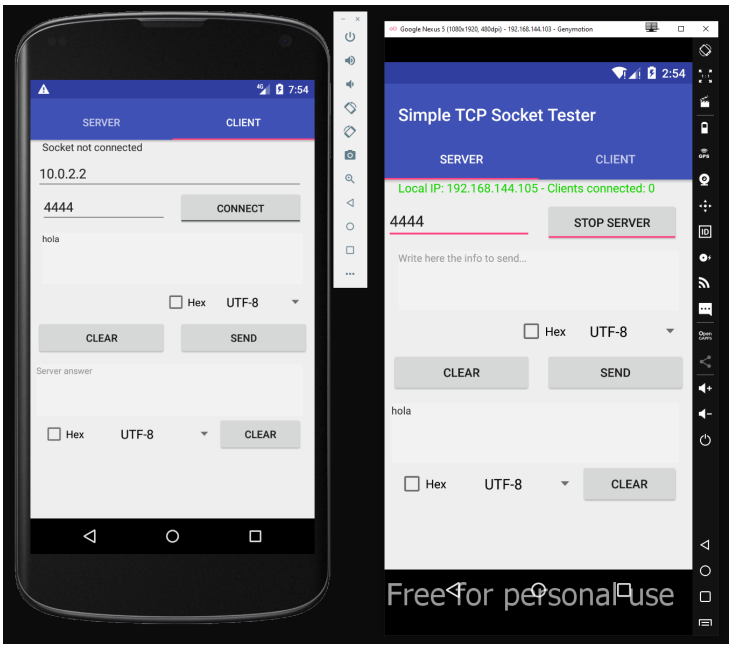
Si trabajamos con Windows haremos lo que debemos hacer es lo propio, pero utilizando la orden *netsh*. Ilustraremos el procedimiento asumiendo una configuración similar a la descrita en la subsección 2.4.1.1, que la que el AVD hará las veces de cliente y que el emulador Genymotion actuará servidor. Aunque ambos emuladores se ejecutarán en una misma máquina física, el emulador de Genymotion estará configurado para encaminamiento por NAT, y en este ejemplo, recibirá la IP 192.168.144.105. Para redireccionar las peticiones que el cliente AVD emita al puerto 4444 de su localhost al puerto homólogo de la máquina 192.168.144.105 abriremos un terminal con permisos de Administrador y ejecutaremos la siguiente orden:

```
Administrador: Símbolo del sistema
Microsoft Windows [Versión 10.0.18363.1379]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\WINDOWS\system32>netsh interface portproxy add v4tov4 listenport=4444 listenaddress=127.0.0.1 connectport=4444 connectaddress=192.168.144.105
```

Comprobamos que la redirección se ha realizado correctamente de la siguiente manera:

```
C:\WINDOWS\system32>netsh interface portproxy show v4tov4

Escuchar en ipv4:       Conectar a ipv4:

Dirección       Puerto      Dirección       Puerto
--------------- ----------  --------------- ----------
127.0.0.1       4444        192.168.144.105 4444
```

Al ejecutar los emuladores vemos que la comunicación funciona:



Y finalmente, para eliminar dicha redirección, ejecutaríamos la orden:

```
C:\WINDOWS\system32>netsh interface portproxy delete v4tov4 listenport=4444 listenaddress=127.0.0.1
```

Puede ocurrir que la comunicación no funcione porque no podamos utilizar el puerto que deseamos al encontrarse éste ocupado por otro proceso. Esto podemos averiguarlo de la manera siguiente:

```
C:\WINDOWS\system32>netstat -ano | findstr 7777
  TCP    127.0.0.1:7777         0.0.0.0:0              LISTENING       5368
```

Y en caso de que sea así bien elegiremos otro puerto o mataremos al proceso que tiene ocupado el puerto que deseamos utilizar.

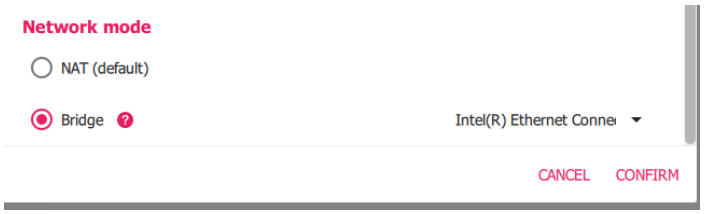
```
C:\WINDOWS\system32>taskkill /F /PID 5368
```

4.3. Comunicación entre un emulador y un back-end virtualizado

La configuración de 2 máquinas virtuales para que éstas puedan comunicar entre sí no es excesivamente difícil. Sin embargo, las máquinas deben poder no sólo comunicar entre sí en la red de área local, sino que deben poder hacerlo también a través de Internet. Típicamente las máquinas virtuales incorporan un único adaptador de red configurado en modo NAT. Pero en nuestro caso necesitaremos que tengan 2 adaptadores. La pregunta es ¿cómo deben configurarse esos adaptadores?

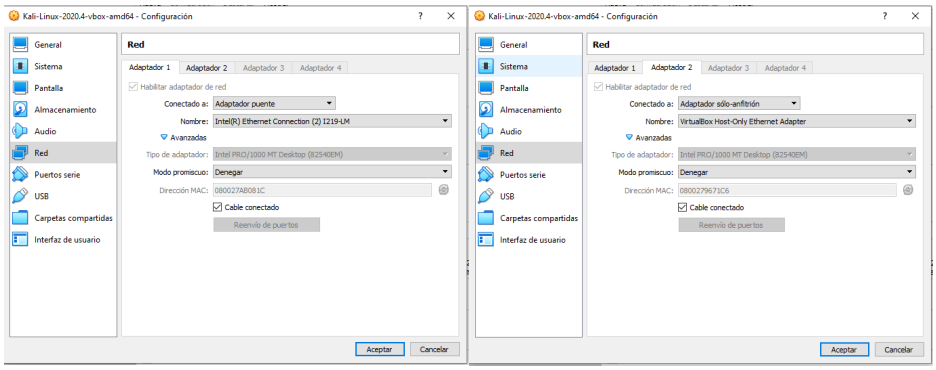
Configuración del emulador de Android de Genymotion

Cuando creemos el emulador de Genymotion nos aseguraremos de que el modo de red sea NAT. Si ya tenemos un emulador creado no hay que eliminarlo y crear otro nuevo. Simplemente editaremos su configuración desde la aplicación de Genymotion y modificaremos su modo de red. Por tanto, es importante que el modo de red sea Bridge tal y como se muestra en la imagen para que el emulador tenga salida a internet.

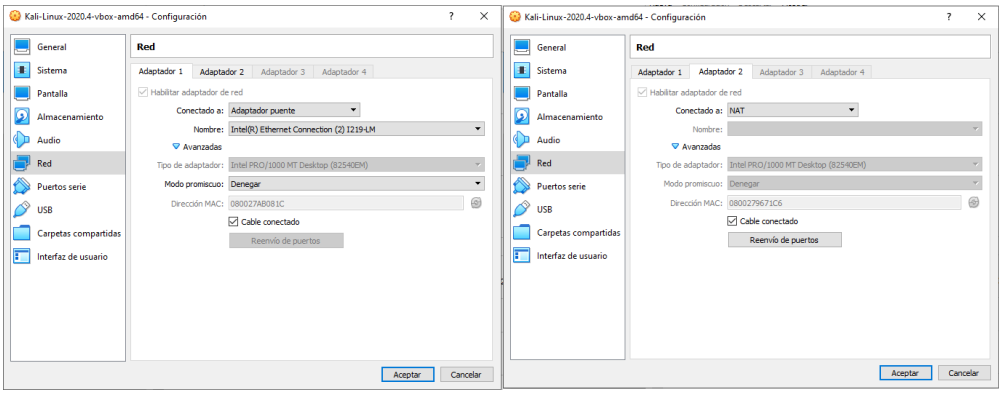


Configuración de los adaptadores de la máquina Kali

Por su parte la máquina Kali Linux en la que correrá el back-end de nuestro caso de estudio, *InsecureBankv2*, deberá configurar sus dos adaptadores de red de la manera siguiente:



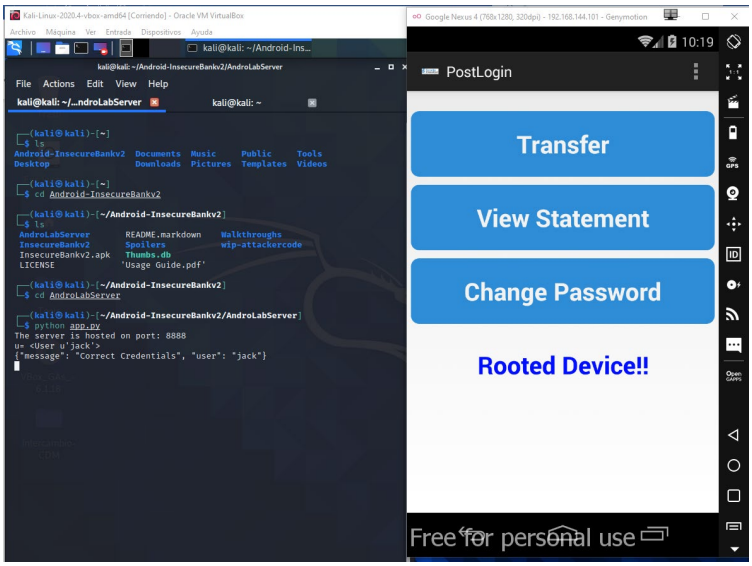
Con esta configuración la máquina virtual Kali no podrá acceder a Internet, pero sí comunicar con el emulador de Genymotion. También es posible configurarla para que tenga acceso a Internet, además de obtener una IP local en nuestra LAN. Para ello adoptaríamos esta otra configuración:



Por supuesto, lo ideal es tener un entorno de trabajo aislado, sobre todo cuando trabajemos con aplicaciones susceptibles de contener algún tipo de malware o ransomware. Sin embargo, también nos puede interesar que nuestro back-end o nuestro emulador acceda a Internet. En cualquier caso, el emulador de nuestro dispositivo deberá dirigirse a la IP de la máquina Kali, para poder comunicarse con ella. Esta IP la obtendremos de la siguiente manera:

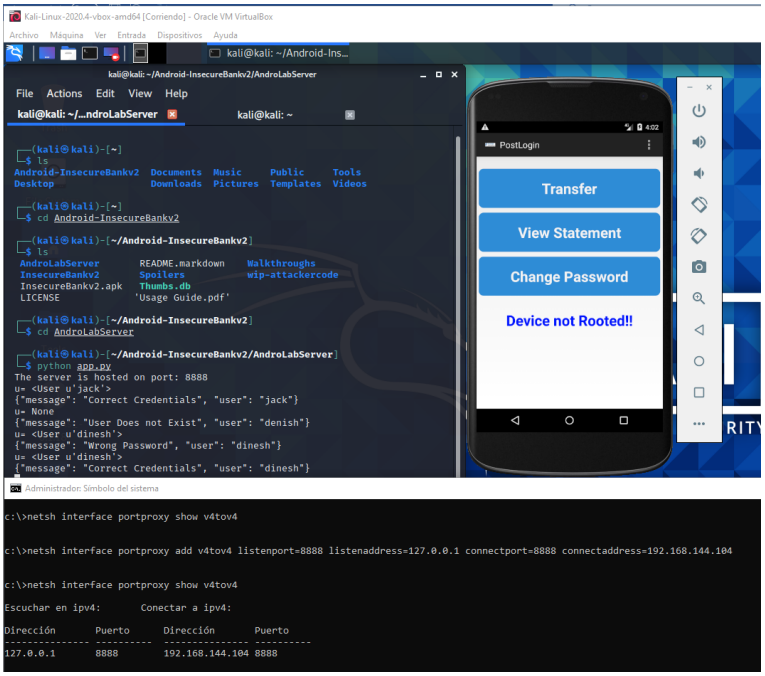
```
┌──(kali㉿kali)-[~]
└─$ ifconfig                                                    2 ×
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 08:00:27:ab:08:1c  txqueuelen 1000  (Ethernet)
        RX packets 5946  bytes 815498 (796.3 KiB)
        RX errors 0  dropped 1  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.144.104  netmask 255.255.255.0  broadcast 192.168.144.
255
        inet6 fe80::a00:27ff:fe96:71c6  prefixlen 64  scopeid 0×20<link>
        ether 08:00:27:96:71:c6  txqueuelen 1000  (Ethernet)
        RX packets 749  bytes 298188 (291.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 58  bytes 6666 (6.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0×10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 13  bytes 676 (676.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 13  bytes 676 (676.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

En la configuración del cliente (accesible a través de la opción *Preferences* del menú que muestra la app en su pantalla inicial) tendremos que sustituir la IP que se propone por defecto (10.0.2.2) por la de la máquina Kali con la que deseamos comunicar. Tras introducir alguna de las credenciales existentes, jack/Jack@123$ o dinesh/Dinesh@123$, obtendremos algo como lo que sigue:



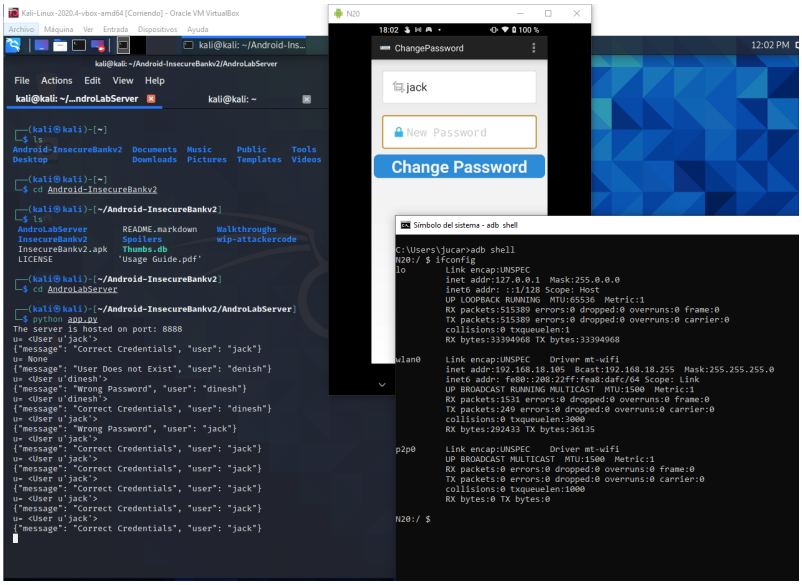
Configuración del emulador AVD

¿Y si el emulador fuera, en lugar del emulador de Genymotion, el emulador AVD? Pues aplicaríamos lo que ya hemos aprendido y redirigiríamos el puerto de comunicación del AVD hacia el servidor, bien mediante iptables o ssh (Linux), bien utilizando netsh (Windows). Si lo hacemos correctamente tendremos que nuestro laboratorio lucirá como sigue:



4.4. Uso de dispositivos reales

Suele ser raro, pero en determinadas circunstancias nos podría interesar trabajar con un dispositivo real, y no con uno emulado. Con todo lo aprendido no debería ser complicado configurar nuestro entorno de trabajo para conseguir algo como lo que se muestra a continuación:



La figura que se adjunta muestra una captura de la pantalla de un dispositivo real que está conectado por USB a la máquina en la que se ejecuta la máquina virtual Kali, aunque podría estarlo a cualquier otra máquina de nuestra LAN. La captura de pantalla se realiza con la aplicación *scrcpy* (ver transparencias para más información) y demuestra que es posible conectarse vía WIFI desde el dispositivo físico, mientras que aprovechamos el hecho de estar conectados por USB a un PC para abrir un Shell con adb sobre el teléfono. En nuestro caso el teléfono está conectado por USB a nuestra máquina, pero podría no estarlo, ya que tal y como vimos en clase, es posible establecer una conexión remota e inalámbrica con un dispositivo físico usando adb.

Con todo lo visto estamos ya en condiciones de afrontar casi cualquier reto que se nos plantee con distintas configuraciones, puesto que hemos creado un entorno de trabajo en el que podemos instanciar y comunicar tanto emuladores, como máquinas virtuales y dispositivos reales. Cabe señalar que todo aquel que trabaje sobre Linux y desee tener una máquina virtual Windows, ésta está disponible para descarga desde la siguiente web de Microsoft: https://developer.microsoft.com/es-es/windows/downloads/virtual-machines. Si visitáis la web podréis constatar que la máquina virtual se suministra como entorno de desarrollo de Windows 10 y está disponible tanto para soluciones de virtualización de VMWare, como de Parallels, VirtualBox e Hyper-V.

5. Cuestiones planteadas: Análisis estático de InsecureBankv2

Se van a proponer varios ejercicios y pistas para resolverlos, pero no se va a imponer ninguna configuración en el entorno de trabajo (elegir de las configuraciones propuestas en el punto anterior de esta memoria, la que mejor se adapte a la solución que cada uno

adopte), ni tampoco se va a exigir la utilización e una herramienta u otra de análisis de las ya vistas en clase (principalmente apktool, adb, aapt, drozer y MobSF).

Con el objetivo de facilitar la corrección de las respuestas resuelve los retos que se plantean utilizando un emulador con una versión de Android 4.4 KitKat (API 19).

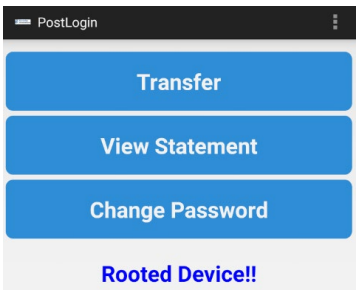
5.1. Ejercicio 1: Análisis estático del apk InsecureBankv2 con MobSF

Instala MobSF y analiza el apk de InsecureBanckv2. Partiendo de la información que proporciona el informe que genera la herramienta, responde a las siguientes preguntas:

- Indica el nombre del paquete de la aplicación.
- ¿Cuántos componentes Android se incluyen en la aplicación? Clasifícalos por tipo e incluye sus nombres, así como si son accesibles o no a componentes externos a la aplicación.
- El informe señala que la actividad *LoginActivity* es la actividad principal de la aplicación, ¿sabrías explicar por qué? Indica en qué se basa MobSF para dicha afirmación.
- Si te fijas se indica que la firma de la aplicación presenta un problema de seguridad, ¿cuál es y en qué consiste?
- ¿Cuántos permisos requiere la aplicación para su ejecución? Enuméralos y señala aquellos que la herramienta clasifica como peligrosos.
- Consulta el análisis que MobSF hace del manifiesto y explica por qué es peligroso que aparezca en el mismo el flag android:allowBackup y a quién afecta dicho flag.
- ¿Se han encontrado Trackers en la app? ¿Cuales?
- El análisis realizado revela que se han encontrado varios “secretos posibles” en el código de la aplicación, ¿cómo podríamos denominar esos secretos si finalmente resultan no serlo?
- Visualiza el informe en pdf que genera la herramienta y busca el nivel de riesgo que se ha asociado al apk analizado. En base al riesgo inferido ¿sería una app que podríamos analizar con tranquilidad o deberíamos adoptar medidas de protección importantes?

5.2. Ejercicio 2: Baipasear la pantalla de Login

Tu objetivo en este ejercicio consiste en evitar la pantalla de login de la app y sin introducir ningún tipo de credenciales llegar a la siguiente pantalla:



Incluye en el informe de tu práctica el proceso (documéntalo como quieras) seguido para realizar la operación solicitada.

Nota: Dependiendo del emulador o dispositivo que utilices el mensaje en azul oscuro cambiará entre "Rooted Device!!" y "Device not Rooted!!"

5.3. Ejercicio 3: Búsqueda de opciones ocultas en la pantalla de Login

El layout de la actividad *LoginActivity* contiene un botón oculto. Observa el código del método *onCreate* de la actividad e idea una manera de hacer que dicho botón aparezca. Incluye en tu memoria una captura de la actividad con el botón activo. Investiga un poco y determina que hace la app cuando se activa dicho botón.

5.4. Ejercicio 4: Comprobación de credenciales y posibles credenciales ocultas

En la actividad *LoginActivity* hay un método, el método *performLogin*(), que se activa cada vez pulsamos el botón de Login. Dicho método delega la verificación de las credenciales remitidas en otra actividad. ¿Cuál es el nombre de dicha actividad? ¿qué mecanismo se utiliza para conseguir su activación y la comunicación de las credenciales?

Una vez esté claro el mecanismo de comunicación de credenciales estudia el método *postData()* de la actividad que acabas de identificar y determina si existe o no alguna credencial que pueda utilizarse y que sea alternativa a las ya conocidas, es decir, jack/Jack@123$ y dinesh/Dinesh@123$. En caso afirmativo, indica cuales son esas credenciales y pruebalas.

5.5. Ejercicio 5: ¿Es el almacenamiento de credenciales seguro o no?

Con el objetivo de facilitar la introducción de credenciales a los usuarios, éstas se almacenan. ¿Podrías decir dónde? ¿Es seguro el almacenamiento realizado? Si no lo es accede a las credenciales y explica cómo pueden obtenerse.

5.6. Ejercicio 6: Análisis del uso de la cache de teclado de Android

Android tiene un diccionario donde son guardadas las palabras introducidas por los usuarios para poder realizar futuras auto-correcciones. Este diccionario está disponible para todas las apps in necesidad de permisos especiales. El problema es que una aplicación puede introducir información potencialmente sensible, como el nombre de un usuario, en este diccionario sin que dicho usuario lo sepa. Este diccionario se encuentra en el espacio de almacenamiento del paquete com.android.providers.userdictionary y tiene el nombre de user_dict.db. ¿Almacena la app bajo estudio algún tipo de información sensible en dicho archivo? Averígualo y de ser así, muestra qué información ha almacenado.

5.7. Ejercicio 7: Análisis del receptor de mensajes exportado

La app exporta en su manifiesto un BroadcastReceiver llamado *MyBroadCastReceiver*. Activa dicho receptor de mensajes. Presta atención a cómo debe realizarse dicha activación. Para ello estudia el código contenido en el método *onReceive* del receptor. Si te fijas la activación correcta del receptor requiere de 2 strings. Analiza qué hace el receptor cuando es activado y explícalo. A la vista de lo que ves, ¿piensas que presenta

este receptor realmente un problema de seguridad? En caso afirmativo describe dicho problema.

5.8. Ejercicio 8: Análisis del proveedor de contenidos exportado

Fíjate que el proveedor de contenidos *TrackUserContentProvider* se exporta en el manifiesto de la app. Recuerda que los proveedores de contenidos son normalmente accedidos utilizando URIs que comienzan por "content://", siguen con el identificador del proveedor, y terminan con el punto de acceso al mismo, sobre el que es posible realizar peticiones (o *queries* en inglés). Activa el proveedor de contenidos inspirándote en lo que se ha explicado en teoría ¿Qué información podemos extraer del mismo?

6. Evaluación

Cada uno de los ejercicios propuestos, y bien resueltos, obtendrá una puntuación de 1.25 puntos. Se espera que el informe de la práctica describa la solución planteada y la razone. El informe debe entregarse en formato pdf, y se tendrá en cuenta el formato y estructura de las respuestas, puesto que es tan importante resolver el problema planteado, como estructurar a posteriori el razonamiento seguido de cara a hacerlo entendible. Además, es importante incluir para cada ejercicio el setup utilizado, así como enumerar las herramientas que han servido para dar respuesta al ejercicio. Este aspecto es de suma relevancia a la hora de plantear la veracidad de la respuesta, y sobre todo, su reproducibilidad, algo fundamental cuando se analiza un sistema y se desean corroborar los resultados obtenidos.