



GRADO EN INGENIERÍA TELEMÁTICA

Curso Académico 2016/2017

Trabajo Fin de Grado

MALWARE EN ANDROID: DESARROLLO DE UNA APLICACIÓN MALICIOSA

Autor : Alejandro Fernández Barroso

Tutor : Dr. Enrique Soriano Salvador

Agradecimientos

En este capítulo quiero agradecer a todas la personas que me han acompañado durante este camino y que me han hecho la persona que soy hoy en día.

En primer lugar a toda mi familia por guiarme en todo momento y ayudarme a tomar decisiones importantes a lo largo de toda mi vida. A mis padres Agustín y Guadalupe por enseñarme una lección cada día, darme la posibilidad de formarme y no parar de insistir para que me convirtiera en la persona que soy. A mi hermana Elena por crecer a mi lado y darme tantos buenos momentos. También a María por apoyarme en todo momento y demostrarme que el esfuerzo siempre tiene su recompensa.

A mis compañeros de grado con los que he compartido buenos y malos momentos durante toda esta experiencia. En especial a Alberto, Adrián, María Cristina y Sara.

No olvidar tampoco a mis compañeros de trabajo los cuales me han introducido en el mundo laboral y me han enseñado mucho de esta nueva etapa.

Por último a todos los profesores que me han formado durante toda la carrera, en especial a Enrique por guiarme en este proyecto e introducirme el mundo de la seguridad informática al cual quiero dedicar mi carrera profesional.

Resumen

Cada vez son más las personas que usan su smartphone con mayor frecuencia tanto para su vida personal, como para su vida profesional. Debido a este crecimiento los ciberdelincuentes han decidido especializarse para poder dirigir sus ataques hacia este tipo de dispositivos, ya que suelen tener gran cantidad de información confidencial.

Suele ser frecuente mezclar aspectos personales y profesionales en los dispositivos móviles, lo que hace que sean un buen objetivo para los ciberdelincuentes. Una de las herramientas que se usan para realizar estos ataques son los RAT.

Un RAT (Remote Access Tool) es una herramienta de administración remota que permite al atacante tener el control total sobre el dispositivo atacado.

En este proyecto se va a desarrollar un RAT para simular un ataque de este tipo sobre un móvil. Además se explicarán las diferentes medidas que se deben tener en cuenta para no sufrir este tipo de ataques en un escenario real.

Índice general

1. Introducción	1
1.1. Riesgos y ataques	2
1.2. Explotación y vulnerabilidades	8
1.2.1. Certifi-Gate	9
1.3. Contramedidas	10
2. Objetivos	13
2.1. Descripción del problema	13
2.2. Objetivos	13
3. Descripción del trabajo desarrollado	15
3.1. Metodología	15
3.2. Iteraciones	16
3.3. Infraestructura y herramientas	17
3.3.1. Android	17
3.3.2. Android Studio	18
3.3.3. Java	19
3.3.4. Bouncy Castle	20
3.4. Desarrollo de la aplicación	21
3.4.1. Protocolo de comunicación	22
3.4.2. Diseño e implementación	25
3.4.3. Explotación de Certifi-gate	31
3.5. Funcionamiento de la aplicación	32
3.5.1. Instalación y ejecución de la aplicación “troyanizada”	32

3.5.2. Funcionamiento del C&C	34
4. Conclusiones	39
4.1. Consecución de objetivos	39
4.2. Competencias utilizadas y adquiridas	41
4.3. Trabajos futuros	41
Bibliografía	43
A. URL de descarga	47

Índice de figuras

1.1. Evolución del uso de los sistemas operativos	2
1.2. Caso real de Adware	3
1.3. Caso real de ataque Phising	3
1.4. Rammsonware Wannacry	5
1.5. Command and Control	6
1.6. RAT Darkcomet	7
1.7. Smash the stack	8
1.8. Permiso privilegiado Read_Frame_Buffer	10
3.1. Desarrollo en espiral	16
3.2. Arquitectura del sistema Android	17
3.3. Dispositivo móvil Samsung Galaxy SII	18
3.4. Entorno de porgramación Android Studio	19
3.5. Entorno de porgramación Eclipse	20
3.6. Esquema general del proyecto	23
3.7. Declaración de permisos en el fichero AndroidManifest.xml	27
3.8. Vulnerabilidad en el plug-in de TeamViewer	32
3.9. Correo electrónico recibido por la víctima	33
3.10. Permisos solicitados por la aplicación maliciosa	33
3.11. Aspecto visual de la aplicación FlashLight	34
3.12. C&C a la espera de la conexión de la aplicación	34
3.13. Aplicación conectada al C&C	35
3.14. Ejecución de la orden contacts	35
3.15. Ejecución de la orden apps	35

3.16. Ejecución de la orden info	36
3.17. Ejecución de la orden screenshot	36
3.18. Ejecución de la orden images	37
4.1. Interfaz visual del troyano DroidJack	42

Capítulo 1

Introducción

Poco a poco los dispositivos móviles han ido recortando terreno a los ordenadores personales, tanto que actualmente los móviles y las tablets igualan en uso a estos.

En cuanto a sistemas operativos se refiere, Android ha sido el sistema que mayor crecimiento ha tenido, llegando a igualar a los sistemas operativos de escritorio como Windows. Esta evolución ha llevado a ciberdelincuentes y otros atacantes, como organizaciones, gobiernos, etcétera, a cambiar sus objetivos de ataque.

Los móviles tienen gran cantidad de información de distintos tipos como pueden ser datos bancarios, datos personales, ocio, etcétera. Además el reciente crecimiento del BYOD (Bring Your Own Device) en las empresas ha hecho que llevemos datos confidenciales de nuestra empresa en nuestro dispositivo personal. Esto ha generado una motivación extra para que los atacantes se especialicen en el campo de los dispositivos móviles.

Se prevé que en el futuro se continúe con este crecimiento. Por tanto debemos conocer de qué manera nos pueden atacar para así saber defendernos.

Este trabajo consiste en el diseño e implementación de un malware de ejemplo para Android con el fin de simular un ataque real. Se define como malware todo programa o código informático cuya función es infiltrarse en un sistema, dañarlo o causar un mal funcionamiento del mismo.

En las siguientes secciones se definen algunos de los riesgos y ataques más populares que existen en Android, como explotar dichos riesgo y cómo defendernos ante los mismos.

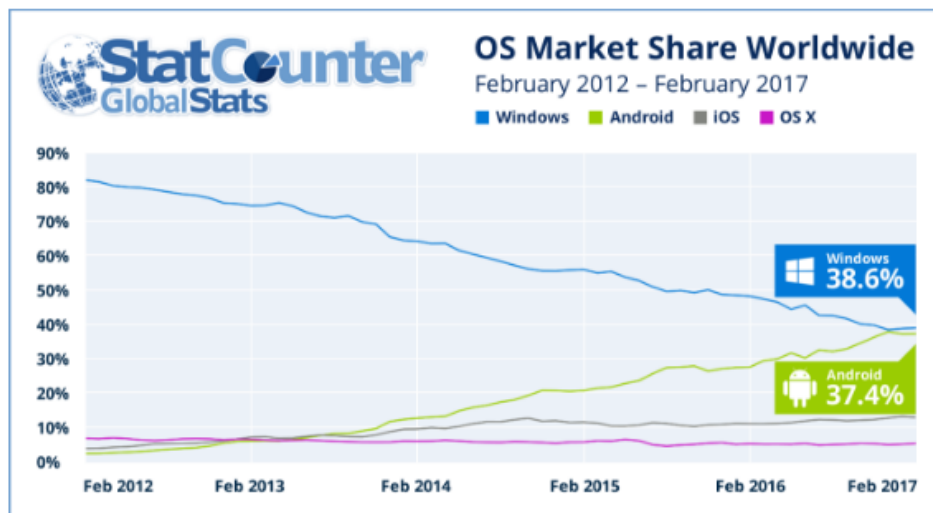


Figura 1.1: Evolución del uso de los sistemas operativos

1.1. Riesgos y ataques

Dentro de Android los riesgos y ataques más comunes actualmente son:

- **Denegación de servicio.** Este ataque es conocido como DoS (Denial of Service) y su objetivo es bloquear el dispositivo atacado y denegar el servicio a los usuarios legítimos del mismo, de manera que queda totalmente inutilizado. Hay diferentes maneras de provocar este efecto. Por ejemplo, provocar que el servicio entre en un bucle infinito, que el dispositivo no deje de reiniciarse o saturar un servicio con peticiones no legítimas para que no pueda atender a peticiones reales. Un ejemplo de este tipo de ataque es el Caso Ronnie [1], cuyo nombre se debe a su autor. El ataque fue dirigido contra los servidores de chats IRC-Hispano y llegó a afectar en algunos momentos al 30 % de los internautas españoles.
- **Adware.** El Adware consiste en la integración de publicidad dentro de las aplicaciones. Es muy común dentro de Android ya que suele ser la manera con la que los desarrolladores monetizan sus aplicaciones, lo que hace más complicado categorizarlo, ya que si la publicidad se integra de forma legítima en la aplicación, no se puede considerar malware.

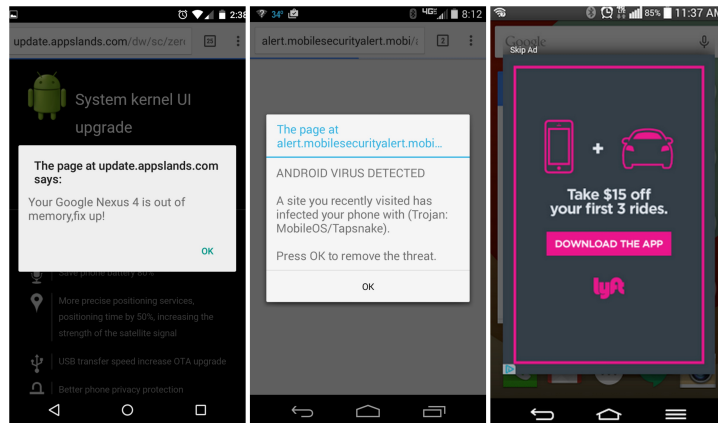


Figura 1.2: Caso real de Adware

- **Phishing.** Este modelo se basa en el engaño para obtener información del usuario. Los ejemplos más comunes de este tipo de ataques son correos electrónicos que simulan proceder del banco del usuario y le solicitan las credenciales. De esta manera el usuario al confiar que es un correo legítimo le proporciona sus claves al atacante.



Figura 1.3: Caso real de ataque Phishing

- **Spyware.** Un spyware tiene como fin la obtención de información: dirección de red del dispositivo, versión, dirección MAC, número de teléfono, cuenta de correo, etcétera. Se

instalan en el dispositivo, monitorizando y enviando al atacante toda la información del sistema. Además bajan el rendimiento del dispositivo ya que trabajan a la vez que las aplicaciones legítimas.

- **Keyloggers.** El objetivo de este tipo de malware la monitorización de la interacción del usuario con el teclado, enviando dicha información al atacante. Se usan para obtener usuarios y contraseñas, números de tarjeta, números PIN, etcétera.
- **Tap-jacking.** En este caso el Tap-jacking consiste en engañar al usuario para que mediante el uso de taps en la pantalla realice una función distinta a la que el usuario desea realizar, superponiendo controles ficticios sobre los que en realidad está presionando.
- **Clickers.** Los atacantes usan Clickers, para cargar webs y hacer clicks sobre enlaces para mejorar el posicionamiento y de esta manera obtener beneficio. Este ataque tiene un impacto negativo en dispositivos que tengan tarifa de datos ya que se hace uso de internet sin consentimiento del usuario.
- **Ramsonware.** Este tipo de malware secuestra los archivos del usuario y pide un rescate por ellos. Técnicamente consiste en el cifrado de los archivos. Para poder recuperar los archivos es necesario descifrar con la clave asociada que se proporcionará (supuestamente) al pagar el rescate.

Un caso reciente de este tipo de malware es el rammsonware Wannacry [2], este ataque afectó en Mayo del 2017 a varias de las grandes empresas españolas. Este ransomware se propaga como un gusano a través de la red gracias a una vulnerabilidad del sistema operativo Windows, y cuando infecta a una máquina, cifra los ficheros y pide un ingreso de una cantidad de Bitcoins como rescate.

- **Servicios de pago.** El atacante usa este tipo de malware con la intención de obtener beneficio al forzar al usuario a que consuma servicios de pago de manera ilegítima. El ejemplo más claro de este modelos son aplicaciones que suscriben al usuario a servicios SMS Premium o que realizan llamadas a números de teléfono con sobrecoste sin tener el consentimiento del usuario.
- **Gusano.** Un gusano es un software que se replica a sí mismo y se propaga por la red. Para que sean más consistentes suelen tener varios métodos de propagación. El gusano



Figura 1.4: Rammsonware Wannacry

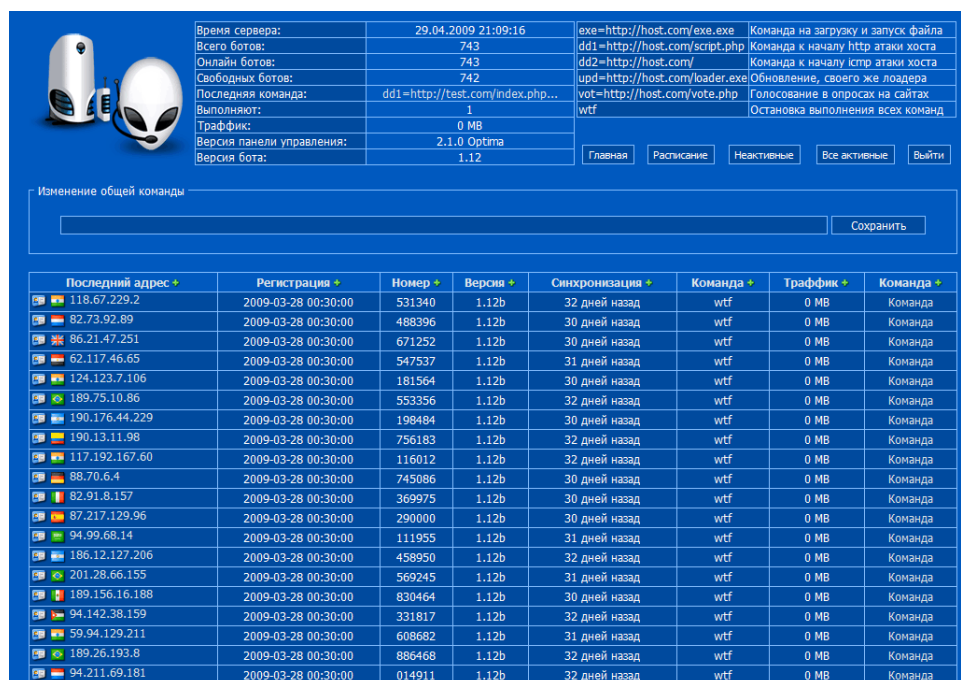
conficker [3] es un ejemplo de este tipo de malware. Este gusano se aprovechaba de una vulnerabilidad de buffer overflow en Windows Server Service y se transmitía en el auto-arranque de discos externos. Bloqueaba peticiones de DNS, deshabilitar actualizaciones, etcétera.

Otro ejemplo de ataque que usaba esta técnica es el Rammsoware Wannacry, definido anteriormente. En este caso el malware se propagaba como un gusano usando una vulnerabilidad en el SMB (Server Message Block) de Windows.

- **Troyano.** El troyano a diferencia del gusano no se propaga a sí mismo, es un malware que simula ser un programa legítimo pero que ocasiona daños intencionadamente. En la mayoría de los casos los troyanos crean una puerta trasera que permite la administración remota del sistema objetivo a un atacante. El troyano denominado Zeus o Zbot [4] tenía como objetivo la obtención de credenciales bancarias u otro tipo de información relevante.
- **Botnet.** Una botnet es una red de equipos infectados y controlados por un atacante de forma remota. Para infectar estos equipos el atacante utiliza un malware y una vez infectados los equipos pasan a llamarse “zombies” o “bots”. Se suelen usar generalmente para realizar ataques de denegación de servicio, ya que al ser equipos de distinta proceden-

cia los servidores atacados no pueden paliar fácilmente el ataque. Actualmente existe un gran mercado con este tipo de redes y es posible comprar botnet de distinta procedencia y capacidad.

Es muy común el uso de un C&C para gestionar y controlar Botnets. C&C son las siglas de *Command and Control* y se refiere a la infraestructura de mando y control que se utiliza para manejar un malware que afecta a un dispositivo atacado.



Последний адрес	Регистрация	Номер	Версия	Синхронизация	Команда	Трафик	Команда
118.67.229.2	2009-03-28 00:30:00	531340	1.12b	32 дней назад	wtf	0 MB	Команда
82.73.92.89	2009-03-28 00:30:00	488396	1.12b	30 дней назад	wtf	0 MB	Команда
86.21.47.251	2009-03-28 00:30:00	671252	1.12b	30 дней назад	wtf	0 MB	Команда
62.117.46.65	2009-03-28 00:30:00	547537	1.12b	31 дней назад	wtf	0 MB	Команда
124.123.7.106	2009-03-28 00:30:00	181564	1.12b	30 дней назад	wtf	0 MB	Команда
189.75.10.86	2009-03-28 00:30:00	553356	1.12b	32 дней назад	wtf	0 MB	Команда
190.176.44.229	2009-03-28 00:30:00	198484	1.12b	30 дней назад	wtf	0 MB	Команда
190.13.11.98	2009-03-28 00:30:00	756183	1.12b	32 дней назад	wtf	0 MB	Команда
117.192.167.60	2009-03-28 00:30:00	116012	1.12b	32 дней назад	wtf	0 MB	Команда
88.70.6.4	2009-03-28 00:30:00	745086	1.12b	30 дней назад	wtf	0 MB	Команда
82.91.8.157	2009-03-28 00:30:00	369975	1.12b	30 дней назад	wtf	0 MB	Команда
87.217.129.96	2009-03-28 00:30:00	290000	1.12b	30 дней назад	wtf	0 MB	Команда
94.99.68.14	2009-03-28 00:30:00	111955	1.12b	31 дней назад	wtf	0 MB	Команда
186.12.127.206	2009-03-28 00:30:00	458950	1.12b	32 дней назад	wtf	0 MB	Команда
201.28.66.155	2009-03-28 00:30:00	569245	1.12b	31 дней назад	wtf	0 MB	Команда
189.156.16.188	2009-03-28 00:30:00	830464	1.12b	30 дней назад	wtf	0 MB	Команда
94.142.38.159	2009-03-28 00:30:00	331817	1.12b	32 дней назад	wtf	0 MB	Команда
59.94.129.211	2009-03-28 00:30:00	608682	1.12b	31 дней назад	wtf	0 MB	Команда
189.26.193.8	2009-03-28 00:30:00	886468	1.12b	32 дней назад	wtf	0 MB	Команда
94.211.69.181	2009-03-28 00:30:00	014911	1.12b	32 дней назад	wtf	0 MB	Команда

Figura 1.5: Command and Control

- **Rootkit.** Son herramientas para ocultar una intrusión de los malwares que se han comentado anteriormente.

Los rookits permiten el acceso privilegiado a un sistema pero mantiene su presencia oculta al control del administrador. El atacante suele integrar el rootkit después de haber penetrado un sistema para así conseguir un acceso permanente sin rastro.

- **RAT.** Un RAT permite al atacante obtener el control remoto del dispositivo, permitiendo realizar todo tipo de operaciones sobre él. Este tipo de operaciones pueden ser: leer mensajes SMS, abrir navegador, obtener capturas de pantalla, obtener contactos y un largo etcétera. RAT son las siglas de Remote Access Tool, si el malware viene oculto dentro de otra aplicación se denomina Remote Access Trojan.

Este tipo de aplicación se comunica con un servidor de tipo C&C (Command and Control) que será el que realice las peticiones para que el dispositivo realice lo que se requiera.

Un ejemplo de este tipo de malware es DarkComet [5], el cual crea un ejecutable personalizado para mandar a la víctima y cuando esta lo ejecuta se conecta a un programa controlador que permite monitorizar el sistema, activar el micrófono, capturar las teclas, etcétera.

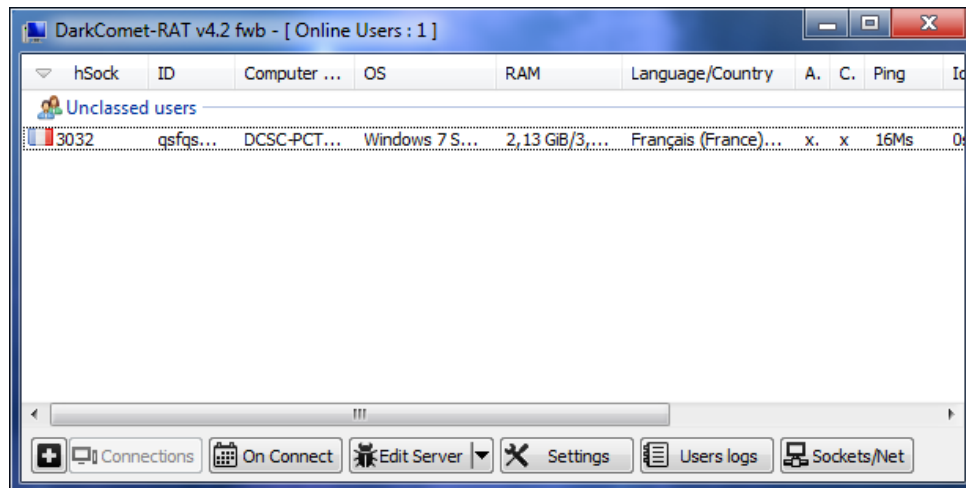


Figura 1.6: RAT Darkcomet

Este modelo de malware es el que se va a reproducir en este trabajo por lo que se definirá con más detalle en apartados posteriores.

1.2. Explotación y vulnerabilidades

Una vulnerabilidad es un fallo del sistema que puede ser utilizada para causar algún daño. Las vulnerabilidades aparecen en el propio sistema operativo o en cualquier aplicación que se instale en el mismo. Para explotar las vulnerabilidades los atacantes utilizan exploits, que son fragmentos de código que se aprovechan de los agujeros de seguridad de un sistema o aplicación para obtener un comportamiento deseado en el mismo.

La técnica más común se basa en el desbordamiento de un buffer para sobrescribir ciertas zonas de la memoria. Un ejemplo es el exploit Smash the stack [6], que consiste en inyectar en un buffer situado en la pila, el código que se quiere ejecutar, y además sobrepasar los límites del mismo para reescribir la dirección de retorno de un registro de activación (o stack frame) para modificar el flujo de ejecución y saltar a dicho código.

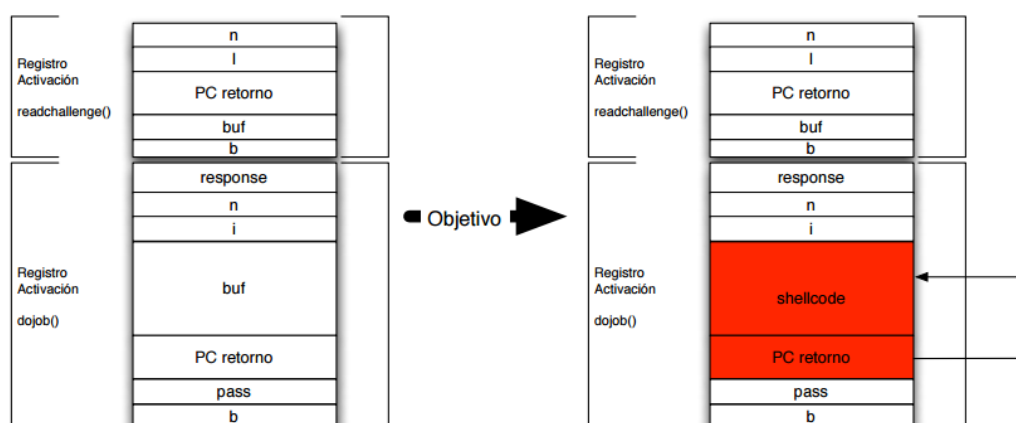


Figura 1.7: Smash the stack

Los exploits se pueden clasificar por conocidos o no conocidos. Los exploits conocidos son aquellos de los cuales se tiene constancia y se pueden tomar medidas para evitar que infecten el sistema. Los fabricantes crean parches de seguridad para solucionar estas vulnerabilidades conocidas.

Los 0-day son exploits que se aprovechan de vulnerabilidades desconocidas por los usuarios y los fabricantes, lo que supone que no existe ninguna actualización o parche de seguridad.

Para este tipo de exploits también existe un mercado ya que permiten a los delincuentes

atacar a empresas, gobiernos o incluso infraestructuras críticas. Por ejemplo, un 0-day para iOS que permita ejecución remota en modo privilegiado tiene un precio de 1.5 millones de dólares en el mercado legal (Zerodium [7])

Un vulnerabilidad reportada es Certifi-gate [8], la cual se va a usar en este proyecto para poder efectuar operaciones privilegiadas en el dispositivo infectado. En la siguiente subsección se va a explicar en detalle en qué consiste dicha vulnerabilidad.

1.2.1. Certifi-Gate

Cualquier aplicación en Android necesita ciertos permisos para ser instalada en el dispositivo. Por ejemplo la aplicación Instagram [9] necesita permisos para acceder a la cámara, al almacenamiento interno y externo, etcétera.

Sin embargo, hay ciertos permisos que no pueden ser demandados por parte de ninguna aplicación de terceros que no esté autorizada para ello, ya que son considerados como permisos privilegiados y se rompería el principio de aislamiento de las aplicaciones Android. Este tipo de permisos nos permiten por ejemplo insertar pulsaciones sobre la pantalla o realizar capturas fuera del contexto de la aplicación.

A estos permisos sólo pueden acceder aplicaciones preinstaladas en la ROM (Read Only Memory) o firmadas por el fabricante OEM (Original Equipment Manufacturer).

Las aplicaciones MRST (Mobile Remote Support Tools), sirven para dar soporte remoto al usuario. Aplicaciones como TeamViewer [10] o Rsupport [11] son ejemplos de este tipo de aplicaciones.

Para poder controlar la pantalla del dispositivo que solicite soporte, estas herramientas de control necesitan obtener estos permisos. Por tanto los desarrolladores de estas aplicaciones se asocian con el fabricante para firmarlas y de esta manera conseguir estos privilegios de acceso.

Para no tener que pasar por el fabricante cada vez que se genere una nueva versión del MRST para que este la firme, este tipo de aplicaciones se separan en dos trozos: la aplicación en sí y un plug-in que es el encargado de realizar las acciones que requieran permisos privilegiados.

Certifi-gate aprovecha las distintas vulnerabilidades que tiene el código del plug-in del MRST para comprobar que la aplicación es legítima. De esta manera, cualquier otra aplicación se puede hacer pasar por una aplicación legítima y conseguir realizar acciones privilegiadas como tomar capturas o insertar pulsaciones.

READ_FRAME_BUFFER

`String READ_FRAME_BUFFER`

Allows an application to take screen shots and more generally get access to the frame buffer data.

Not for use by third-party applications.

Constant Value: "android.permission.READ_FRAME_BUFFER"

Figura 1.8: Permiso privilegiado Read_Frame_Buffer

1.3. Contramedidas

A continuación se enumeran una serie de medidas que se deben tomar para tener nuestro dispositivo lo más protegido posible:

- Tener la última versión del sistema operativo. Las versiones antiguas pueden tener vulnerabilidades que hayan sido parcheadas en versiones posteriores. Tener actualizado nuestro dispositivo con la última versión del sistema reducirá la probabilidad de éxito de un atacante.
- Al igual que en versiones antiguas del sistema operativo, las aplicaciones que no estén actualizadas a su última versión pueden contener también fallos de seguridad que pueden ser explotados. Por tanto debemos las aplicaciones actualizadas a su última versión.
- Al instalar una nueva aplicación se debe comprobar los permisos que solicita para su funcionamiento. Por ejemplo, no es lógico que una aplicación que enciende la linterna nos solicite tener acceso a la cámara, contactos, internet, etcétera.
- Las aplicaciones que instalemos siempre deben ser descargadas desde la plataforma de distribución de aplicaciones oficial de cada sistema, ya que estas plataformas analizan las aplicaciones e intentan evitar que se distribuyan aplicaciones maliciosas. Por ejemplo, la plataforma oficial de Android es Google Play [12], la cual es más segura que otras plataformas como Aptoide [13] en la cual se pueden encontrar multitud de aplicaciones fraudulentas.
- Además de descargar las aplicaciones desde la plataforma oficial, también debemos evitar instalar aplicaciones que tengan orígenes desconocidos. Android por defecto no permite

instalar ninguna aplicación que cuya fuente sea desconocida. Pero es muy común por los usuarios deshabilitar esta opción para poder instalar cualquier tipo de aplicación.

- No conectarse a redes Wi-Fi desconocidas, un atacante puede simular ser una red legítima, proporcionar acceso a internet pero a la vez obtener todo el tráfico que estemos cursando: las paginas web que visitamos, usuarios y contraseñas, tarjetas del banco, etcétera.
- Además de todas estas medidas debemos de tener en cuenta otras por si nos sustraen el móvil y el atacante tiene acceso total al dispositivo (Big Stick Principle). Tener un buen código de desbloqueo no permitirá al atacante acceder fácilmente al dispositivo. Otra buena medida es tener el dispositivo cifrado, de esta manera aunque el atacante conectase el móvil a un ordenador y consiguiera acceder a los datos no podría leerlos ya que estarán cifrados.

Capítulo 2

Objetivos

2.1. Descripción del problema

Con el fin de aprender a defendernos, se va a simular un ataque malware RAT para el cual se necesita desarrollar un malware de tipo troyano, que se instale haciéndose pasar por una aplicación convencional. De esta manera se entenderá como funcionan este tipo de ataques para poder prevenirlos en un futuro.

2.2. Objetivos

Una vez enunciado el problema se plantean los siguientes objetivos:

1. Desarrollo de una aplicación para Android que ejecute un troyano en background.
2. Explotación de la vulnerabilidad Certifi-gate, detallada en la subsección 1.2.1, con el fin de acceder a operaciones privilegiadas, como obtener capturas de pantalla de otras aplicaciones.
3. Implementación de un C&C que permita controlar remotamente el dispositivo atacado. Desde el C&C se podrán realizar distintas peticiones que serán respondidas por el troyano.
4. Conexión reversa desde la aplicación “troyanizada”. El troyano debe conectarse al C&C automáticamente (“llamar a casa”), con el objetivo de traspasar los cortafuegos que pro-

tegen la red en la que se encuentra el dispositivo afectado. El cortafuegos interno suele ser menos restrictivo que el externo.

5. Obtención de la información del dispositivo. El C&C debe de ser capaz de obtener la siguiente información del dispositivo:
 - a)* Capturas de pantalla fuera del contexto de la aplicación.
 - b)* Aplicaciones instaladas.
 - c)* Contactos.
 - d)* Información del dispositivo.
 - e)* Imágenes de galería.
 - f)* Archivos.
6. Desarrollo de un protocolo de comunicación basado en llamadas a procedimiento remoto (RPC) no dependiente de un lenguaje de programación o una plataforma concreta.
7. Cifrado de la comunicación. El intercambio de información entre la aplicación y el C&C debe estar cifrado, de manera que si se está observando el tráfico de la red no se pueda entender la comunicación entre C&C y aplicación.

Capítulo 3

Descripción del trabajo desarrollado

3.1. Metodología

Para el desarrollo de este proyecto se ha llevado a cabo la metodología conocida con el nombre de “desarrollo en espira” [14]. Este modelo nos permite desarrollar el software de manera incremental, aumentando progresivamente la dificultad de los problemas planteados y presentando al final de cada etapa una versión funcional del software. El desarrollo en espiral se compone de una serie de ciclos. Cada ciclo se compone de cuatro actividades, las cuales se detallan a continuación:

1. **Determinar Objetivos.** En esta fase se fijan los objetivos que se deben cumplir al final de cada iteración, teniendo en cuenta el objetivo final del software.
2. **Análisis del Riesgo.** Se estudian las causas de las amenazas y eventos no deseados junto con los daños que estos pueden producir. También se buscan soluciones para evitar dichos daños.
3. **Desarrollar y probar.** Se desarrolla el software y una vez que se finaliza se realizan una serie de pruebas para cerciorar que se cumplen los objetivos definidos en la iteración.
4. **Planificación.** En la última actividad se comienza a planificar la siguiente fase, teniendo en cuenta los objetivos obtenidos y los problemas surgidos en la fase actual para evitarlos en la siguiente.

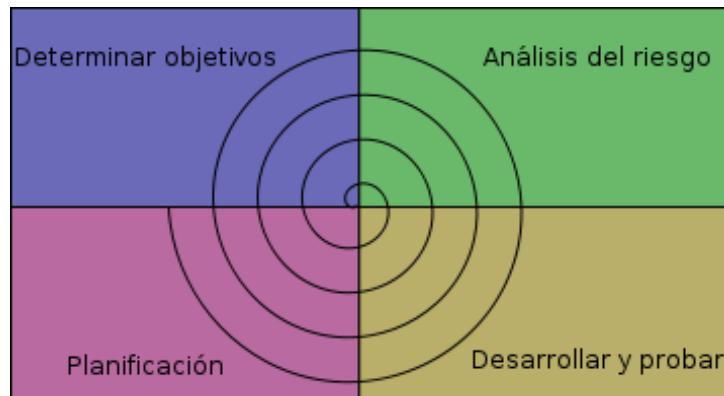


Figura 3.1: Desarrollo en espiral

3.2. Iteraciones

Para el desarrollo del proyecto se llevaron a cabo las siguientes iteraciones:

- Creación de un protocolo de intercambio de mensajes. En esta fase se configuró un protocolo de comunicación en el cual se basan la aplicación y el C&C para intercambiar sus mensajes.
- Desarrollo aplicación Android que nos permita la obtención de toda la información posible del dispositivo usando los permisos definidos en el archivo `AndroidManifest.xml`.
- Desarrollo de servidor Java que mediante el uso de RPC permita la obtención de la información de la aplicación desarrollada en Android.
- Integración del protocolo de mensajes en ambos programas. Se acopla el protocolo de mensajes en el código para que ambos programas puedan comunicarse entre sí.
- Evolución del protocolo de mensajes a un protocolo seguro y cifrado. Tal y como nos marcan los objetivos se mejora el protocolo de mensajes para que la comunicación entre C&C y aplicación sea cifrada y segura.
- Reproducción del ataque Certifi-gate. En esta etapa se analiza el código que explota la vulnerabilidad Certifi-gate y se adapta a un programa para conseguir reproducir el ataque.
- Integración de Certifi-gate en el código de la aplicación Android. Se introduce el código que explota la vulnerabilidad en la aplicación Android.

- Pruebas. En esta última fase se realizan distintas pruebas para comprobar que ambas partes(C&C y aplicación) están bien acopladas, que el código funciona correctamente y que podemos obtener toda la información que se había planteado en los objetivos.

3.3. Infraestructura y herramientas

En este apartado se va a describir la infraestructura en la que se basa el proyecto: lenguajes de programación, bibliotecas de soporte y sistemas operativos usados.

3.3.1. Android

Android es un sistema operativo para dispositivos móviles cuyo núcleo está basado en Linux. El sistema operativo proporciona las interfaces necesarias para desarrollar aplicaciones que accedan a todas las funciones del teléfono de manera muy cómoda usando Java como lenguaje de programación.



Figura 3.2: Arquitectura del sistema Android

En concreto para este proyecto se va a usar la versión 4.1.2 de Android que es la que contiene la vulnerabilidad se va a explotar.

El dispositivo Android sobre el que se va realizar la simulación del ataque es un Samsung SII. Se ha elegido este dispositivo por disponibilidad, por tener la versión de Android deseada y por ser una marca que contiene la vulnerabilidad a explotar.



Figura 3.3: Dispositivo móvil Samsung Galaxy SII

3.3.2. Android Studio

Para desarrollar en Android se ha usado el IDE (Integrated Development Environment) Android Studio [15].

Este entorno de desarrollo contiene las siguientes características:

- Integración de ProGuard y funciones de firma de aplicaciones.
- Renderizado en tiempo real.
- Consola de desarrollador: consejos de optimización, ayuda para la traducción, estadísticas de uso.
- Soporte para construcción basada en Gradle.
- Refactorización específica de Android y arreglos rápidos.

- Un editor de diseño enriquecido que permite a los usuarios arrastrar y soltar componentes de la interfaz de usuario.
- Herramientas Lint para detectar problemas de rendimiento, usabilidad, compatibilidad de versiones, y otros problemas.
- Plantillas para crear diseños comunes de Android y otros componentes.
- Soporte para programar aplicaciones para Android Wear.
- Soporte integrado para Google Cloud Platform, que permite la integración con Google Cloud Messaging y App Engine.
- Un dispositivo virtual de Android que se utiliza para ejecutar y probar aplicaciones.

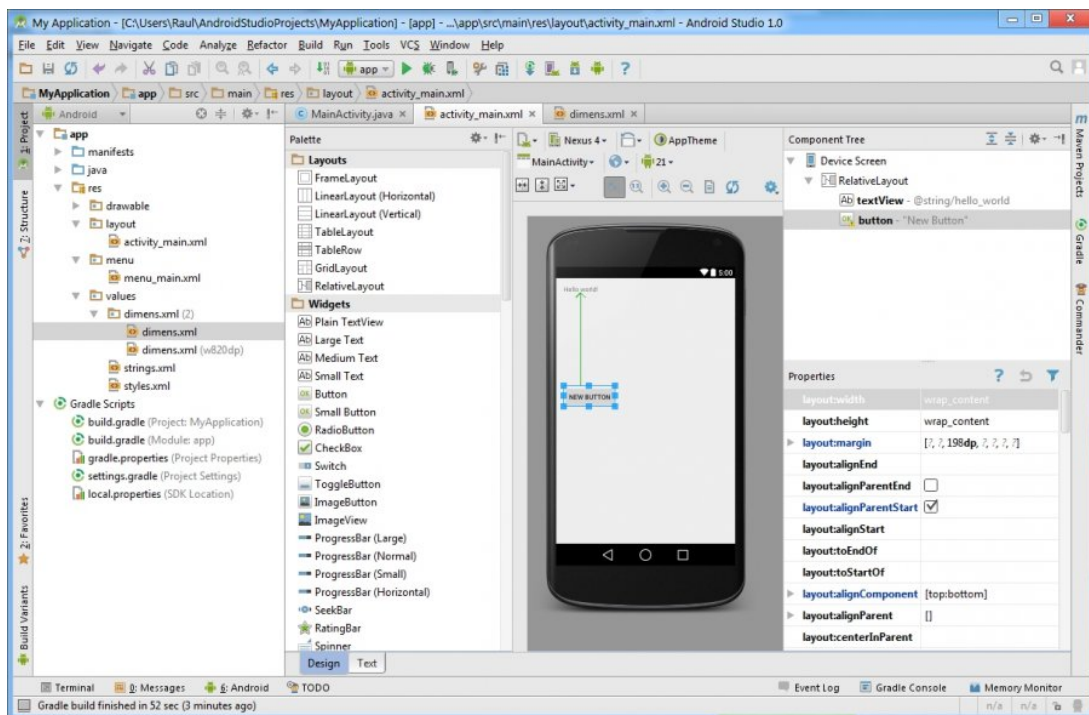


Figura 3.4: Entorno de programación Android Studio

3.3.3. Java

Java [16] es un lenguaje de programación de propósito general, concurrente y orientado a objetos. Fue diseñado para que los programadores escriban el código una sola vez y lo ejecuten

en cualquier dispositivo, este modelo se conoce como WORA (Write Once, Run Anywhere), lo que significa que el código ejecutado en una plataforma no tiene que ser recompilado para ejecutar en otra.

Este lenguaje es el que se suele usar comúnmente para el desarrollo de programas cliente-servidor, como es el caso de este proyecto y uno de los motivos por los cuales se ha elegido este lenguaje de programación. Otro de los muchos motivos por los cual se ha elegido Java es la facilidad de integración de librerías externas, las cuales se comentarán en apartados posteriores.

Para desarrollar en este lenguaje se ha usado el IDE Eclipse [17].

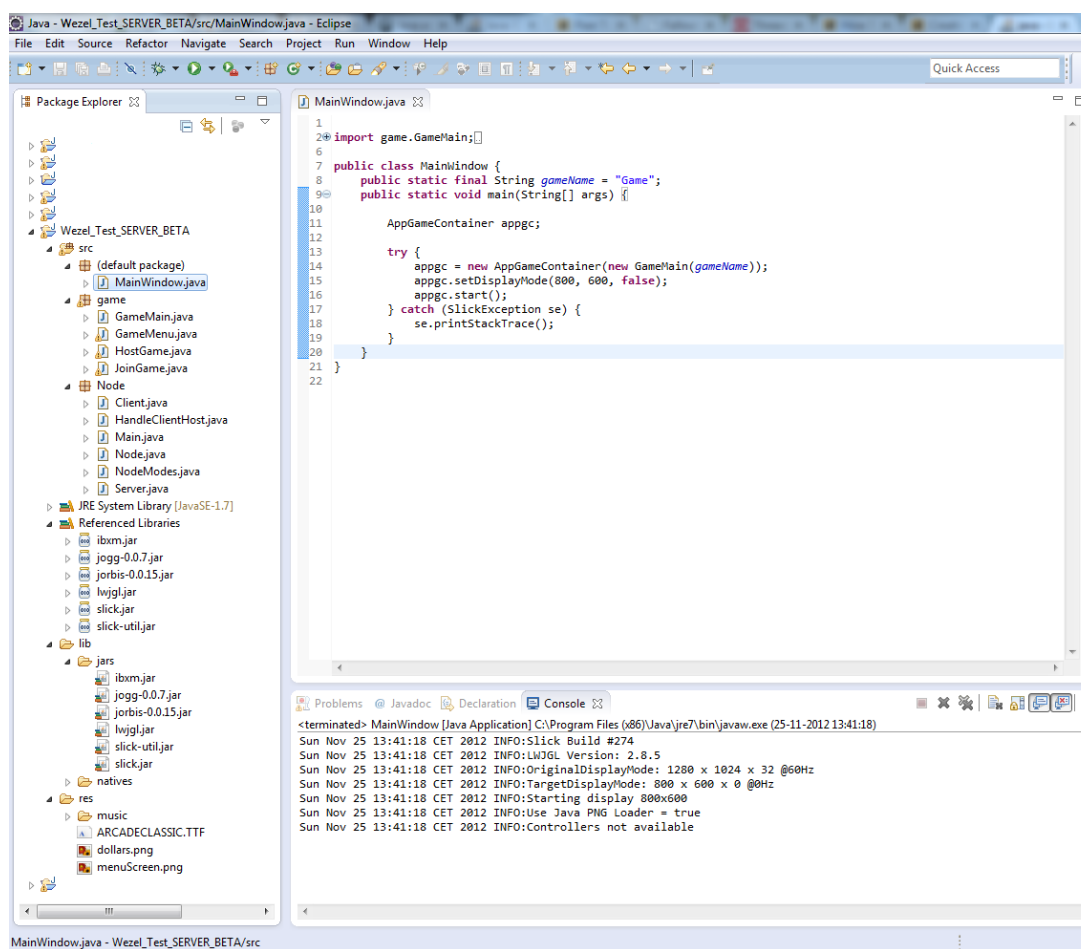


Figura 3.5: Entorno de programación Eclipse

3.3.4. Bouncy Castle

Bouncy Castle [18] es una agrupación de APIs (Application Programming Interface) utilizados en criptografía. El API está optimizado para usar los algoritmos de cifrado eficientemente

en entornos de bajo recursos o en los cuales no se pueden usar librerías. La biblioteca contiene:

- Un API criptográfico ligero para Java y C#.
- Implementación de JCE 1.2.1.
- Una librería para lectura y escritura de objetos ASN. 1.
- API ligero para TLS y DTLS.
- Generador de Certificados X.509 v.1/v.3, CRLs v.2 y PKCS12.
- Generador/Procesador para S/MIME and CMS.
- Generador/Procesador para OCSP.
- Generador/Procesador para TSP.
- Generador/Procesador CMP y CRMF.
- Generador/Procesador para OpenPGP.
- Generador/Procesador para ECA (Extended Access Control).
- Generador/Procesador para DVCS (Data Validation and Certification Server).

Spongy Castle [19] es la versión, con ligeros cambios, de Bouncy Castle que está preparada para integrar en Android.

3.4. Desarrollo de la aplicación

Para la realización de la simulación del ataque es necesario desarrollar tanto la aplicación que tiene el código malicioso, como el C&C que envía las ordenes para obtener la información.

En este caso la aplicación, llamada **Flashlight**, será desarrollada en Android, ya que el sistema operativo que se va a explotar es la versión de Android 4.1.2.

El desarrollo del C&C, llamado MRATServer, será en Java, los motivos por los cuales se ha elegido este lenguaje se definieron en el apartado de infraestructura. Aunque se podría haber

usado cualquier otro lenguaje para este desarrollo ya que el protocolo de mensajes es interoperable con los distintos lenguajes de programación.

El usuario comprometido ejecutará la aplicación llamada **Flashlight** y al pulsar el botón LIGHT empezará a correr un servicio en background que intentará conectarse al C&C desde el cual se envían las peticiones.

Por otro lado el C&C, ejecutado en un PC, esperará la conexión de algún dispositivo infectado, una vez que tenga conexión el C&C podrá realizar peticiones para controlar el dispositivo afectado.

3.4.1. Protocolo de comunicación

Tal y como se enunciaba en los objetivos, la comunicación entre la aplicación y el C&C debe estar cifrada para que no sea detectada si alguien está monitorizando la red. Por tanto se ha diseñado el siguiente protocolo.

Todas claves usadas en el protocolo son claves AES [20] (Advanced Encryption Standard) de 256 bits. AES es un esquema de cifrado simétrico por bloques adoptado como el estándar actual. El modo de operación que se usa en este proyecto es GCM [21] (Galois Counter Mode), ya que usando este modo se proporciona confidencialidad e integridad.

Al iniciar el servicio de la aplicación se producirá un handshake (proceso de negociación) entre esta y el C&C para obtener la clave de sesión de toda la comunicación. La aplicación enviará un mensaje `SendID` con el ID que tiene predefinido. El C&C responderá con un mensaje `TSendKey` con la clave de sesión, cifrado con la clave simétrica asociada al ID que la aplicación y el C&C comparten. Cuando la aplicación recibe este mensaje lo descifra con la clave asociada a su ID, lo almacena para usarlo en los siguientes mensajes y responde con un mensaje `ROk`, con el que confirma al C&C que ha recibido el mensaje anterior y que se pueden comenzar a realizar peticiones.

Además, para que este protocolo no sea comprometido se ha añadido a cada mensaje una marca de tiempo (`TimeStamp`) y un número aleatorio (`Nonce`). De esta manera no se podría atacar al protocolo realizando un *Replay attack*, que consiste en capturar un mensaje entre el C&C y la aplicación, modificarlo y reenviarlo, ya que el mensaje replicado sería identificado como repetido gracias al nonce, o antiguo gracias al timestamp.

Al llegar un mensaje en el C&C se comprobará si el `TimeStamp` está dentro del margen de

tiempo y si el nonce corresponde al nonce que se envió en la petición aumentado en una unidad. Si se cumplen estos condicionantes el mensaje será legítimo, si no será descartado.

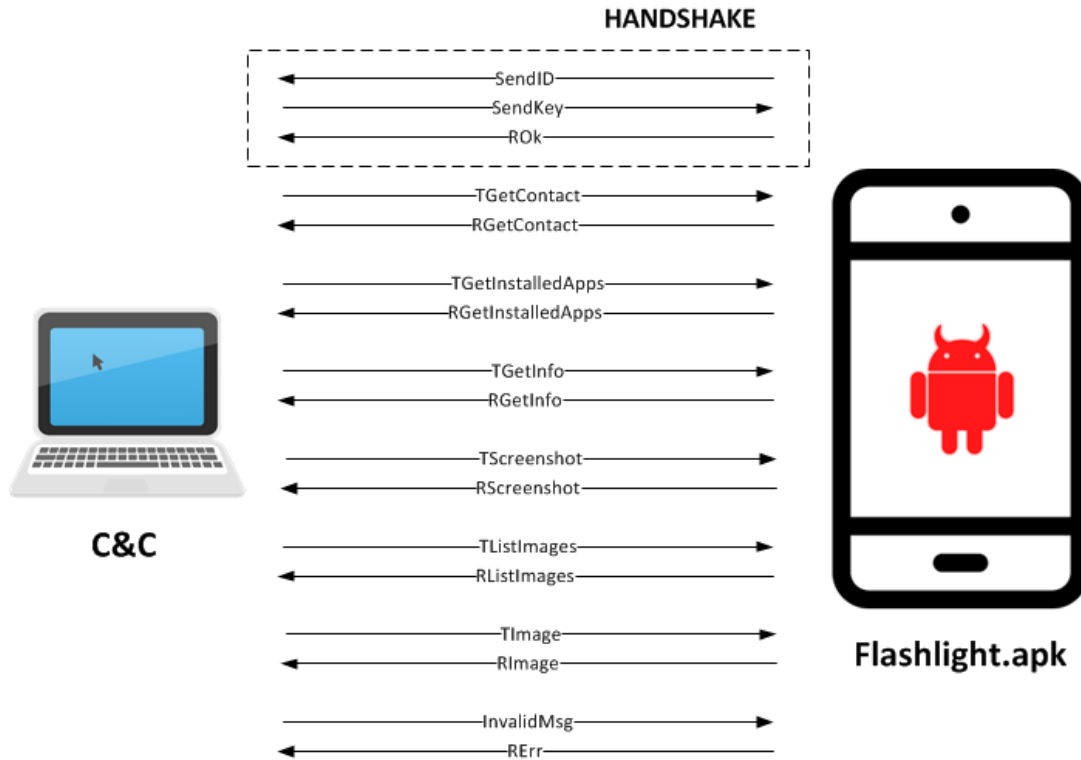


Figura 3.6: Esquema general del proyecto

El aspecto de los mensajes que componen el handshake es el siguiente:

■ Mensaje SendID:

- $C\&C \leftarrow App: SENDID, ID$

■ Mensaje SendKey:

- $C\&C \rightarrow App: SENDKEY, K_{id}(K_{session}, nonce, TS)$

■ Mensaje ROk:

- $C\&C \leftarrow App: ROK, K_{session}(nonce + 1, TS')$

Finalizado el handshake se podrá comenzar a usar el protocolo de RPCs entre C&C y la aplicación. A continuación se muestra el aspecto de cada mensaje por cada RPC:

■ RPC Contactos:

1. $C\&C \rightarrow \text{App: } TGETCONTACT, K_{session}(nonce, TS)$
2. $C\&C \leftarrow \text{App: } RGETCONTACT, K_{session}(ncontacts, names, numbers, nonce + 1, TS')$

■ RPC Aplicaciones instaladas:

1. $C\&C \rightarrow \text{App: } TGETINSTALLEDAPPS, K_{session}(nonce, TS)$
2. $C\&C \leftarrow \text{App: } RGETINSTALLEDAPPS, K_{session}(numapps, apps, nonce + 1, TS')$

■ RPC Información del dispositivo:

1. $C\&C \rightarrow \text{App: } TGETINFO, K_{session}(nonce, TS)$
2. $C\&C \leftarrow \text{App: } RGETINFO, K_{session}(info, nonce + 1, TS')$

■ RPC Captura de pantalla:

1. $C\&C \rightarrow \text{App: } TSCREENSHOT, K_{session}(nonce, TS)$
2. $C\&C \leftarrow \text{App: } RSCREENSHOT, K_{session}(size, screenshot, nonce + 1, TS')$

■ RPC Listado de imágenes:

1. $C\&C \rightarrow \text{App: } TLISTIMAGES, K_{session}(nonce, TS)$
2. $C\&C \leftarrow \text{App: } RLISTIMAGES, K_{session}(numimages, nameimages, nonce + 1, TS')$

■ RPC Imagen:

1. $C\&C \rightarrow \text{App: } TIMAGE, K_{session}(nonce, TS)$
2. $C\&C \leftarrow \text{App: } RIMAGE, K_{session}(size, image, nonce + 1, TS')$

■ RPC Mensaje no válido:

1. $C\&C \rightarrow \text{App: } INVALIDMSG, K_{session}(invalid, nonce, ts)$
2. $C\&C \leftarrow \text{App: } RERR, K_{session}(error, nonce + 1, TS')$

3.4.2. Diseño e implementación

Dentro de este apartado debemos separar el código fuente de la aplicación y el C&C:

3.4.2.1. MRATServer

Dentro del código del C&C nos encontramos los siguientes ficheros fuentes:

Main.java Es el punto de inicio del programa, en este fichero, se crea una instancia una clase `Server` y se procesa la entrada de texto. Una vez procesada la orden se ejecutará el método correspondiente del C&C.

Server.java Se define la clase `Server`, para poder instanciar una clase se llamará al constructor pasándolo por parámetro el número de puerto en el que va a escuchar. Además esta clase contiene los siguientes métodos:

- **Start():** se arranca el servidor, es decir el `Server Socket` definido en el constructor espera la conexión de una aplicación. Una vez establecida la conexión y con el id de la aplicación obtenido en el método `getId()` se genera una clave de sesión que será usada para cifrar todo el protocolo de mensajes. Esta clave se enviará en el método `sendKey()`.
- **sendKey():** usando el id de la aplicación conectada, se buscará la clave asociada a ese id para cifrar el primer mensaje. Una vez obtenida esta clave, se cifrará y se enviará un mensaje para que la aplicación tenga la clave de sesión.
- **GetAppsInstalled():** se construye y se envía el mensaje de petición de las aplicaciones instaladas en el dispositivo, y se devuelve un array de `String` con las aplicaciones si se han podido obtener o `null`.
- **GetScreenshot():** se construye y se envía el mensaje de petición de captura de pantalla y se devuelve un array con los bytes de la captura de pantalla.
- **GetContact():** se construye y se envía el mensaje de petición de contactos y se devuelve un array con los `Strings` de los nombres y números de los contactos.
- **GetInfo():** se construye y se envía el mensaje de petición de información y se devuelve un `String` con la información general del dispositivo.

- **GetImages():** se construye un mensaje de petición cada una de las imágenes que obtenemos al llamar al método **GetImagesPath()**, posteriormente se envían estos mensajes al C&C recibiendo cada imagen solicitada.
- **GetImagesPath()** se genera y se envía al C&C un mensaje solicitando una lista con todas las imágenes que contiene.
- **Close():** en este método se cierra el Serversocket y con esto se finaliza el servidor.

Apps.java Este código contiene la definición de la clase Apps, para instanciarla se debe llamar al constructor con el array de Strings que contiene el nombre de las aplicaciones instaladas. La clase sólo contiene el método **toString()**.

Contacts.java Al igual que la clase Apps, se instancia llamando al constructor con un array de array de Strings con los contactos, sólo contiene el método **toString()**.

Screenshot.java Se define el código de la clase Screenshot, para instanciarla hay que llamar al constructor con un array de bytes. Los métodos que contiene esta clase son: **show()** para mostrar la imagen en pantalla y **save(String filename)**. que guarda la imagen en el path definido en filename.

Keys.java En la clase Keys están almacenadas las distintas claves asociadas a los id's de cada aplicación, además contiene los siguientes métodos:

- **getKey(int id):** dado el id de la aplicación se retorna el valor de su clave en un array de bytes.
- **generateKey():** usando la clase `SecureRandom` del API de Java se genera una secuencia de bytes pseudoaleatorios que son criptográficamente seguros. Con estos bytes se genera la clave AES de 256 bits y se devuelve en un array de bytes.
- **generateNonce():** se genera un nonce con números aleatorios y se devuelve como int.

3.4.2.2. FlashLight.apk

La aplicación Android contiene unos ficheros descriptivos en formato `.xml`, los cuales se describen a continuación:

AndroidManifest.xml Este fichero es el punto de partida de toda aplicación, en él se especifica toda la información que necesita Android para identificar las acciones que se pueden realizar gracias a los permisos que se declaren en él.

Si una aplicación requiere realizar alguna acción especial sobre el dispositivo como usar la cámara, leer los contactos, leer la memoria externa, etcétera, debe definirse en este fichero. En nuestro caso se van a solicitar los siguientes:

- **Acceso a Internet.** Para poder tener acceso a internet y poder realizar conexiones TCP/IP.
- **Lectura/Escritura en almacenamiento externo.** Para poder acceder a la memoria externa del dispositivo.
- **Lectura de contactos.** Para obtener la lista de contactos del dispositivo.
- **Lectura del estado del teléfono.** Gracias a este permiso podemos obtener la información general del dispositivo.
- **Acceso a localización.** Aceptando este permiso la aplicación puede tener acceso al GPS para obtener la ubicación.
- **Acceso a Registro de llamadas.** Para obtener el registro de llamadas del dispositivo.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.READ_CALENDAR"/>
```

Figura 3.7: Declaración de permisos en el fichero AndroidManifest.xml

activity_main.xml En el fichero `activity_main.xml` se define cual es el diseño visual de la aplicación, además nos dice que widgets hay en el interfaz de usuario y como están situados.

En este caso como la aplicación simula ser una linterna, este fichero sólo tiene definido un botón en el centro de la pantalla que al pulsarse hará que empiece a funcionar al programa.

A continuación se definen el código fuente en Java de la aplicación:

MainActivity.Java `MainActivity` es la clase que está definida en `AndroidManifest.xml` para que se ejecute cuando se arranque la aplicación.

El código que incluye muestra el contenido del fichero de layout `activity_main.xml` y registra un botón que al ser pulsado inicia la clase `MainLogic`.

MainLogic.java En esta clase se comprueba, si se puede hacer uso de la vulnerabilidad Certifi-Gate y se instancia una clase `Client` con la IP y puerto donde se está ejecutando el C&C.

Client.java Esta clase contiene toda la lógica del MRAT, en ella se responde a las peticiones del servidor y se obtiene la información que se le solicita. Para instanciar una clase `Client` se debe llamar al constructor con la IP y puerto del C&C, y el contexto de `MainLogic`.

Al llamar al constructor se invoca al método **sendID()** y después al método **run()**, estos métodos y todos los que contiene la clase se detallan a continuación:

- **sendID():** en este método se envía el primer mensaje de conexión con el C&C, en el cual se le indica el ID de la aplicación para que el C&C le envíe un mensaje que contiene la clave de sesión, con la que irán cifrados todos los mensajes, cifrado con la clave correspondiente a ese ID.
- **run():** en este método se reciben y se procesan todos los mensajes enviados por el C&C. Cuando se reciba un mensaje se filtrará y se llamará a los métodos correspondientes para obtener y enviar la información que se solicita en el mensaje.
- **saveKey(Mensaje.TSendKey t):** se extrae la clave de sesión con la que se cifrarán todos los mensajes entre la aplicación y el C&C.

- **sendContacts(Mensaje.TGetContact t):** se envía al C&C los nombres y números que hay en la libreta de contactos del dispositivo.
- **getNumContacts():** este método devuelve un entero con el número de contactos que tenemos en el dispositivo.
- **getContacts(String[] names, String[] numbers):** se devuelve en los arrays de String names y numbers el contenido de los contactos que tenemos en el dispositivo.
- **sendInstalledApps(Mensaje.TGetInstalledApps t):** se envía al C&C un listado con todas las aplicaciones que están instaladas en el dispositivo.
- **getNumInstalledApps():** este método devuelve un entero con el número de aplicaciones instaladas en el dispositivo.
- **getInstalledApps():** se retorna un array de String con los nombres de las aplicaciones que hay instaladas.
- **sendScreenshot(Mensaje.TScreenshot t):** en este método se envía al C&C una captura de pantalla del dispositivo.
- **captureScreen():** el método devuelve una array de bytes con el contenido de la captura de pantalla.
- **sendInfo(Mensaje.TGetInfo t):** se responde al C&C con la información del dispositivo.
- **getInfo():** se devuelve un String con toda la información del dispositivo.
- **sendListImages(Mensaje.TListImages t):** se responde a la petición del C&C con una lista con todos los nombres de las imágenes que almacena el dispositivo.
- **getPathImages():** se retorna un array de String con los nombres de todas las imágenes contenidas en el dispositivo.
- **sendImage():** se responde al C&C con la imagen que solicita en su petición.
- **readImage(String imagePath):** se lee la imagen que tiene el nombre que se pasa como parámetro en imagePath y se devuelve como un array de bytes.

3.4.2.3. Clases comunes

Además de las clases que se han descrito anteriormente existen dos clases que son comunes al la aplicación y el C&C, dichas clases son:

CipherAES.java Esta clase contiene la implementación del cifrado/descifrado AES usado en el C&C. Para instanciar una clase de este tipo se debe llamar al constructor pasando por parámetro una clave en un array de bytes.

CipherAES tiene dos métodos:

- **cipherInGCMMode(byte[] plaintextBytes):** se cifra el texto en claro que se pasa en el array de bytes plaintextBytes y se devuelve en un array de bytes.
- **decipherInGCMMode(byte[] ciphertextBytesWithNonce):** eel método complementario al anterior, se descifra el array de bytes contenido en ciphertextBytesWithNonce y se retorna en un array de bytes.

Mensaje.java La clase Mensaje contiene el código de todos los tipos de mensajes que existen para la comunicación entre la aplicación y el C&C. Los mensajes están serializados de manera que el protocolo permite que el servidor pueda estar desarrollado en cualquier lenguaje de programación, sólo se debe tener en cuenta cómo está compuesto cada mensaje para poder leerlo correctamente. Dentro de esta clases tenemos una clase interna por cada tipo de mensaje que extienden la clase Mensaje.

Las clases que contienen la implementación de los mensajes del protocolo son:

- **TGetContact:** mensaje de petición de los contactos.
- **RGetContact:** mensaje de respuesta con los contactos del dispositivo.
- **TGetInstalledApps:** mensaje de petición de las aplicaciones instaladas.
- **RGetInstalledApps:** mensaje de respuesta con las aplicaciones instaladas en el dispositivo.

- **TGetInfo:** mensaje de petición de la información del dispositivo.
- **RGetInfo:** mensaje de respuesta con la información del dispositivo.
- **SendKey:** mensaje que contiene la clave de cifrado de la comunicación entre C&C y aplicación.
- **SendID:** mensaje que contiene el ID de la aplicación.
- **TScreenshot:** mensaje de petición de una screenshot.
- **RScreenshot:** mensaje de respuesta con la screenshot realizada.
- **ROk:** mensaje para confirmar que una operación anterior se ha realizado correctamente.
- **RErr:** mensaje de respuesta a una petición que no se ha podido tramitar correctamente.
- **TListImages:** mensaje de petición de los nombres de las imágenes que contiene el dispositivo.
- **RListImages:** mensaje de respuesta con los nombres de las imágenes del dispositivo.
- **TImage:** mensaje de petición de una imagen.
- **RImage:** mensaje de respuesta con la imagen solicitada.

3.4.3. Explotación de Certifi-gate

Como se ha definido en la introducción, Certifi-gate aprovecha las distintas vulnerabilidades que existen en el plug-in del MRST para comprobar si la aplicación que intenta hacer uso de las acciones privilegiadas es legítima. Para este proyecto vamos a explotar la vulnerabilidad que tiene el plug-in de la aplicación de control remoto TeamViewer [10].

La aplicación TeamViewer al arrancar se comunica con el plug-in para identificarse, este carga el certificado de la aplicación que se identifica y verifica que el número de serie es el mismo que el que tiene codificado.

```

.method static constructor <clinit>()V
    .registers 2
    00000000 new-instance        v0, BigInteger
    00000004 const-string        v1, "1287658381"
    00000008 invoke-direct      BigInteger-><init>(String)V, v0, v1
    0000000E sput-object        v0, TVAddonService->serialNum_v:BigInteger
    00000012 return-void
.end method

.method private checkCallerCertSerialMatch__v(String)Z
    .registers 4
    00000000 invoke-virtual      TVAddonService->getApplicationContext()Context, p0
    00000006 move-result-object  v0
    00000008 invoke-static      certMgr_v->return_caller_cert__v(String, Context)X509Certificate, p1, v0
    0000000E move-result-object  v0
    00000010 invoke-virtual      X509Certificate->getSerialNumber()BigInteger, v0
    00000016 move-result-object  v0
    00000018 sget-object        v1, TVAddonService->serialNum_v:BigInteger
    0000001C invoke-virtual      BigInteger->equals(Object)Z, v0, v1
    00000022 move-result        v0
    00000024 if-eqz            v0, :2C
    :28
    00000028 const/4            v0, 1
    :2A
    0000002A return        v0
    :2C
    0000002C const-string      v0, "TVAddonService"
    00000030 const-string      v1, "checkSignature(): serial mismatch - onBind will fail"
    00000034 invoke-static  Logging->a(String, String)V, v0, v1
    0000003A const/4            v0, 0
    0000003C goto        :2A
.end method

```

Figura 3.8: Vulnerabilidad en el plug-in de TeamViewer

En Android, cada desarrollador puede generar su certificado autofirmado para firmar una aplicación. Esto permite que el desarrollador decida cual es el número de serie del certificado.

Por tanto, en este caso hemos generado un certificado X.509 con el mismo número de serie que el plug-in tiene codificado. De esta manera la aplicación se hará pasar por una aplicación legítima y podrá tener acceso a los permisos privilegiado. En el caso de la aplicación desarrollada, para poder obtener capturas de pantalla.

3.5. Funcionamiento de la aplicación

En este apartado se va a detallar el funcionamiento de la aplicación. Primero se verá como se instala y arranca la aplicación en el dispositivo y posteriormente el control desde el C&C.

3.5.1. Instalación y ejecución de la aplicación “troyanizada”

En esta simulación se ha enviado un correo electrónico al usuario atacado con la app maliciosa, tal y como vimos en el apartado de contramedidas de la introducción, no se deben instalar aplicaciones que no sean de la tienda oficial del sistema. Si el usuario no se descarga la aplicación del correo no podría ser infectado.

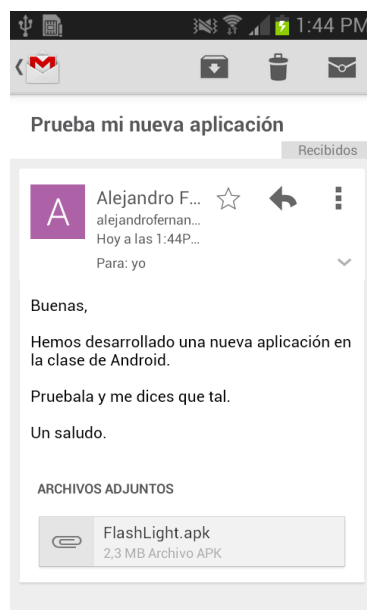


Figura 3.9: Correo electrónico recibido por la víctima

Una vez descarga la aplicación el usuario procederá a instalar el fichero .apk al instalarlo se solicitarán los permisos. En este punto también si el usuario hubiera seguido los consejos vistos no hubiera sido afectado ya que para una aplicación que simula ser un flash se solicitan muchos más permisos de los necesarios.

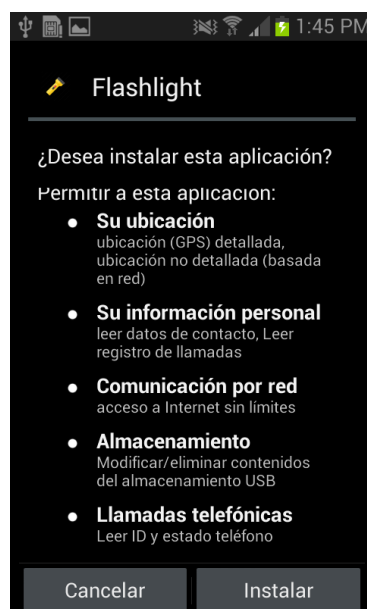


Figura 3.10: Permisos solicitados por la aplicación maliciosa

Cuando se haya finalizado la instalación el usuario la ejecutará y pulsará el botón para

comprobar si funciona, llegados a este punto empezará a correr un servicio en background a la espera de las peticiones del C&C.

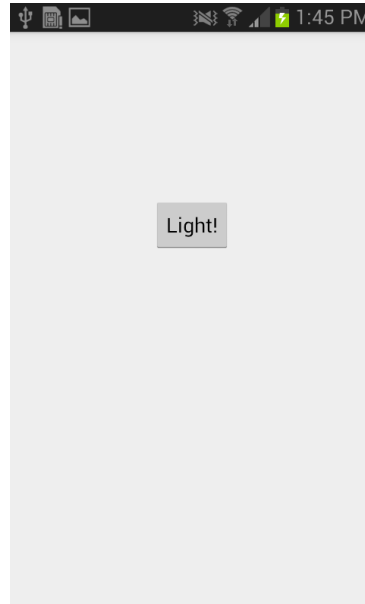


Figura 3.11: Aspecto visual de la aplicación FlashLight

3.5.2. Funcionamiento del C&C

El C&C desarrollado tiene un interfaz que permite dar instrucciones por línea de comandos (CLI).

Antes de que el usuario atacado reciba la aplicación el atacante estará ejecutando el C&C, el C&C al ejecutarse estará esperando a que alguna víctima instale la aplicación.

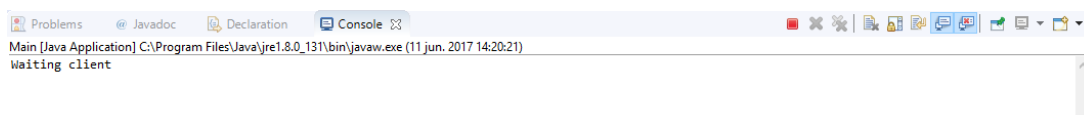


Figura 3.12: C&C a la espera de la conexión de la aplicación

El C&C escribirá por pantalla el ID de la aplicación que se ha conectado y después esperará las órdenes que se deseen ejecutar.

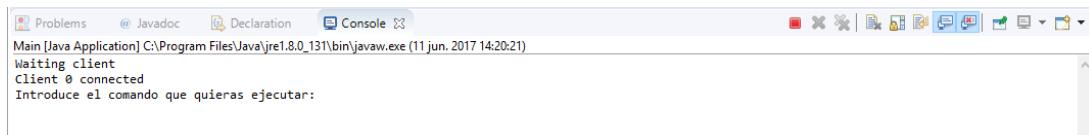


Figura 3.13: Aplicación conectada al C&C

3.5.2.1. Obtención de contactos

Para obtener los contactos se debe escribir la orden **contacts**, acto seguido se recibirán en el C&C todos los contactos que tenga el dispositivo atacado.

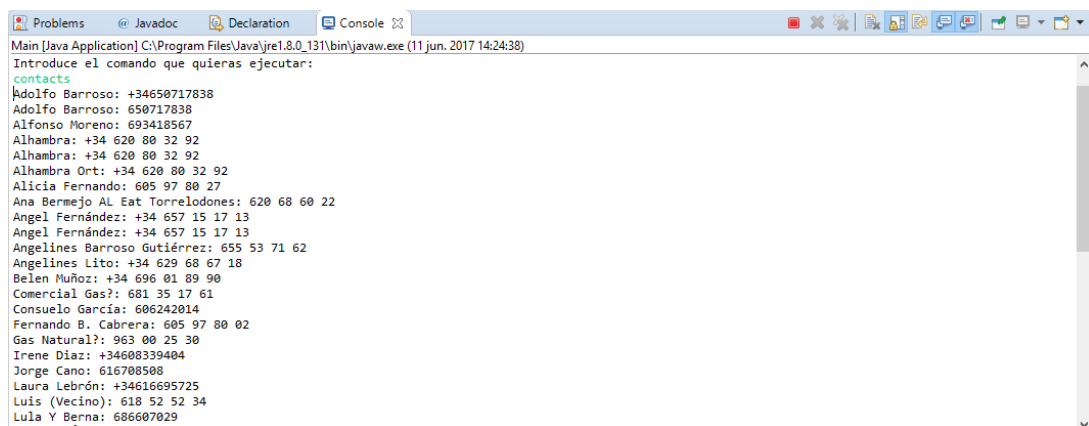


Figura 3.14: Ejecución de la orden contacts

3.5.2.2. Obtención de aplicaciones instaladas

Si se introduce la orden **apps** podremos observar todas las aplicaciones que el dispositivo tiene instaladas.

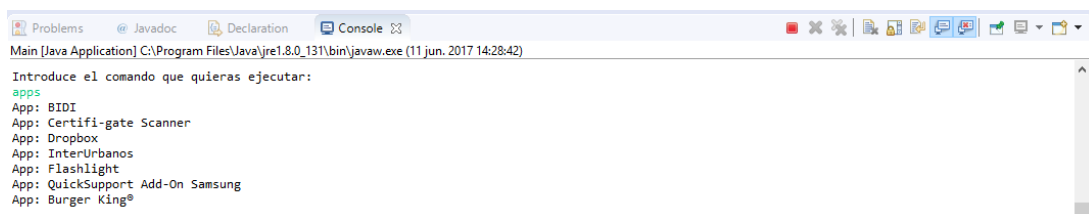


Figura 3.15: Ejecución de la orden apps

3.5.2.3. Obtención de información del dispositivo

Para saber la información general del dispositivo se debe introducir la orden **info** en el C&C.



Figura 3.16: Ejecución de la orden info

3.5.2.4. Obtención de captura de pantalla

Si se quiere obtener una captura de pantalla del dispositivo se introducirá la orden **screenshot** y se mostrará en la pantalla del C&C la pantalla que tenga el dispositivo una vez que se le demanda la orden.

Gracias a la vulnerabilidad de Certifi-gate vista en la subsección 1.2.1 y su posterior explotación podemos obtener dichas capturas, ya que el sistema Android no permite a una aplicación desarrollada por terceros acceder al permiso de obtener la pantalla.

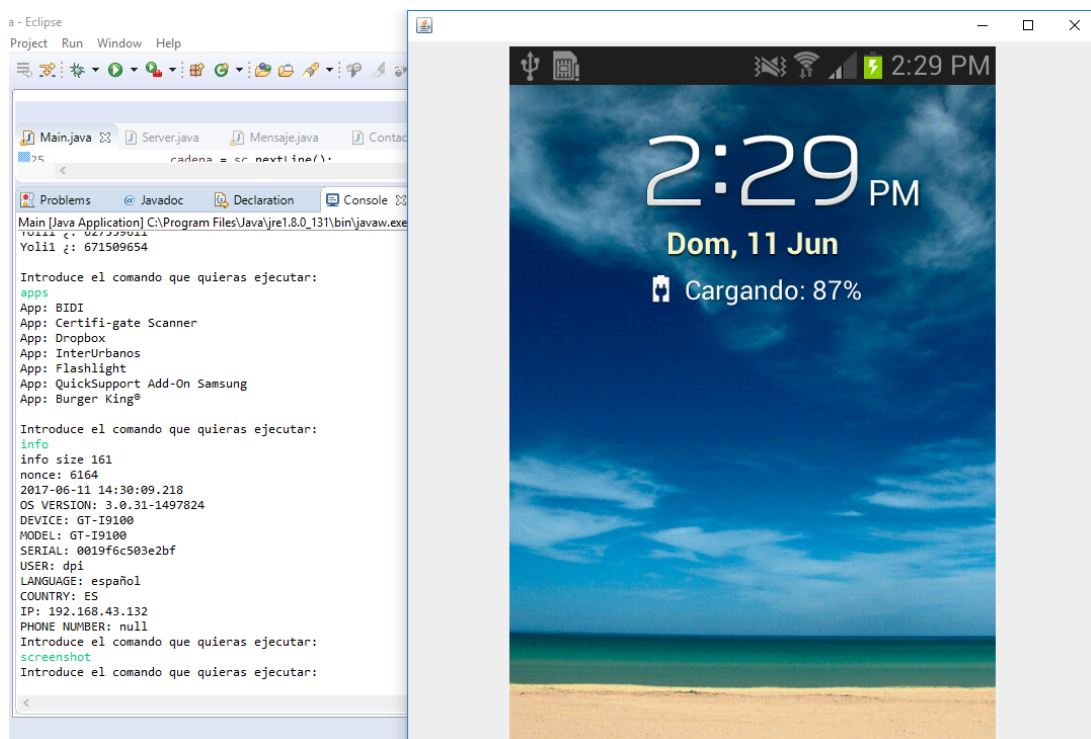


Figura 3.17: Ejecución de la orden screenshot

3.5.2.5. Obtención de las imágenes del dispositivo

Por último al introducir por comando **images** se mostrarán por pantalla todas las imágenes que tenga el dispositivo atacado.

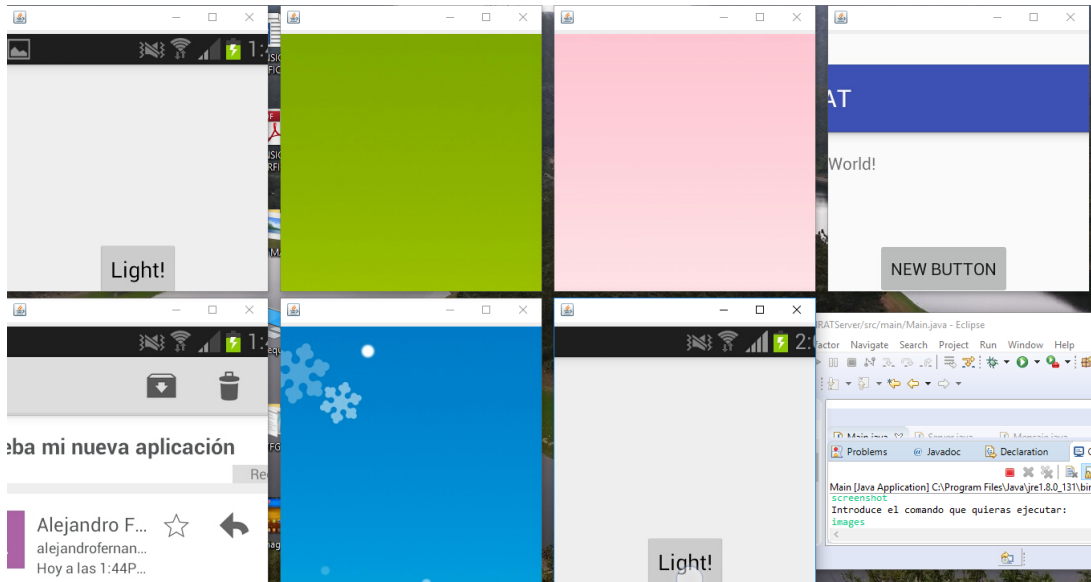


Figura 3.18: Ejecución de la orden images

Capítulo 4

Conclusiones

En esta memoria se ha definido el proceso para reproducir un ataque MRAT desarrollando una aplicación maliciosa y un C&C. En este apartado se va a analizar la consecución de los objetivos propuestos. También se listarán las competencias del grado en las cuales se ha apoyado el proyecto y las adquiridas en el desarrollo del mismo. Y por último se redactarán unas ideas sobre cómo se podría mejorar el proyecto desarrollado.

4.1. Consecución de objetivos

En este apartado se va a analizar si se cumplen los objetivos definidos al inicio. Para ellos se enumerará cada objetivo y se hará una valoración.

- **Desarrollo de una aplicación para Android que ejecute un troyano en background.**

Este objetivo se ha cumplido ya que como se comento en el apartado de desarrollo de la aplicación al pulsar el boton LIGHT de la aplicación empieza a correr un servicio en background que no es detectado por el usuario.

- **Explotación de la vulnerabilidad Certifi-gate, detallada en la subsección 1.2.1, con el fin de acceder a operaciones provilegiadas, como obtener capturas de pantalla de otras aplicaciones.** En este caso el proyecto también ha cumplido con este objetivo, ya que se ha conseguido usar la vulnerabilidad para poder realizar capturas de pantalla.

- **Implementación de un C&C que permita controlar remotamente el dispositivo atacado.** Desde el C&C se podrán realizar distintas peticiones que serán respondidas

por el troyano. Este objetivo se ha cumplido con éxito como se ha documentado durante el capítulo de desarrollo de la aplicación.

- **Conexión reversa desde la aplicación “troyanizada”.** El troyano debe conectarse al C&C automáticamente (“llamar a casa”), con el objetivo de traspasar los cortafuegos que protegen la red en la que se encuentra el dispositivo afectado. El cortafuegos interno suele ser menos restrictivo que el externo. Nuestra aplicación al arrancar se conecta al C&C y se identifica con un ID, por tanto este objetivo también se ha cumplido.
- **Obtención de la información del dispositivo.** El C&C debe de ser capaz de obtener la siguiente información del dispositivo:
 1. **Capturas de pantalla.**
 2. **Aplicaciones instaladas.**
 3. **Contactos.**
 4. **Información del dispositivo.**
 5. **Imágenes de galería.**

En el apartado Funcionamiento de la aplicación se ha realizado una prueba de concepto en la cual se puede comprobar que hemos obtenido toda la información que se requería en este objetivo. Por tanto podemos dar este objetivo como cumplido.

- **Desarrollo de un protocolo de comunicación basado en llamadas a procedimiento remoto (RPC) y no dependiente de un lenguaje de programación o una plataforma concreta.** Como se ha explicado en el apartado de código fuente, la clase Mensaje está serializada de tal manera que se permite que el C&C esté desarrollado en cualquier lenguaje de programación. Por lo que este último objetivo también se ha cumplido.
- **Cifrado de la comunicación.** El intercambio de información entre la aplicación y el C&C debe ser cifrado, de manera que si se está observando el tráfico de la red no se pueda entender la comunicación entre C&C y aplicación. En este caso el objetivo se ha cumplido con éxito, ya que además de construir un protocolo de comunicación cifrado, se ha desarrollado de tal manera que el protocolo no puede ser comprometido gracias a

las marcas de tiempo y a los números aleatorios que se incluyen por cada mensaje. El protocolo está definido en el apartado protocolo de comunicación 3.4.1.

4.2. Competencias utilizadas y adquiridas

Durante el grado se han adquirido unas competencias que han servido de apoyo a la hora de realizar este proyecto, asignaturas como Ampliación de Sistemas Telemáticos y Laboratorio de Sistemas Móviles y Ubicuos han sido fundamentales ya que el proyecto se ha desarrollado en los lenguajes aprendido en dichos cursos. Otro pilar fundamental en el proyecto es la asignatura de Seguridad en las Redes de Ordenadores puesto que en ella se aprenden los conocimientos de seguridad y cifrados aplicados en el proyecto.

Por otra parte durante el desarrollo de este proyecto se han adquirido otros conocimientos como pueden ser la simulación de ataque a una vulnerabilidad, el uso de bibliotecas de cifrado o la documentación de un proyecto extenso.

4.3. Trabajos futuros

A continuación se dan una serie de mejoras que se podrían incluir en el proyecto para mejorarlo:

- **Implementación de pulsaciones.** Usando la vulnerabilidad de Certifi-Gate es posible realizar pulsaciones sobre la pantalla en remoto. Esta utilidad mejoraría notablemente el proyecto ya que remotamente se podría manejar la pantalla del dispositivo.
- **Obtención de capturas de pantalla en tiempo real.** Obteniendo la captura de pantalla y con la posibilidad de realizar pulsaciones sobre la misma se podría tener un control total sobre el dispositivo atacado.
- **Simulación de un interprete de comandos.** El C&C podría simular un interprete de comandos (CLI) para controlar el dispositivo infectado, y que permita moverse entre las distintas carpetas y obtener los distintos archivos.
- **Creación de una interfaz visual para el C&C.** Actualmente el interfaz del C&C es una

consola en la cual se introducen cadena de texto, una interfaz visual haría el proyecto más atractivo y fácil de usar.

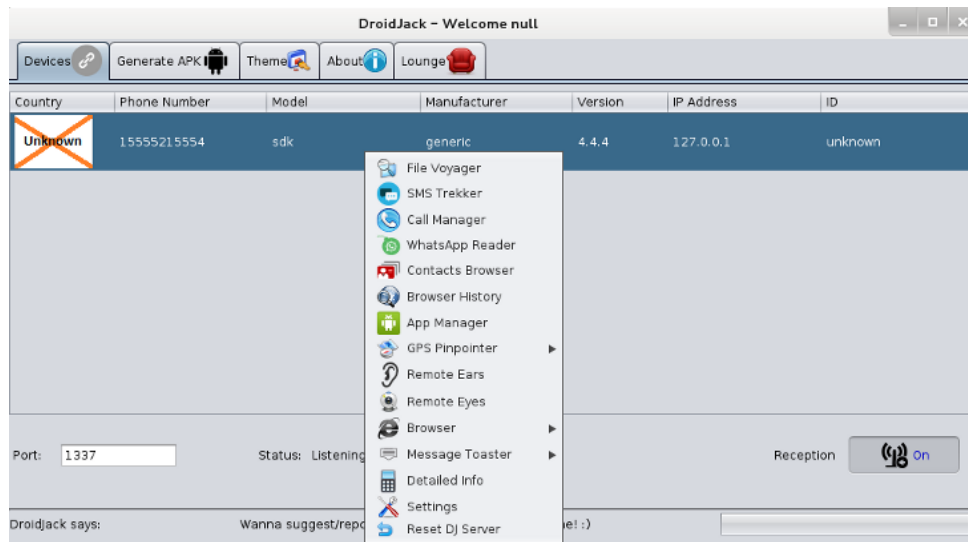


Figura 4.1: Interfaz visual del troyano DroidJack

Bibliografía

- [1] EL CASO RONNIE (ATAQUE DoS: DENEGACIÓN DE SERVICIO DISTRIBUIDOS)

<http://www.eoi.es/blogs/ciberseguridad/2016/04/19/el-caso-ronnie-ataque-dos-denegacion-de-servicio-distribuidos/>

- [2] RAMSOM.WANNACRY, *Symantec Security Response*

https://www.symantec.com/security_response/writeup.jsp?docid=2017-051310-3522-99

- [3] WIN32/CONFICKER

<https://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Win32%2fConficker>

- [4] TROJAN.ZBOT, *Ben Nahorney and Nicolas Falliere*

https://www.symantec.com/security_response/writeup.jsp?docid=2010-011016-3514-99

- [5] DARKCOMET

<http://www.darkcomet-rat.com/>

- [6] SMASH THE STACK

<http://insecure.org/stf/smashstack.html>

- [7] ZERODIUM

<http://zerodium.com/>

- [8] CERTIFI-GATE: HUNDREDS OF MILLIONS OF ANDROID DEVICES COULD BE PWNED,
Check Point Research Team

<http://blog.checkpoint.com/2015/08/06/certifigate/>

- [9] INSTAGRAM

<https://www.instagram.com/>

- [10] TEAMVIEWER

<https://www.teamviewer.com>

- [11] RSUPPORT

<http://www.rsupport.com>

- [12] GOOGLE PLAY

<https://play.google.com>

- [13] APTOIDE

<https://es.aptoide.com/>

- [14] A SPIRAL MODEL OF SOFTWARE DEVELOPMENT AND ENHANCEMENT, *Boehm, B, 1986, ACM*

<http://dl.acm.org/citation.cfm?doid=12944.12948>

- [15] ANDROID STUDIO

<https://developer.android.com/studio/index.html>

- [16] JAVA

<https://www.java.com>

- [17] ECLIPSE

<https://eclipse.org/>

- [18] BOUNCY CASTLE

<https://www.bouncycastle.org/>

[19] SPONGY CASTLE

<https://rtyley.github.io/spongycastle/>

[20] ADVANCED ENCRYPTION STANDARD, *Miller, Frederic P. and Vandome, Agnes F. and McBrewster, John, 2009, Alpha Press*

[http://searchsecurity.techtarget.com/definition/
Advanced-Encryption-Standard](http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard)

[21] GALOIS COUNTER MODE, *David A. McGrew and John Viega*

[http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/
proposedmodes/gcm/gcm-spec.pdf](http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf)

Apéndice A

URL de descarga

La URL del repositorio en el cual se encuentra tanto el código del C&C como la aplicación Android es: <https://github.com/afernandezb92/MRAT>