**UNIVERSIDAD POLITÉCNICA DE MADRID**

****

**MÁSTER UNIVERSITARIO EN CIBERSEGURIDAD**

**TRABAJO FIN DE MÁSTER**

***Desarrollo de mecanismos para auditoría de tráfico de aplicaciones Android***

**Andrea del Nido García**

**2020**

**TRABAJO FIN DE MÁSTER**

**TÍTULO:** Desarrollo de mecanismos para auditoría de tráfico de aplicaciones Android

**AUTOR:** D. Andrea del Nido García

**TUTOR 1/DIRECTOR:** D. José María del Álamo Ramiro

**Escuela Técnica Superior de Ingenieros de Telecomunicación**

**Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación**

**Escuela Técnica Superior de Ingenieros Informáticos**

**Escuela Técnica Superior de Ingeniería de Sistemas Informáticos**

**TRIBUNAL:**

**PRESIDENTE:**

**VOCAL:**

**SECRETARIO:**

**FECHA DE LECTURA:**

**CALIFICACIÓN:**

Fdo: El Secretario del Tribunal

Agradecimientos

Opcional

Resumen

Cada día más y más usuarios disponen de teléfono móviles, y hoy en día se asocia a las personas con números de teléfono. Todo este gran desarrollo de tecnología conlleva a que, como bien es sabido por todos, nuestros móviles sean algo fundamental en todo momento. La mayor parte de datos personales de los usuarios se encuentran en sus dispositivos móviles. Es por esto por lo que es realmente necesario definir unos protocolos de seguridad claros y concisos, que permitan poder hacer uso de estas tecnologías sin temer ataques a la privacidad, robo de identidad, etc.

Cada vez hay más y más aplicaciones disponibles en nuestros teléfonos móviles que hacen que nuestra vida sea algo más sencilla o incluso más entretenida en ciertos momentos. Ahora bien, es cierto que dichas aplicaciones hacen uso de datos que se almacenan en los terminales, como, por ejemplo, posicionamiento GPS, acceso a cámara, contactos, entre otros. Solicitando permiso al usuario claro, pero, ¿quién va a negar el permiso de la cámara a Candy Crash?, si no lo aceptas entonces no te será permitido hacer uso del juego.

Con el trabajo se quiere conocer lo que hacen las aplicaciones móviles con los datos de los usuarios. Analizando los mecanismos que emplean las aplicaciones para transmitir información a servidores u otros sistemas, se descubre que las comunicaciones de las aplicaciones van cifradas por lo que, realmente no es posible conocer que datos de los usuarios están trasmitiendo y a quién los están transmitiendo. La solución que se plantea en el trabajo, consiste en desarrollar mecanismos que faciliten la auditoría de tráfico cifrado en un entorno de laboratorio.

Abstract

Every day more and more users have mobile phones, and nowadays people are associated with phone numbers. All this great development of technology means that, as is well known by all, our mobiles are essential at all times. Most of the personal data of users is found on their mobile devices. This is why it is really necessary to define clear and concise security protocols that allow you to make use of these technologies without fear of attacks on privacy, identity theft, etc.

There are more and more applications available on our mobile phones that make our lives a little easier or even more entertaining at certain times. However, it is true that these applications make use of data that is stored in the terminals, such as, for example, GPS positioning, camera access, contacts, among others. Requesting permission from the user of course, but who is going to deny the permission of the camera to Candy Crash? If you do not accept it then, you will not be allowed to use the game.

With the project we want to know what mobile applications do with user data. Analyzing the mechanisms that applications use to transmit information to servers or other systems, it is discovered that the communications of the applications are encrypted so that it is not really possible to know what user data they are transmitting and to whom they are transmitting it. The solution proposed in the work consists of developing mechanisms that facilitate the audit of encrypted traffic in a laboratory environment.

Contenido

[Agradecimientos 2](#_Toc40202055)

[Resumen 3](#_Toc40202056)

[Abstract 4](#_Toc40202057)

[Contenido 5](#_Toc40202058)

[Listado de figuras 7](#_Toc40202059)

[Listado de tablas 8](#_Toc40202060)

[Acrónimos 9](#_Toc40202061)

[Definiciones previas 10](#_Toc40202062)

[1 Introducción 13](#_Toc40202063)

[1.1 Introducción 13](#_Toc40202064)

[1.2 Objetivos 14](#_Toc40202065)

[2 Estado del arte 17](#_Toc40202066)

[2.1 Introducción 17](#_Toc40202067)

[2.2 Certificados 18](#_Toc40202068)

[2.3 Comunicación segura: SSL/TLS 20](#_Toc40202069)

[2.4 Mecanismos de seguridad de comunicación en Android 22](#_Toc40202070)

[2.4.1 Fijación de certificados 22](#_Toc40202071)

[2.5 Captura de tráfico 25](#_Toc40202072)

[2.6 Análisis de las aplicaciones Android 26](#_Toc40202073)

[3 Desarrollo 28](#_Toc40202074)

[3.1 Introducción 28](#_Toc40202075)

[3.2 Herramientas 29](#_Toc40202076)

[3.3 Entorno 31](#_Toc40202077)

[3.3.1 Sistemas operativos y despliegue 31](#_Toc40202078)

[3.3.2 Componentes de las máquinas 33](#_Toc40202079)

[3.3.2.1. Proxy de intercepción 33](#_Toc40202080)

[3.3.2.2. Entorno de ejecución 33](#_Toc40202081)

[3.4 Método de identificación de bibliotecas de fijación de certificados 34](#_Toc40202082)

[3.4.1 Requisitos 36](#_Toc40202083)

[3.4.2 Ejecución 36](#_Toc40202084)

[3.4.3 Salida 38](#_Toc40202085)

[3.5 Método de inyección de código 39](#_Toc40202086)

[3.5.1 Caso 1: Inyección de código con nuevas librerías 40](#_Toc40202087)

[3.5.2 Caso 2: Inyección de código nuevo de librerías ya conocidas (APKs simples)…. 40](#_Toc40202088)

[3.5.3 Caso 3: Inyección de código nuevo de librerías ya conocidas (Bundle APKs)……. 40](#_Toc40202089)

[3.6 Método de ejecución de múltiples APKs 41](#_Toc40202090)

[4 Resultados 42](#_Toc40202091)

[4.1 Introducción 42](#_Toc40202092)

[4.2 Análisis resultados inyección de código 42](#_Toc40202093)

[4.3 Gestión de registros 46](#_Toc40202094)

[5 Conclusiones y líneas futuras 49](#_Toc40202095)

[Apéndices y/o anexos 50](#_Toc40202096)

[Bibliografía 51](#_Toc40202097)

Listado de figuras

[Figura 2.1. Diagrama de la cadena de confianza 17](C:\\Users\\Andrea\\Desktop\\Andrea\\Máster Ciberseguridad UPM\\TFM\\Memoria\\Memoria_V2_TFM_MUCS_Andrea_del_Nido.docx" \l "_Toc40179343)

[Figura 2.2. Esquema de funcionamiento certificate pinning 19](C:\\Users\\Andrea\\Desktop\\Andrea\\Máster Ciberseguridad UPM\\TFM\\Memoria\\Memoria_V2_TFM_MUCS_Andrea_del_Nido.docx" \l "_Toc40179344)

[Figura 2.3. Esquema de funcionamiento mitmproxy modo transparente HTTP 22](C:\\Users\\Andrea\\Desktop\\Andrea\\Máster Ciberseguridad UPM\\TFM\\Memoria\\Memoria_V2_TFM_MUCS_Andrea_del_Nido.docx" \l "_Toc40179345)

[Figura 3.1. Esquema del entorno de trabajo 28](C:\\Users\\Andrea\\Desktop\\Andrea\\Máster Ciberseguridad UPM\\TFM\\Memoria\\Memoria_V2_TFM_MUCS_Andrea_del_Nido.docx" \l "_Toc40179346)

[Figura 3.2. Esquema del sistema de búsqueda de bibliotecas con certificate pinning 30](C:\\Users\\Andrea\\Desktop\\Andrea\\Máster Ciberseguridad UPM\\TFM\\Memoria\\Memoria_V2_TFM_MUCS_Andrea_del_Nido.docx" \l "_Toc40179347)

[Figura 3.3. Ejecución del script pinning\_libraries.sh 31](C:\\Users\\Andrea\\Desktop\\Andrea\\Máster Ciberseguridad UPM\\TFM\\Memoria\\Memoria_V2_TFM_MUCS_Andrea_del_Nido.docx" \l "_Toc40179348)

[Figura 3.4. Ejecución del script find\_libraries\_pinning.sh 32](C:\\Users\\Andrea\\Desktop\\Andrea\\Máster Ciberseguridad UPM\\TFM\\Memoria\\Memoria_V2_TFM_MUCS_Andrea_del_Nido.docx" \l "_Toc40179349)

[Figura 3.5. Ejecución del script check\_pinning\_library.py 32](C:\\Users\\Andrea\\Desktop\\Andrea\\Máster Ciberseguridad UPM\\TFM\\Memoria\\Memoria_V2_TFM_MUCS_Andrea_del_Nido.docx" \l "_Toc40179350)

[Figura 3.6. Contenido de uno de los ficheros json 33](C:\\Users\\Andrea\\Desktop\\Andrea\\Máster Ciberseguridad UPM\\TFM\\Memoria\\Memoria_V2_TFM_MUCS_Andrea_del_Nido.docx" \l "_Toc40179351)

[Figura 3.7. Contenido de un fichero CSV generado por el sistema 34](C:\\Users\\Andrea\\Desktop\\Andrea\\Máster Ciberseguridad UPM\\TFM\\Memoria\\Memoria_V2_TFM_MUCS_Andrea_del_Nido.docx" \l "_Toc40179352)

**¡Error! Referencia de hipervínculo no válida.¡Error! Referencia de hipervínculo no válida.¡Error! Referencia de hipervínculo no válida.¡Error! Referencia de hipervínculo no válida.¡Error! Referencia de hipervínculo no válida.¡Error! Referencia de hipervínculo no válida.**

Listado de tablas

[Tabla 2.1. Librerías de fijación de certificados con métodos característicos 23](#_Toc40550988)

[Tabla 3.1. Resumen de librerías previas y de las librerías nuevas 45](#_Toc40550989)

[Tabla 4.1. Resumen de las aplicaciones analizadas 48](#_Toc40550990)

**¡Error! Referencia de hipervínculo no válida.**Acrónimos

A continuación, se listan los acrónimos empleados a lo largo del desarrollo del trabajo.

|  |  |
| --- | --- |
| **HTTP** | Hypertext Transfer Protocol |
| **SSL** | Secure Socket Layer |
| **TLS** | Transport Layer Security |
| **HTTPS** | Hypertext Transfer Protocol Secure |
| **DHCP** | Dynamic Host Configuration Protocol |
| **CA** | Certification Authority |
| **PKI** | Public Key Infrastructure |
| **MiTM** | Man in The Middle |
| **APK** | Android Package |

Definiciones previas

Se resumen a continuación las definiciones de conceptos utilizados a lo largo del trabajo desarrollado.

**Integridad:** este principio básico de la seguridad de la información versa sobre que los datos que son transmitidos no sean modificados durante el trayecto.

**Confidencialidad:** principio de la seguridad de la información que consiste en que, si se interceptan los datos o el mensaje, sea ininteligible, de manera que únicamente los verdaderos integrantes de la comunicación puedan comprender el contenido del mensaje.

**Autenticidad:** este principio consiste en que, al recibir información de alguna de las partes de la comunicación, ésta sea enviada realmente por ese integrante y no por otra entidad que no pertenece a la comunicación.

**Modelo cliente-servidor:** este tipo de arquitectura se basa en que los servicios y recursos de un determinado software son proporcionados por un servidor y, aquellas entidades que demandan estos servicios y recursos son los clientes.

**HTTP:** es un protocolo de comunicación que permite la transferencia de datos e información comúnmente utilizado en la World Wide Web (www). En la capa de transporte es soportado por el protocolo TCP y en el nivel de red por el IP. Este protocolo sigue un modelo cliente-servidor. El cliente realiza peticiones al servidor (request) y el servidor proporciona recursos y servicios contestando mediante una respuesta (response). Este protocolo tiene definidos varios métodos para poder realizar las peticiones y en las respuestas se incluyen códigos para verificar el estado de la comunicación. El puerto estándar para este protocolo es el 80.

**SSL/TLS:** SSL es un protocolo predecesor del protocolo TLS. Estos protocolos criptográficos consisten en proporcionar confidencialidad e integridad en las comunicaciones en una red de comunicación. Tratan de garantizar que la información que es transmitida por un canal no sea modificada ni interceptada y, que los integrantes de la comunicación puedan acceder a ella. Tal y como se muestra en [1], estos protocolos son muy utilizados para la seguridad de las comunicaciones web, pero para la situación de este proyecto, hay que tener en cuenta que para dispositivos Android se utilizan estos protocolos para la comunicación cifrada.

**HTTPS:** protocolo HTTP pero que utiliza el cifrado del protocolo SSL/TLS en las comunicaciones.El protocolo funciona igual que HTTP, pero la diferencia es que al cifrar las comunicaciones ya no se pueden realizar ataques estilo MiTM ya que al interceptar las tramas HTTPS estarían cifradas.

**DHCP:** por sus siglas en inglés, *Dynamic Host Configuration Protocol* [2], es un protocolo de red cuya arquitectura de red es el modelo cliente servidor, el cual tiene como función asignar direcciones IP a dispositivos de forma dinámica, es decir, sin que la dirección IP tenga que ser fijada por un usuario. Para ello, se dispone de una lista de direcciones DHCP que se van asignando en base al uso de cada una de ellas. Además, se almacenan ciertos parámetros de configuración necesarios para poder llevar a cabo de forma satisfactoria esta asignación de direcciones, algunos parámetros almacenados son, por ejemplo, la máquina a la que se asigna la dirección, la IP asignada, la duración de la asignación de dicha dirección, entre otros parámetros.

**Bundle APK:**

Smali

# Introducción

## Introducción

Cada vez más y más la población hace uso de terminales móviles. De hecho, habitualmente es más común que en las familias cada uno de los miembros disponga de un dispositivo móvil y no, por ejemplo, ordenadores de sobremesa o portátiles. En estos dispositivos se genera, recolecta y almacena grandes cantidades de información personal del usuario (y de sus contactos, personas próximas) que puede ser consultada y utilizada por las aplicaciones del móvil. Según el artículo “*Las aplicaciones que deberían preocuparte son estas y no FaceApp*” de Belén Afonso [3], un estudio sobre 36 aplicaciones populares sobre salud demostró que 33 de ellas transmitían datos privados de los usuarios y 29 vendían los datos a empresas como Google o Facebook. Con esta información, el lector claramente puede darse cuenta de que los datos privados de los usuarios que son recogidos por las aplicaciones pueden acabar siendo utilizados al margen de su control.

Además de lo comentado anteriormente, es necesario tener en cuenta la existencia de las redes públicas Wi-Fi, en las cuales cualquiera puede conectarse y hacer uso de sus servicios y, además, las redes de uso común como pueden ser las de las universidades o centros de trabajo, etc. Tener todo esto en cuenta supone darse cuenta de que algún individuo podría ser capaz de interceptar el tráfico de los diferentes terminales conectados a una red. Si este atacante consiguiera interceptar tráfico, esto significaría que es capaz de obtener en claro los mensajes y sería capaz de obtener datos que ofrecen las aplicaciones. Tal y como se comenta en la primera parte de esta introducción, no se puede saber realmente que hacen las aplicaciones con los datos de los usuarios ya que van cifradas y protegidos con mecanismos de seguridad. Es decir, una aplicación puede estar enviando datos personales a una empresa sin que nos percatemos de ello.

En el Departamento de Ingeniería de Sistemas Telemáticos (DIT) se ha desarrollado una plataforma que permite auditar aplicaciones móviles para conocer sus características de privacidad y protección de datos personales. El trabajo se inició con los Trabajos Fin de Master de Esperanza Zamora [4] y Lucas María Tomé [5]. Continuó con el Trabajo Fin de Grado de Sergio Framiñán García [6]. Entre las capacidades de la plataforma desarrollada, se encuentra el poder interceptar las comunicaciones que establecen aplicaciones móviles para poder determinar si contienen información personal de los usuarios. Ahora mismo, con los mecanismos ya implementados en la plataforma, se puede hacer incluso cuando las aplicaciones cifran su tráfico y, en algunos casos, se protegen con mecanismos de fijación de certificados. Sin embargo, se han detectado otras situaciones en las que no se ha podido romper el cifrado por utilizar mecanismos avanzados o no conocidos de fijación de certificados. Por ello, el trabajo presente trabajo tiene como objetivo contribuir al avance de esta línea, para poder aumentar los casos en los que se puedan inspeccionar las comunicaciones, y con ello tener más certeza sobre el comportamiento de las aplicaciones respecto al tratamiento de datos personales.

## Objetivos

El principal objetivo de este Trabajo Fin de Master consiste en primer lugar, investigar y estudiar los diferentes mecanismos de implementación más utilizados y comunes de protección de comunicaciones, como Certificate Pinning, ya que estos serán los que aparezcan en la gran mayoría de aplicaciones Android, además, después de este estudio, se diseñarán soluciones que permitan sobrepasar dichos mecanismos mediante el empleo de técnicas de instrumentación dinámica de las aplicaciones. A continuación, se listas los objetivos más concretos.

* Estudiar y comprender los diferentes ataques que se pueden producir a las aplicaciones Android mediante el tráfico de red.
* Analizar las posibles soluciones ya aplicadas a los ataques producidos.
* Estudio del Certificate Pinning, su funcionamiento e implementación en las diferentes aplicaciones de sistemas Android.
* Análisis y estudio de las herramientas más comunes para poder realizar ataques de tráfico en aplicaciones Android.
* Estudio e implantación del entorno de trabajo.
* Selección y estudio de herramientas que se puedan utilizar para sobrepasar el mecanismo de seguridad Certificate Pinning de Android.
* Analizar diferentes aplicaciones que dispongan de soluciones Certificate Pinning mediante técnicas de instrumentación dinámica para conseguir romper esta seguridad y, tratar de encontrar patrones para automatizar los ataques.
* Diseño de posibles soluciones para poder sobrepasar el mecanismo de seguridad de las aplicaciones Android basado en el método de Certificate Pinning.
* Pruebas del correcto funcionamiento de los métodos o soluciones implementadas.
* Análisis de los resultados obtenidos frente a anteriores propuestas de trabajo sobre auditoría de tráfico a las aplicaciones Android.
* Documentación de todo el trabajo realizado.

# Estado del arte

## Introducción

A lo largo de este capítulo se explicarán los estudios teóricos que se han llevado a cabo para poder desarrollar de forma efectiva el proyecto. En la primera parte, se expondrá el funcionamiento de la estructura de certificados ya que es objeto de estudio para el proyecto. Se trata de conseguir romper la fijación de certificados por lo que, se considera necesario ahondar en el concepto de la estructura de los mismos.

A continuación, se desarrollará la noción de las comunicaciones seguras, concretamente el estudio se focalizará en la arquitectura TLS/SSL. Es necesario desarrollar estos conceptos ya que, para el trabajo, será necesario evitar estos mecanismos de seguridad.

Posteriormente, el estudio de las comunicaciones seguras se centra en el sistema Android, ya que el proyecto se desarrolla sobre éste. Particularmente, se hace especial insistencia en el mecanismo de fijación de certificados. Se analiza su funcionamiento y cómo se puede implementar en las aplicaciones Android.

Consecutivamente, se comenta cómo se puede conseguir capturar tráfico entre dos sistemas diferentes. Finalmente, se explica de qué manera se puede analizar las aplicaciones Android, específicamente su código. Se indica cómo se puede obtener un APK y a partir de éste, adquirir el código de la aplicación y estudiarlo.

## Certificados

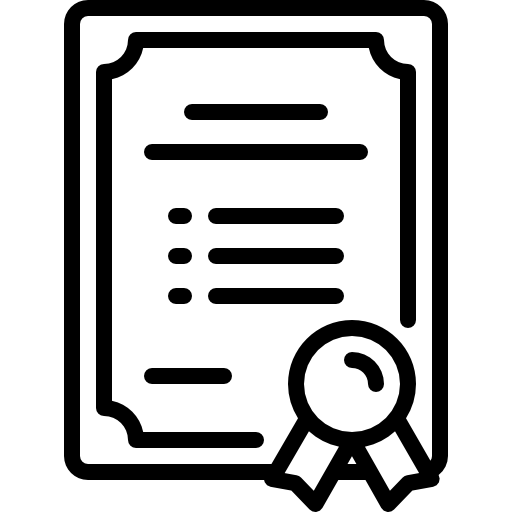
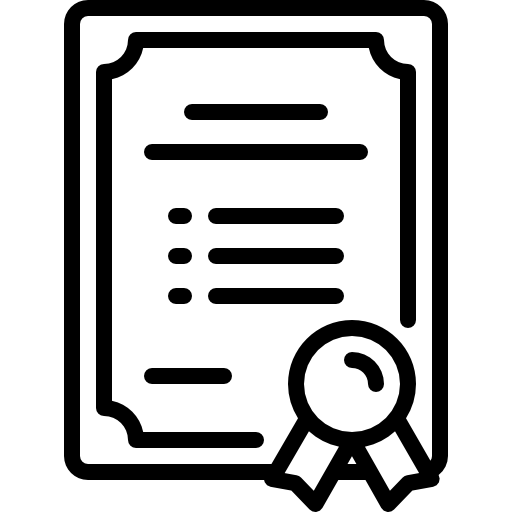
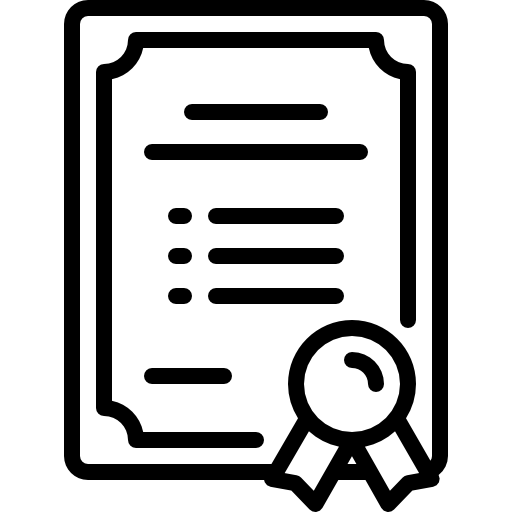
Los certificados son mecanismos esenciales para poder garantizar la autenticidad de una entidad y, además, el no repudio. Los certificados se pueden encontrar comúnmente en navegadores web, pero también se emplean para el acceso a máquinas mediante SSH), de tal forma que se requiere de un certificado específico para acceder a los recursos, consiguiendo diseñar mecanismos de control de acceso.

Los certificados tienen una gran importancia ya que, por ejemplo, en el caso que ocupa este estudio, si nos centramos en el uso de los certificados de las aplicaciones Android; se emplean para poder identificar la aplicación y el servidor de manera unívoca. Si se utiliza un certificado diferente en la aplicación, éste sería rechazado por el servidor y no se terminará al instante la comunicación entre ellos.

A continuación, se indican los distintos tipos de certificados:

* **Certificado autofirmado**: se genera por una entidad que no es una autoridad certificadora. Debido a esto, no se puede utilizar para autenticar la identidad real ya que, no es respaldado por ninguna autoridad.
* **Certificado raíz**: este tipo de certificado es generado por una CA (*Certification Authority*), en la cual se puede confiar. A partir de este certificado fiable, se pueden verificar otros certificados y firmarlos para así, validar la identidad. Cabe destacar que los certificados raíz son certificados autofirmados, ya que dependen directamente de las CA y no hay certificados por encima de ellos.
* **Certificado subordinado**: son certificados reconocidos por una o más CA y, además, son capaces de emitir otros certificados a terceras partes.
* **Certificado externo**: este tipo de certificado es reconocido por otro certificado subordinado de una CA y, además, incluye la firma encriptada con la clave privada del poseedor del certificado. Esto es necesario para poder establecer conexiones seguras.
* **Certificado final**: este certificado hace referencia a aquellos que se ubican en entidades finales, tales como páginas web o ciertos dominios de los que ya no cuelgan más certificados.

Para poder establecer la confiabilidad entre todos los certificados, se construyen las cadenas de confianza. Estas cadenas, establecen las distintas relaciones entre certificados y la jerarquía de los mismos. De esta manera, la cadena comienza con un certificado raíz expedido por una Autoridad de Certificación y ésta a su vez, valida certificados intermedios hasta llegar al certificado final. En el siguiente diagrama [7] se muestra dicha estructura.



Nombre emisor (CA)

**Certificado Intermedio**

Firma emisor

Clave pública propietario

Propietario

autofirma

Firma de la CA raíz

Clave pública propietario

Propietario

referencia

Clave pública CA raíz

Nombre CA raíz

referencia

Nombre emisor (CA)

**Certificado Raíz**

Firma emisor

firma

**Certificado Final**

firma

Figura 2.1. Diagrama de la cadena de confianza de los certificados

Como se observa en el esquema anterior, los certificados deben incluir cierta información como la fecha de creación, la autoridad que validó el certificado, la organización a la que corresponde, el dominio del certificado. Además de esa información, es necesario que el certificado incluya una firma ya que con ella se podrá validar la autoridad del certificado. Para este proceso de validación de la firma, se requiere de la clave pública del propietario del certificado.

## Comunicación segura: SSL/TLS

Los protocolos SSL (Secure Sockets Layer) y TLS (Transport Layer Security) [8] permiten a dos entidades (arquitectura cliente/servidor) que se identifiquen y autentiquen recíprocamente y así, consiguen comunicarse con confidencialidad e integridad de la información que transmiten. El protocolo SSL es la evolución del protocolo TLS, cabe destacar que no interoperan entre sí. Estos protocolos proporcionan seguridad de las comunicaciones mediante Internet, logran que la comunicación entre cliente y servidor sea segura, mediante la confidencialidad, confiabilidad e integridad. Seguidamente, se identifican y explican los pasos [9] de ambos protocolos para establecer la comunicación entre el cliente y el servidor.

1. El cliente envía un mensaje “*client* *handshake*” que incluye información sobre los algoritmos criptográficos (CipherSuite) que se pueden usar y la versión del protocolo SSL o TLS. También se incluye una ristra de bytes aleatorios que se utilizarán para posteriores operaciones y, opcionalmente métodos de compresión de datos.
2. El servidor responde con un mensaje tipo “*server* *handshake*” que indica el conjunto de algoritmos de cifrado que admite de la lista que proporcionó el saludo del cliente, un identificador de la sesión y otra cadena de bytes aleatoria. En este mensaje, el servidor envía también su certificado, para que el cliente pueda verificar la autoría. Opcionalmente, puede enviar una solicitud de certificado del cliente si es que dispone de uno.
3. El cliente verifica el certificado enviado por el servidor.
4. El cliente envía al servidor la cadena de bytes aleatoria a partir de la cual se calcula la clave secreta, tanto el cliente como el servidor, que les permitirá cifrar los mensajes y así conseguir confidencialidad. Esta ristra de bytes está cifrada con la clave pública de servidor.
5. Este apartado es opcional ya que, solamente se produce si resulta que el cliente dispone de un certificado y el servidor envió la solicitud. El cliente envía un conjunto de bytes aleatoria cifrada con la clave privada del propio cliente, junto con su certificado.
6. El servidor verifica el certificado del cliente, si es que dispone de él y fue solicitado por el servidor.
7. El cliente envía un mensaje de finalización al servidor, el cual se cifra con la clave secreta, demostrando que la parte de la comunicación del cliente está completada.
8. El servidor envía otro mensaje de finalización al cliente que también se cifra con la clave secreta, demostrando que esta parte de la comunicación del servidor está completada.
9. Finalmente, el cliente y el servidor ya pueden intercambiar mensajes protegidos con cifrado simétrico de forma segura.

En la figura que se muestra a continuación, se representa de forma gráfica el proceso redactado anteriormente. En la figura se hace uso de la abreviatura OP para indicar que es un paso opcional, tal y como se explica anteriormente.

Saludo del cliente

Información criptográfica

1

2

Saludo del servidor

6

3

CipherSuite y certificado servidor

Verificar certificado cliente (OP)

Verificar certificado servidor y parámetros criptográficos

4

Solicitud certificado cliente (OP)

Intercambio de claves

Enviar certificado cliente

5

Finalización cliente

7

Finalización servidor

8

9

Intercambio de mensajes

**Cliente SSL**

**Servidor SSL**

Figura 2.2. Descripción gráfica del protocolo SSL/TLS

## Mecanismos de seguridad de comunicación en Android

En esta sección del trabajo se explican los principales mecanismos que se utilizan en los dispositivos móviles para poder comunicarse entre ellos, ya sea para transferencia de ficheros o comunicaciones entre las aplicaciones que residen en el terminal con sus respectivos servidores para solicitar recursos.

Este apartado se divide en los diferentes mecanismos de transferencia y comunicación y mecanismos de seguridad que se utilizan para proteger el tráfico de datos entre dispositivos y servidores. Se explicarán los siguientes: comunicación TLS/SSL, certificate pinning (objeto principal de estudio de este trabajo), PONER MÁS

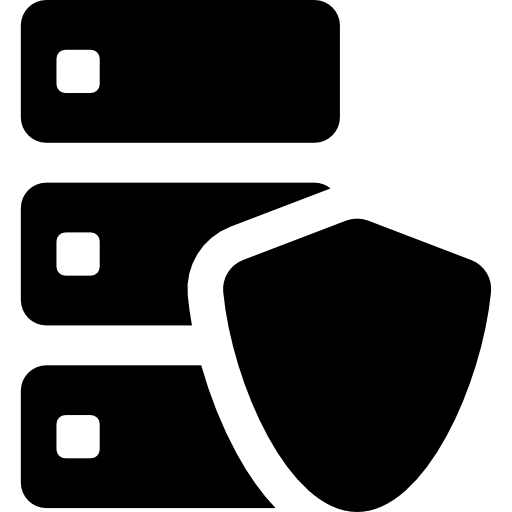
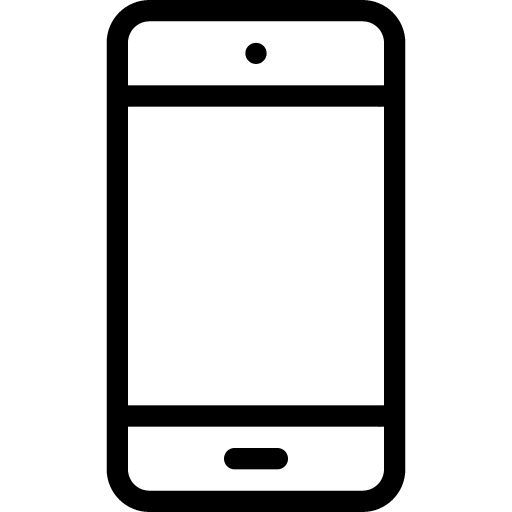
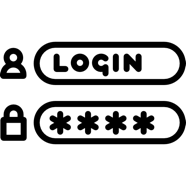
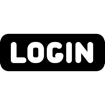
### Fijación de certificados

El certificate pinning [10] o traducido, fijación de certificados, es un mecanismo a partir del cual se relaciona un host con su respectiva clave pública de tal manera que trata de conseguir solucionar o mitigar el problema que se produce en la verificación de las cadenas de certificados. Por lo general, la confianza en los certificados SSL se reduce simplemente a los certificados raíz, es decir, se escalan certificados hasta alcanzar el que es fiable (certificado raíz). Estos certificados vienen por defecto en el sistema operativo, navegadores y demás dispositivos, pero el problema realmente llega a la hora de verificar el certificado del propio dispositivo. Únicamente se hacen comprobaciones de los certificados del servidor.

La fijación de certificados se refiere al proceso por el cual se importan certificados en un dispositivo o máquina host en su depósito de certificados de confianza de manera que, no es necesario confiar en los certificados de las CA (Autoridades de Certificación), es decir, mitiga el riesgo de que las CA se vean comprometidas.

En el caso de Android, las aplicaciones llevan incluidos desde el código de la propia aplicación una copia del certificado de tal manera que, se compara este certificado anclado con el certificado del servidor, para así evitar problemas con la validez de las CA y de los certificados raíz. Ahora bien, un problema evidente es que es necesario actualizar los certificados manualmente al caducar.

En la figura que se muestra a continuación, se representa en la parte derecha un terminal móvil Android que dispone de aplicaciones. Como se indica, dicha aplicación en el terminal contiene un certificado que corresponde al del servidor con el que se conecta el dispositivo móvil. El primer paso, es el “saludo” por parte del terminal indicando que quiere establece una comunicación con el servidor, esta fase (1), se conoce como *SSL/TLS Handshake*. Una vez producido el saludo, el servidor envía su certificado al terminal, fase (2) y este, verifica que corresponde con el certificado que tiene almacenado mediante el uso de la clave, fase (3). Cuando ya se ha comprobado la validez del certificado, entonces ya puede realizarse una comunicación entre el servidor y el terminal móvil, fase (4).



SSL/TLS Handshake

4

**Terminal móvil**

Verificación de la clave fijada

Envío de certificado

1

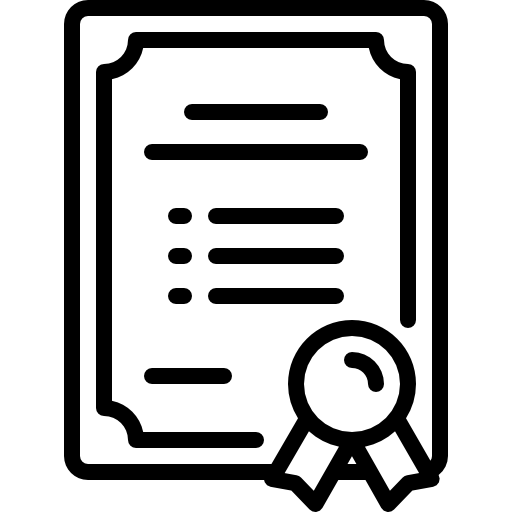
**Servidor**

2

3

**App**

Comienzo de la comunicación



**Certificado**

Figura 2.3. Esquema de funcionamiento certificate pinning

Una de las maneras más comunes de implementar la fijación de certificados es mediante el uso de métodos de diferentes librerías diseñadas concretamente para esta función. Es por ello que, en la siguiente tabla se recogen las librerías [11] más comunes con los métodos más significativos que resuelven el problema de la fijación de certificados en las aplicaciones Android.

|  |  |
| --- | --- |
| **Librería** | **Métodos** |
| TrustManager | X509TrustManager |
| SSLContext |
| TrustManagerFactory |
| KeyStore |
| TrustManagerImpl | TrustManagerImpl |
| verifyChain |
| Android WebViewClient | WebViewClient |
| onReceivedSslError |
| SslErrorHandler |
| OpenSSLSocketImpl Conscrypt | OpenSSLSocketImpl |
| verifyCertificateChain |
| OpenSSLEngineSocketImpl Conscrypt | OpenSSLEngineSocketImpl |
| verifyCertificateChain |
| OpenSSLSocketImpl Apache | OpenSSLSocketImpl |
| verifyCertificateChain |
| OkHTTPv3 | CertificatePinner |
| Certificate |
| Trustkit | OkHostnameVerifier |
| SSLSession |
| X509Certificate |
| PinningTrustManager |
| Appcelerator Titanium | PinningTrustManager |
| CheckServerTrusted |
| PhoneGap sslCertificateChecker | sslCertificateChecker |
| execute |
| IBM MobileFirst | WLClient |
| pinTrustedCertificatePublicKey |
| IBM WorkLight | HostNameVerifierWithCertificatePinning |
| verify |
| Conscrypt CertPinManager | CertPinManager |
| isChainValid |
| CWAC-Netsecurity | CertPinManager |
| isChainValid |
| Worklight Androidgap | WLCertificatePinningPlugin |
| execute |
| Netty FingerprintTrustManagerFactory | FingerprintTrustManagerFactory |
| checkTrusted |
| Squareup CertificatePinner | CertificatePinner |
| Check |
| Certificate |
| Squareup OkHostnameVerifier | OkHostnameVerifier |
| verify |
| X509Certificate |
| Apache Cordova WebView | CordovaWebViewClient |
| onReceivedSslError |
| SslErrorHandler |
| Boye AbstractVerifier | AbstractVerifier |
| verify |

Tabla 2.1. Librerías de fijación de certificados con métodos característicos

## Captura de tráfico

El tráfico que se quiere interceptar de las aplicaciones Android, son peticiones o consultas que se realizan entre la propia aplicación y servidores. Para que estas consultas se puedan llevar a cabo, es necesario disponer de una conexión a Internet típicamente, por lo que se utilizará este medio para conseguir capturar el tráfico.

La manera más habitual de interceptar tráfico entre dos sistemas es mediante el uso de un proxy, de tal forma que esté ubicado entre medias de los dos sistemas y así, lograr interceptar las peticiones que se realizan.

Existen muchas herramientas con funcionalidad de proxy. Las más comunes son Burp Suite [12] y mitmproxy [13]. La principal diferencia entre ellas es que Burp Suit utiliza un formato de interfaz gráfica, mientras que, mitmproxy es un proxy que funciona mediante línea de comandos.

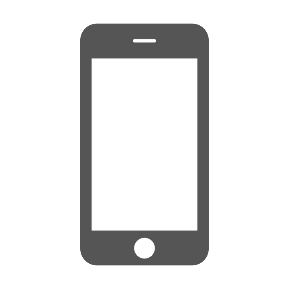
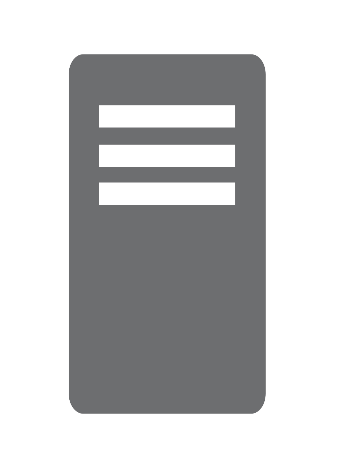
Para este proyecto se ha utilizado la herramienta mitmproxy ya que los módulos del proyecto requieren de interconexión mediante comandos. Además, como se utilizan diferentes herramientas para acceder al terminal móvil con comandos y se implementa código que se inyectará con la ayuda de la herramienta Frida, se ha decidido utilizar mitmproxy, ya que, para el desarrollo de este proyecto, es mejor solución que Burp Suite. Mencionar también que mitmproxy ha desarrollado una API y una UI Web que facilita la visualización de los mensajes interceptados.

A continuación, se describe el funcionamiento de mitmproxy ya que es el que se utilizará a lo largo del desarrollo del trabajo. Destacar que este proxy dispone de diferentes funcionalidades y opciones, pero para el desarrollo de este proyecto, las opciones que se han utilizado son mitmdump y mitmproxy con el modo transparente. Este modo transparente, simplemente hace referencia a que, al usar el proxy, la conexión se redirige directamente a un proxy en la capa de red, esto lógicamente provoca que no sea necesario realizar configuraciones en el cliente; por ello resulta realmente interesante para analizar aplicaciones Android, debido a que no se puede acceder al cliente. Mitmproxy es capaz de interceptar tanto en comunicaciones HTTP como HTTPS. El diagrama que se muestra a continuación, representa el proceso del modo transparente con peticiones HTTP del proxy explicado previamente. Como se aprecia, el cliente envía una petición al servidor, pero esta es interrumpida por mitmproxy, una vez redirigida la petición, se envía al servidor. Si se requiere de una conexión HTTPS, es necesario comenzar con los mensajes propios de TLS/SSL, es decir, el handshake y sus respectivas respuestas.

1

Conexión

Router



4

2

Redirección

Petición

mitmproxy

3

**Cliente**

Petición HTTP

**Servidor**

**Transparente**

Figura 2.4. Esquema de funcionamiento mitmproxy modo transparente HTTP

## Análisis de las aplicaciones Android

El análisis de las aplicaciones Android es un proceso largo y complejo. Durante el desarrollo de este trabajo, se ha tenido que analizar gran cantidad de código de diferentes APKs para poder tratar de comprender el funcionamiento de los mecanismos de seguridad que implementan los distintos sistemas de fijación de certificado. Se ha tratado de simplificar esta tarea de búsqueda de métodos que implementen certificate pinning, mediante el desarrollo de sistemas como los que se explican en la sección tres de este mismo documento, aunque sigue siendo una tarea complicada.

Principalmente, hay cuatro problemas a la hora de tratar de analizar código de aplicaciones Android. El primero de ellos es que un APK ya está compilado, es decir, no podemos acceder al código. El segundo de ellos es que suponiendo que hemos conseguido acceder al código, éste estará en el lenguaje Smali, a priori puede parecer que esto no es un problema, pero es un lenguaje complicado si no se ha estudiado nunca; se puede obtener el código Java de la aplicación, pero es muy probable que se pierda información. El tercer problema es el gran tamaño y la complejidad de las múltiples clases que se implementan, resulta muy tedioso analizar y buscar en cada método. Finalmente, hay que mencionar que muchas aplicaciones aplican técnicas de ofuscación de código para dificultar aún más la tarea de identificación de clases y métodos relevantes.

Para facilitar la resolución de los problemas citados anteriormente, a continuación, se proponen métodos y herramientas para solventarlo. Lo primero de todo es que, si se quieren obtener APKs, la forma más fácil de obtenerlos es mediante páginas como “*APK Pure”* [14], simplemente se indica el nombre de la aplicación a partir de la cuál se quiere obtener el APK y se descarga. Una vez hecho esto, para poder resolver el primer problema indicado, es decir, para poder obtener el código del APK, se utilizan herramientas como *Apktool* [15], además esta misma herramienta permite obtener el .jar (contiene las clases y métodos en java) del código smali extraído. En cuanto a la complejidad y extensión del código, se ha desarrollado un sistema, el cual se explica en la siguiente sección, que consiste en revisar todo el código previamente obtenido, y buscar algún indicio de que se están utilizando librerías y métodos concretos, en lo que ocupa a este trabajo, se centran en la búsqueda de mecanismos de certificate pinning. En cuanto a la ofuscación del código [16], es un proceso complejo que trata de acortar los nombres de clases, métodos y variables, logrando que sea muy complicado entender el código. Este proceso de ofuscación actualmente se realiza mediante ProGuard y para desofuscar el código es necesario disponer de un archivo específico llamado mapping.txt.

El análisis explicado anteriormente, hace referencia al análisis general del código de una aplicación, cómo obtener ese código y cómo poder estudiarlo de forma más sencilla. Por otro lado, se distinguen dos tipos de análisis en la ingeniería inversa. El análisis estático, que estudia el comportamiento del código sin que se esté ejecutando y por otro lado, el análisis dinámico, que observa el comportamiento del código de un programa mientras se está ejecutando.

# Desarrollo

## Introducción

A lo largo de este capítulo se expone de qué manera se logra desarrollar diferentes mecanismos para la auditoría de tráfico en aplicaciones Android. Se divide en diferentes secciones. La primera de ellas, comenta las herramientas que se han utilizado a lo largo del desarrollo para poder completar los objetivos.

En el segundo apartado se explica el entorno de trabajo en el que se implementan los mecanismos para la auditoría de tráfico.

En la siguiente sección, se detalla un sistema diseñado para poder identificar el uso de bibliotecas concretas de fijación de certificados, ya que el código de las aplicaciones es muy extenso y complejo de analizar.

Seguidamente, el quinto apartado incluye una guía detallada de los diferentes casos o situaciones que se han encontrado al analizar las aplicaciones, con el objeto de implementar nuevos mecanismos que rompan la fijación de certificados. En cada caso se explica cómo son las modificaciones que se han llevado a cabo para evitar el elemento de seguridad mencionado previamente.

En el último subcapítulo se analiza un componente desarrollado el cual se utiliza para poder procesar varias APKs simultáneamente sin tener que ir ejecutando cada una de ellas de forma individual. Es necesario esta implementación para poder ahorrar tiempo y, además, conseguir analizar muchas más aplicaciones rápidamente.

A continuación, se muestra el flujo con diferentes etapas aproximadas que se han seguido a lo largo del desarrollo del proyecto.

Diagrama del flujo de trabajo seguido

## Herramientas

En este apartado se explican las herramientas que se han utilizado para poder desarrollar la implementación de las soluciones sobre la fijación de certificado en las aplicaciones Android.

* **VirtualBox**: es un software de código libre cuya funcionalidad es la de ser un host hipervisor [17], es decir, permite crear múltiples máquinas virtuales de diferentes sistemas operativos sin tener que crear particiones o modificar el propio sistema de la máquina anfitrión. Para el trabajo desarrollado, ha sido necesario su uso para poder crear dos máquinas virtuales con sistemas operativos Parrot y Android x86, que permitan realizar el estudio, la implementación y pruebas necesarias sobre las soluciones a certificate pinning.
* **Parrot**: es un sistema operativo basado en la distribución Debian GNU de Linux [18]. Este sistema operativo se utiliza como proxy de intercepción (servidor MiTM), a partir del cual se obtendrá la imagen de Docker (en la que se encuentra parte del trabajo previo) y con lo que se trabajará para desarrollar los mecanismos para lograr romper la fijación de certificados. Se ha escogido este sistema operativo debido a que cuenta con un gran número de herramientas de seguridad y, está especialmente diseñado para labores de seguridad informática.
* **Android x86**:sistema basado en la imagen de un terminal Android [19] de forma que se puede emular dicho dispositivo. Se utilizará para realizar las correspondientes pruebas de funcionamiento del bypass de la fijación de certificados, mediante la instalación de forma remota (desde la máquina Parrot) de aplicaciones.
* **Docker**:es un software de código abierto que trata de agilizar y automatizar el lanzamiento de aplicaciones en contenedores [20], lo que provoca que se cree una capa más de virtualización. En este proyecto se utilizará ya que, al continuar con proyectos anteriores, se almacena toda la plataforma en un contenedor para facilitar su despliegue. Además de utilizar el contenedor que ya está propuesto, los cambios que se realicen serán modificados en el contendor.
* **Adb**:Android Debug Bridge,es una herramienta de línea de comandos la cual permite la comunicación con terminales [21]. Algunas de las acciones que se pueden realizar con esta herramienta son, por ejemplo, acceder al terminal, modificar aplicaciones (instalar y desinstalar APKs), eliminar ficheros, transferencia de ficheros entre la máquina conectada y el dispositivo Android.
* **Apktool**:herramienta que se utiliza para descompilar y así obtener el código de la aplicación Android [15]. El código que se obtiene a partir de la aplicación está desarrollado en lenguaje Smali, si se quiere obtener el código de la aplicación en leguaje Java, serán necesarias otras herramientas. En el caso de este trabajo, se utilizará Apktool para acceder al código de determinadas aplicaciones Android para estudiarlo y modificarlo y así, lograr realizar el bypass de certificate pinning.
* **Mitmproxy**:esta herramienta de línea de comandos es un proxy que se utiliza para interceptar tráfico de las comunicaciones entre diferentes dispositivos [13].Además, dispone de interfaz web y de API las cuales facilitan la visualización del tráfico. En este proyecto se hará uso de esta herramienta para poder interceptar las comunicaciones entre la máquina base (Parrot) y el terminal Android x86.
* **Frida**:es una herramienta de instrumentación de aplicaciones, [22] mediante la cual se pueden inyectar scripts en distintos sistemas, lanzar hooks (ganchos) contra APIs para lograr acceder sin necesidad de tener las credenciales, entre otras funciones.En este proyecto se utilizará Frida para interceptar llamadas a APIs y reemplazarlas por ejecución de código propio, y de esta manera conseguir evitar la fijación de certificados en las aplicaciones de Android.
* **JD-GUI**: es una herramienta de entorno gráfico [23] que muestra código fuente Java, concretamente de los archivos “.class”. En el proyecto se utiliza para analizar el código de las APKs. El código que se obtiene en un primer momento está en lenguaje Smali pero con la herramienta Apktool se puede convertir a lenguaje Java. Hay que destacar que se puede perder información al hacer este cambio, pero puede resultar útil para definir la estructura de métodos y clases de la aplicación.
* **VSCodium**: es una bifurcación de la herramienta [24] Visual Studio de Microsoft y al igual que esta, es un editor de código, permite gestionar y compilar proyectos de diferentes lenguajes de programación. En el proyecto se utiliza para poder analizar el código obtenido de las APKs.
* **Filebeat**: herramienta cuyo objetivo es reenviar y centralizar los datos de los registros generados [25]. Filebeat se instala como un agente que interactúa con otras herramientas con función de servidor cuya función es el procesado de logs y visualización.
* **Elasticsearch**: es un servidor con un motor de búsqueda de texto y de registros [26] que además dispone de interfaz que procesa ficheros con formato JSON. Se utiliza para poder localizar e interpretar resultados de logs obtenidos a partir de herramientas como Filebeat o Logstash. Además, envía estos resultados a la herramienta de visualización Kibana.
* **Kibana**: esta herramienta se emplea para buscar y visualizar datos [27] indexados por Elasticsearch. Se encarga de analizar los datos que recibe de los índices proporcionados por Elasticsearch mediante filtros y creaciones de gráficos. Es una herramienta de monitorización, administración y análisis de registros.

## Entorno

A continuación, se explica y detalla el entorno que se ha levantado para poder llevar a cabo el proyecto. Además, se comentará en qué consiste la plataforma en la que se dispone todo el sistema, cuáles son sus elementos, detallando más concretamente aquellos en los que se centra el desarrollo de este trabajo.

### Sistemas operativos y despliegue

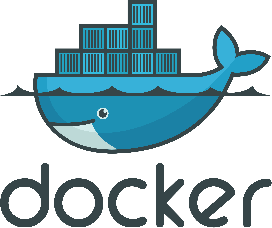
Se emplean dos sistemas operativos diferentes para el desarrollo de este proyecto. El primer sistema operativo que se utiliza es Parrot (Linux, de distribución Debian GNU), este sistema juega el rol de proxy de intercepción por lo que alberga los componentes necesarios para interceptar mensajes. El segundo sistema operativo es Android x86, que es el entorno de ejecución de las aplicaciones, las cuales serán analizadas y su tráfico será interceptado por el servidor proxy de la máquina Parrot. En las siguientes líneas de esta sección se explican las configuraciones iniciales para poder desplegar el entorno de trabajo.

En la figura que se muestra a continuación, se detalla el entorno de trabajo establecido. El entorno de trabajo se desarrolla bajo el uso de la aplicación Virtual Box [17], a partir del cual se han instalado por un lado el sistema operativo Parrot, que será el proxy de intercepción o servidor MiTM y, por otro lado, se ha instalado el sistema Android x86 [19], que es un emulador del sistema Android y que se utilizará como sistema de pruebas en lugar de un terminal móvil físico.

La única parte que cabe remarcar sobre la instalación de los dos sistemas en Virtual Box, son las configuraciones de red ya que como se aprecia en la figura de abajo, es necesario disponer de una red interna y de otra externa. La red interna se utilizará para la comunicación entre las máquinas de Parrot y Android x86, por tanto, es necesario establecer direcciones IP de las máquinas en la misma subred. Por otro lado, la red externa se empleará para conectar a Internet las máquinas. Tal y como se observa en la figura, la máquina Parrot es la que dispondrá de otro adaptador más a partir del cual se conectará a internet. Este adaptador tiene que ser del tipo puente, para permitir comunicaciones entre las máquinas virtuales a la vez que se permite la comunicación con internet. En la máquina Parrot se establecerá el proxy (mitmproxy [13]) a través del cual se conectará la máquina Android x86 de forma que tendrá acceso a Internet y así, desde el sistema Parrot se captura el tráfico.

Es necesario tener en cuenta que la máquina servidor MiTM, en este caso con sistema operativo Parrot, dispondrá de dos interfaces de red. Uno en modo puente y otro en modo red interna. La interfaz de red en modo puente tendrá asignada una dirección IP por DHCP, en cuanto a la otra interfaz de red, la interna, del sistema base, tendremos que asignarle una dirección IP estática, cuya subred sea diferente a la de la otra interfaz. En cuanto a la máquina Android x86, al arrancar la máquina hay que verificar que la conexión a Internet se produce a través de una interfaz de red en modo DHCP. Para asignar la dirección IP a esta máquina, se utilizará un fichero de configuración el cual hace uso del servicio dnsmasq [28]. Se puede encontrar más información acerca de este servicio y configuración en el Anexo de este documento.

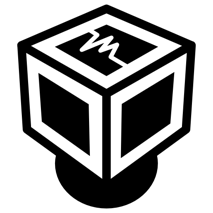
Por otro lado, para poder hacer uso de los diferentes componentes que permiten analizar el tráfico de las aplicaciones, es necesario acceder al contenedor en el que se han desarrollado, desde la máquina virtual Parrot ya que es en esta donde se desarrolla e implementan los métodos para anular la fijación de certificados de las aplicaciones Android que estarán situadas en la máquina del entorno de ejecución (Android x86).



**Docker**



**Red externa**

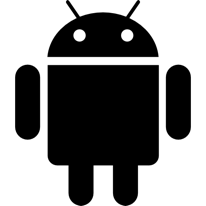


**Parrot (proxy)**

**Red interna**

**Internet**

**Virtual Box**





**Android x86**

Figura 3.1. Esquema del entorno de trabajo

### Componentes de las máquinas

Cada una de las dos máquinas descritas en el apartado anterior (el proxy de intercepción y el entorno de ejecución), disponen de componentes para poder realizar sus funciones. A continuación, se enumeran y explican los componentes de cada parte indicando de qué manera se comunican el proxy y el entorno de ejecución.

#### Proxy de intercepción

Tal y como se ha explicado en el apartado previo, el proxy de intercepción tiene como principal objetivo interferir en la comunicación entre las aplicaciones y los servidores a lo que realiza peticiones. Además, este proxy se encargará de realizar modificaciones de forma dinámica en la aplicación. Hay que destacar que todo este sistema está desarrollado como se comenta previamente, en un contenedor, el cual se divide en componentes. Los componentes que se utilizan en este proyecto son: tráfico, análisis, registros e intercepción (fijación de certificados).

* **Mitmproxy**: es el proxy que se utiliza para poder interceptar las comunicaciones de las aplicaciones.
* **Frida**: esta herramienta se utiliza para poder inyectar código modificado de las aplicaciones. Lógicamente, se modificará código de las aplicaciones para lograr la ruptura de la fijación de certificado de la aplicación.
* **Sistema de logs**: durante la ejecución de las aplicaciones y las fases de análisis e intercepción, se generan muchos registros con información relevante, tales como nombre APK, información sobre librerías de fijación de certificados, si se ha logrado interceptar el certificado, origen de comunicaciones de la aplicación, entre otros. Para gestionar estos registros se hace uso de las herramientas Filebeat, Elasicsearch y Kibana. Se explica este proceso con más detalle en la sección 4 de “Resultados”.

#### Entorno de ejecución

Este entorno de ejecución Android tiene como principal misión albergar las aplicaciones que serán objeto de análisis y estudio, además, debe ser capaz de comunicarse con el proxy y soportar las modificaciones que se realicen en las aplicaciones que se instalen. Los componentes necesarios para el entorno se indican a continuación.

* **Certificado del proxy**: mitmproxy dispone de certificados para poder interferir en las comunicaciones, es necesario que el entorno de ejecución incluya el certificado del proxy. Además de ser necesario para que funcione el proxy, también se utilizará para poder inyectar código mediante Frida como se explicará en los posteriores apartados.
* **Frida**: es necesario que esta herramienta esté instalada en el entorno de ejecución ya que permite la instrumentación de las aplicaciones y la suplantación de código.

## Método de identificación de bibliotecas de fijación de certificados

Uno de los primeros problemas que surgen al comienzo de este proyecto es saber qué librerías son las que implementan algún tipo de mecanismo de fijación de certificados. Con varias búsquedas se encontraron bastantes librerías que trata el problema comentado, como se puede consultar en [11], pero entonces, es necesario averiguar si realmente hay aplicaciones Android que hacen uso de estas librerías. En el capítulo 2 de este documento, “Estado del arte”, se puede ver una tabla que contiene las librerías encontradas durante el proceso de búsqueda.

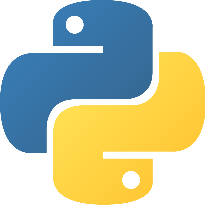
Para poder establecer si realmente las librerías encontradas son utilizadas en el desarrollo de las aplicaciones, se diseñan unos scripts que serán capaces de identificar si se hace uso de alguna de estas librerías en el código de cada aplicación que se quiera analizar. Para poder realizar este proceso, en primer lugar, es necesario disponer de varios APKs y no solamente esto, también hay que disponer del código de cada APK. Para ello, se utilizarán técnicas de ingeniería inversa y mediante el uso de la herramienta Apktool [15], se obtendrá de cada APK el código smali. Una vez obtenido el código, simplemente será necesario realizar una búsqueda en el código de cada una de las librerías que son conocidas por implementar certificate pinning.

El funcionamiento de los scripts es el que se relata a continuación. Por cada APK se crea un directorio el cual contiene el código de dicha APK. A partir de estos directorios se crean otro nuevos en los que se almacenarán ficheros en formato json cuyo nombre será el de la librería que se está buscando y que contendrán los resultados de las búsquedas. Es decir, habrá ficheros que si no han encontrado ningún resultado de la búsqueda estarán vacíos. Este proceso lo realizará un fichero en bash llamado *find\_libraries\_pinning.sh*. Posteriormente, mediante otro script en Python llamado *check\_pinning\_library.py* se analizarán los ficheros json anteriormente creados. Este script de Python verifica si están vacíos o no, en base a esta comprobación se generan ficheros en formato CSV que incluyen toda la información acerca de las APKs analizadas con sus respectivas librerías y si se ha encontrado algún indicio de que el código smali de cada APK tiene alguna de las librerías se indica en el CSV mediante la palabra clave “True”, si por el contrario no se ha encontrado nada entonces se indicará con “False”. Para unificar estos dos scripts, se ha implementado un script en bash, llamado *pinning\_libraries.sh*. Este script tiene como función llamar a los dos anteriores para así simplemente tener que ejecutar este y que realice todas las operaciones.

En el siguiente esquema se muestra gráficamente el funcionamiento general de los scripts explicados anteriormente para lograr identificar librerías que implementen mecanismos de certificate pinning en el código smali obtenido mediante Apktool [15].

1

**pinning\_libraries.sh**

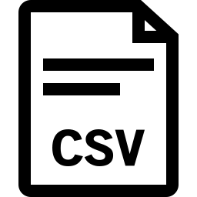


2

4

**check\_pinning\_library.py**

**find\_libraries\_pinning.sh**



3

5

Figura 3.2. Esquema del sistema de búsqueda de bibliotecas con certificate pinning

### Requisitos

Para comenzar, es requerimiento indispensable disponer de un sistema que disponga de terminal para poder ejecutar el programa. Por otro lado, como ya se mencionó, uno de los scripts del conjunto del programa está escrito en Python 3 por lo que será necesario disponer de dicho lenguaje de programación. Además, en el script de Python se utilizan dos importaciones de módulos, uno de ellos es *os* [29] y el otro es *csv* [30]. Es necesario que estén disponibles estas librerías para poder ejecutar el programa de forma correcta.

Este programa, como ya se explicó en el anterior apartado, hace uso de la herramienta Apktool [15]. Para que funcione correctamente la herramienta es necesario pasar como argumento un APK, por lo que es necesario disponer de algún APK antes de la ejecución del programa. Además, es necesario que el APK esté guardado en un directorio con el nombre “APKs” y que se localice en la misma ruta en la que se ubica el script del programa.

Finalmente, el usuario debe saber que al ejecutar el programa se crearán varios directorios. Estos directorios se llaman, “Resources” que contendrá otros tres directorios: “Code”, que contendrá el resultado de la ejecución de Apktool, es decir, el código de todos los APKs que se encuentren en el directorio “APKs”. “Files”, en este directorio se almacenan directorios correspondientes a cada APK disponible, que a su vez tendrán los ficheros json con las librerías encontradas y un fichero CSV con la información recogida. Finalmente, habrá otro directorio llamado “Logs”, que almacenará los diferentes ficheros de log generados por la ejecución del programa.

### Ejecución

Como se ha explicado en el apartado anterior, se ha creado un script en bash que funciona de enlace entre los otros dos, de tal forma que únicamente es necesario ejecutar este script para obtener los resultados esperados (ficheros CSV) acerca de si un APK utiliza para su implementación ciertas librerías conocidas de certificate pinning.

Para ejecutar el script genérico, *pinning\_libraries.sh*, es decir, el que devuelve todos los resultados sin tener que ejecutar ningún otro script, es suficiente con escribir *sudo* *bash pinning\_libraries.sh*. Como se aprecia en el comando, es necesario tener permisos de administrador. En la imagen que se muestra debajo, se indica de forma visual cómo se ha de ejecutar el comando y cuáles son las primeras líneas que aparecen al ejecutar el script.

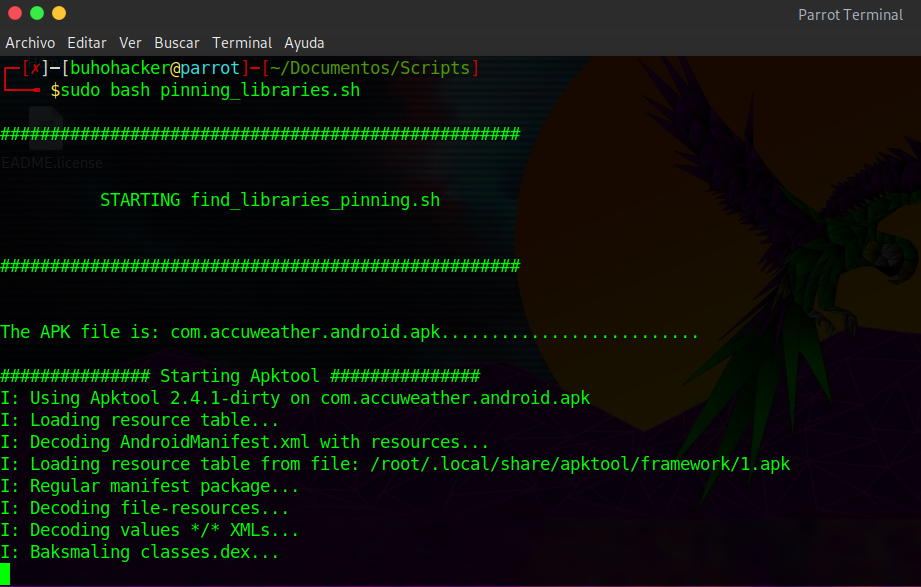


Figura 3.3. Ejecución del script pinning\_libraries.sh

Otra opción de ejecución sin necesidad de utilizar el script base *pinning\_libraries.sh*, sería ejecutar por separado los otros dos scripts, esto puede ser interesante si es necesario analizar las operaciones que realiza el código de cada uno de los scripts. Hay que tener en cuenta que para que funcione el script *check\_pinning\_library.py*, es necesario haber ejecutado previamente el otro script de bash, *find\_libraries\_pinning.sh*. Esto se debe a que el script de bash crea unos ficheros con formato json que posteriormente recogerá para analizar el script de Python. Para ejecutar *find\_libraries\_pinning.sh*, simplemente es necesario escribir en la terminal *sudo bash find\_libraries\_pinning.sh*. La ejecución del otro script requiere tener instalado Python en el sistema, para lanzar el script escribir el comando *python3* *check\_pinning\_library.py*, como observación, mencionar que en principio se puede ejecutar también con *python* en lugar de *python3,* pero se aconseja utilizar las versiones más recientes directamente.

En las siguientes imágenes se muestra en la primera, la ejecución del comando que lanza el script *find\_libraries\_pinning.sh* y a continuación, una imagen que muestra el comando y el principio de la ejecución del script *check\_pinning\_library.py.*

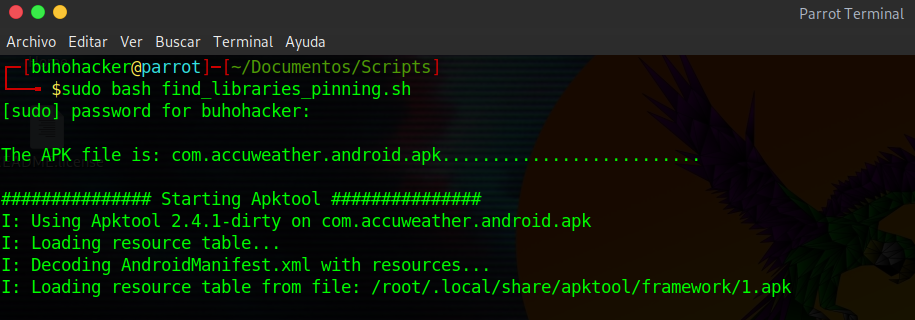


Figura 3.4. Ejecución del script find\_libraries\_pinning.sh

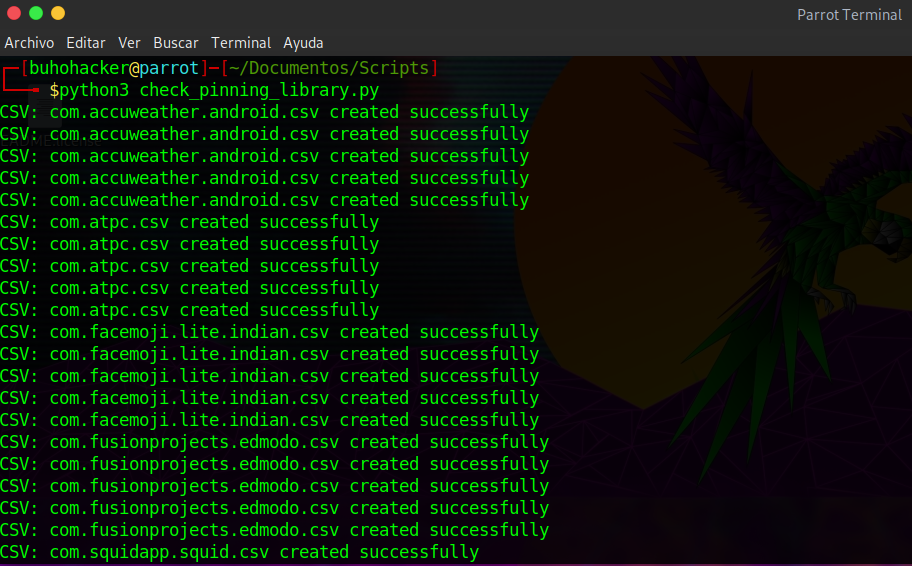


Figura 3.5. Ejecución del script check\_pinning\_library.py

Durante la ejecución del programa se van generando diferentes comentarios informativos por pantalla para que el usuario pueda saber en qué punto se encuentra del proceso y, sobre diferentes errores que puedan haber surgido durante ejecución. Además, se generan logs con posibles errores surgidos y para almacenar el resultado de las ejecuciones. Dichos ficheros se pueden encontrar en el directorio *Logs*.

### Salida

Una vez ejecutados los scripts, como se muestra en el esquema de funcionamiento del sistema en el primer apartado, se obtiene como resultado varios ficheros. Al ejecutar el script *find\_libraries\_pinning.sh*, se obtienen varios ficheros en formato json con las ocurrencias de las librerías encontradas (sí que se ha hecho uso de ellas en el desarrollo de la aplicación que se está analizando). En la siguiente imagen se muestra el contenido de uno de los ficheros. El nombre que recibirá cada uno de estos ficheros corresponde al nombre de la librería que se está analizando.

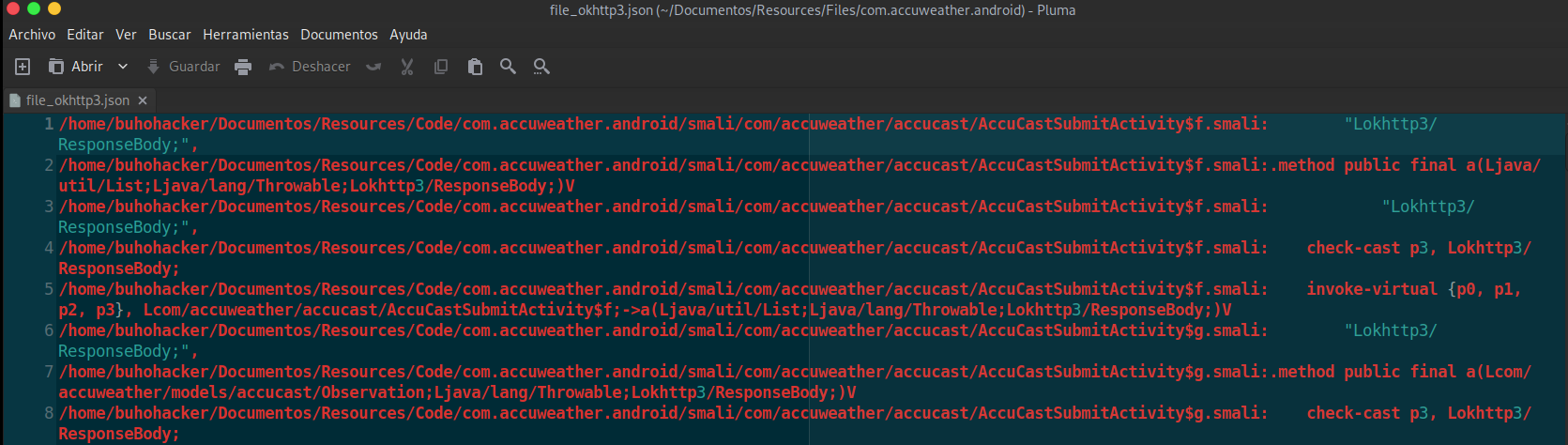


Figura 3.6. Contenido de uno de los ficheros JSON

Por otro lado, el otro script de Python, después de realizar comprobaciones y determinar si realmente hay librerías o no en el código del APK analizado, crea un fichero con formato CSV cuyo nombre corresponderá con el del APK analizado. Este fichero CSV tiene como columnas los siguientes valores: nombre del APK, nombre de la librería y finalmente el valor “True” si utiliza la librería o “False” si no la utiliza.

En la imagen que se muestra a continuación, se ve el contenido de uno de los ficheros CSV generados por el sistema desarrollado. Como se aprecia en la imagen, el APK con el que se ha trabajado para esta demostración es *com.facemoji.lite.indian*, en la siguiente columna aparecen los nombres de las librerías con las que se ha probado, *opensslsocketimpl*, *okhttp3*, *trustkit* y *trustmanager.* Y en la última columna se indica si ese APK en concreto tiene o no dichas librerías.

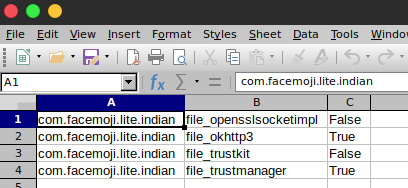


Figura 3.7. Contenido de un fichero CSV generado por el sistema

## Método de inyección de código

Una vez que ya se ha conseguido averiguar qué aplicaciones son las que incluyen mecanismos de certificate pinning y más concretamente, se ha podido averiguar qué librería en particular utilizan, el siguiente paso consiste en modificar el código de aquellos métodos que se utilicen para la fijación de certificados. Para inyectar posteriormente el código cambiado, se hace uso de la herramienta Frida, de tal forma que, teniendo el servicio levantado, únicamente es necesario indicar el fichero (*pinning\_cases.js*), en este caso escrito en Java Script, que contiene los casos detectados de certificate pinning con el código modificado para que, al lanzar el proxy y una aplicación se modifique su código.

El primer apartado de esta sección, desarrolla mecanismos que se han empleado para inyectar código modificado de tal forma que se intercepte el certificado fijado en la aplicación. Concretamente, se hace referencia a los casos de librerías nuevas descubiertas.

En el siguiente apartado, se detallan los mecanismos de inyección de código para el caso de librerías ya conocidas pero cuyos métodos implementados anteriormente no eran capaces de interceptar el certificado fijado en ciertas aplicaciones. Por ello, se desarrollan nuevos métodos.

Un aspecto importante que hay que tener en cuenta es el tipo de APK con el que se realizan las pruebas de los mecanismos diseñados para evitar la fijación de certificados. En este proyecto se han distinguido dos tipos de APKs diferentes. En primer lugar, las APKs simples, son aquellas que únicamente disponen de un único fichero a partir del cual se instala la aplicación en el entorno de ejecución. Por otra parte, el segundo tipo de APK que se ha analizado durante el desarrollo del trabajo, son las Bundle APKs [31]. Este tipo de APK se diferencia de la anterior en que se incluye todo el código compilado y los recursos de una aplicación, pero, sin embargo, la generación de la firma y certificado de la aplicación es delegada al servicio de Google Play. Además, resulta que para poder instalar estos paquetes con la herramienta adb, es necesario utilizar un comando específico y también modificar los scripts que interceptan las comunicaciones ya que no se puede indicar un APK concreto, ahora es el paquete de la aplicación el que se debe instalar.

Segundo apartado? 🡪 IDA

Poner los diferentes métodos de saltarse Pinning

Varios escenarios?? 🡪 Más con bibliotecas y otro método diferente

### Caso 1: Inyección de código con nuevas librerías

Este caso hace referencia a la situación en la que se han detectado nuevas librerías que implementan métodos de fijación de certificados en aplicaciones Android [11]. Es cierto que estas librerías al ser más recientes, muchas aplicaciones no las implementan, pero, aun así, se han llevado a cabo mecanismos capaces de modificar el código para romper la fijación de certificados.

A continuación, se indican las librerías nuevas con una descripción del proceso seguido para poder evitar la fijación de certificados.

* **OpenSSlSocketImpl Conscrypt**: la modificación del código consiste en crear mediante Frida un nuevo socket SSL. Además, con la función *verifyCertificateChain*, que tiene una lista de los certificados válidos y que, si el certificado no está se lanza una excepción; se utiliza con Frida de tal forma que se sustituyen los valores de la función y si se consigue interceptar el certificado, se imprime un mensaje indicándolo. Por el contrario, si no se consigue interceptar, se envía un mensaje indicando que no se hace uso de esta librería.
* **OpenSSlEngineSocketImpl Conscrypt**: el mecanismo desarrollado que utiliza Frida funciona de la misma manera que el apartado anterior, simplemente se realizar una sobreescritura del mismo método en el que se trata de comprobar los certificados y el método de autenticación, de forma que si se consigue interceptar el certificado se envía un mensaje a los registros. Si no se consigue realizar esta modificación, entonces se envía un mensaje al registro indicando que no se utiliza esta librería.
* **OpenSSlSocketImpl Apache Harmony**: muy similar a los dos métodos anteriores con la diferencia de que cambia la librería empleada. Con Frida se consiguen pasar otros parámetros diferentes a la función *verifyCertificateChain* de forma que, si se está utilizando esta librería, se puede interceptar la fijación de certificados. Se envía al log un mensaje de que se ha interceptado correctamente o, por el contrario, de que no se ha detectado el uso de esta librería en la aplicación.
* **PhoneGap sslCertificateChecker**: en este caso lo que se modifica es el método *execute* de la clase *sslCertificateChecker*. En definitiva, lo que se hace con Frida es tratar de obviar las huellas del certificado que se quiere utiliza de manera que, lo que realmente acaba haciendo la función es imprimir un mensaje evitando que salte una excepción por error de conexión no segura.
* **IBM MobileFirst**: en primer lugar, se obtiene mediante una variable el *Activity* de la aplicación. Después, lo que se hace es modificar la función de la clave pública del certificado (*pinTrustedCertificatePublicKey*) de forma que si se intercepta se envía al registro un mensaje indicándolo.
* **IBM WorkLight**: en esta modificación con Frida lo que se hace es cambiar la función que comprueba el nombre del servidor y también el certificado que se utilizará. De esta forma, lo único que hace es imprimir un mensaje en el log indicando si se ha interceptado o si no se usa esta librería en la aplicación.
* **Conscrypt CertPinManager**: en esta situación con en otras previas, lo que hace la función *isChainValid* es comprobar la validez de los certificados de manera que si no es válido se lanza una excepción. Mediante Frida se logra que se modifique la función y que, en el momento en el que se intenta comprobar el certificado, lo que hará realmente la función con esta modificación es imprimir un mensaje sin llegar a lanzar la excepción.
* **CWAC-Netsecurity**: el funcionamiento es el mismo que el de la librería anterior. Se utiliza el método *isChainValid* pero de la propia librería. Utilizando Frida se modifica la función de manera que lo único que hará será enviar un mensaje y se saltará la excepción que es lo que hace que se rechace el certificado no válido.
* **Worklight Androidgap**: este caso es similar al de PhoneGap, la librería utiliza un método *execute* que cumple funciones similares a las del caso ya mencionado. Frida realiza cambios muy parecidos a los descritos previamente de forma que se evita que se lance la excepción, consiguiendo así interceptar la fijación.
* **Netty FingerprintTrustManagerFactory**: el mecanismo que se implementa con Frida consiste en modificar el método *checkTrusted*, de manera que se cambia el funcionamiento normal, es decir, verificar que es un certificado de confianza pasando a la función un string que indica el tipo (cliente o servidor) y una cadena que es el certificado. La modificación consiste en incluir los propios parámetros obtenidos de la aplicación, logrando cambiar el comportamiento de la función y haciendo que se imprima un mensaje que indique si se ha interceptado la fijación.
* **Squareup CertificatePinner**: se modifica el comportamiento del método *check* que comprueba que los certificados sean válidos, de manera que lo que se consigue con Frida es evitar que salte la excepción o error cuando la función trata de verificar que el certificado es correcto. Realmente lo que hace Frida es lanzar un mensaje por pantalla.
* **Squareup OkHostnameVerifier**: esta librería se modifica de la misma manera que la explicada anteriormente. La diferencia es que utiliza un método llamado *verify*, pero la funcionalidad es la misma que la comentada en la librería Squareup CertificatePinner.
* **Android WebViewClient**: el método que se modifica es *onReceivedSslError*. Este método se emplea para notificar a la aplicación errores en el certificado SSL. Con Frida se modifica de forma que el método no sea capaz de lanzar la excepción y simplemente, se enviará un mensaje indicando que se ha interceptado la fijación del certificado, si es que se ha utilizado esta librería en la aplicación que se está analizando.
* **Apache Cordova WebViewClient**: el mecanismo que se utiliza en esta librería para evitar la fijación, es igual que el explicado en la anterior librería. Se varía el método *onReceivedSslError* consiguiendo que se intercepte la fijación y el método imprimirá un mensaje con el serultado.
* **Boye AbstractVerifier**: mediante la función *verify*, se comprueba la validez de un certificado, si resulta que no es válido se produce un error en el método. Con la modificación de Frida se consigue acceder en tiempo de ejecución modificando los parámetros de entrada del método logrando interceptar el mecanismo de fijación de certificado. Lo que devolverá realmente la función es un mensaje por pantalla o a los registros, en vez de lanzar la excepción de la validación del certificado.

### Caso 2: Inyección de código nuevo de librerías ya conocidas (APKs simples)

Este segundo caso se refiere a la situación en la que ya se habían desarrollado métodos para evitar certificate pinning con librerías conocidas, pero, en determinadas aplicaciones no se logró interceptar el certificado. Es decir, se registró que se hace uso de una librería concreta pero los métodos diseñados con Frida para romperla no son eficaces. Por este motivo, en esta sección se indicarán esas bibliotecas y además los mecanismos nuevos que se han diseñado a partir de los cuales se logra interceptar la fijación de certificados. Además, es necesario realizar una breve distinción entre APKs simples y Bundle APKs ya que se han realizado pruebas con los dos tipos y se tiene que tener en cuenta ciertos aspectos determinantes para poder instalar dichas aplicaciones en el entorno de desarrollo aunque si es cierto que realmente, el código implementado para inyectar las modificaciones con la herramienta de instrumentación dinámica Frida, no es necesario modificarlo, es decir, se utiliza el mismo código para los dos tipos de APKs.

Las bibliotecas ya conocidas se listan a continuación, y, además, se añade una descripción del proceso que se ha seguido para poder desarrollar los mecanismos que logran interceptar la fijación de certificados.

* **TrustManager**: para esta librería se han desarrollado varios mecanismos ya que las aplicaciones utilizan una gran variedad de combinaciones de clases y métodos para implementar la fijación de certificados.
* **TrustManagerImpl**: para esta librería se ha modificado el código que se inyecta con Frida. Concretamente se ha modificado la función *checkTrustedRecursive*. Esta función verifica una lista de certificados de forma recursiva por lo que simplemente basta con que Frida imprima un mensaje en el momento en el que se va a verificar el certificado. Este mensaje será indicativo de que se ha logrado saltar el certificado. Si no se consigue es que entonces la aplicación no usa esa librería o la utiliza, pero con otras funciones no identificadas aún.
* **Android WebViewClient**: Se ha desarrollado otra manera de evitar la fijación con esta librería y para ello, se hace uso del método *loadUrl*, consiguiendo que Frida lance un mensaje evitando que se realice la conexión entre la aplicación que se está ejecutando y el servidor correspondiente a la aplicación.

Tick Ok Check - Free vector graphic on PixabayEn la tabla que se muestra a continuación, se indican las librerías que ya se utilizaban previamente, en la fase anterior a la realización de este proyecto y, las librerías que se han añadido nuevas. Además, en las librerías de resultados previos se indica con que se han implementado métodos nuevos para lograr su funcionamiento.

|  |  |
| --- | --- |
| Librerías previas | Librerías nuevas |
| Tick Ok Check - Free vector graphic on PixabayTrustmanager | Android WebViewClient |
| Okhttp3 | OpenSSLSocketImpl Conscrypt |
| Truskit | OpenSSLEngineSocketImpl Conscrypt |
| Tick Ok Check - Free vector graphic on PixabayTrustmanagerImpl | OpenSSLSocketImpl Apache |
| Appcelerator | PhoneGap sslCertificateChecker |
|  | IBM MobileFirst |
|  | IBM WorkLight |
|  | Conscrypt CertPinManager |
|  | CWAC-Netsecurity |
|  | Worklight Androidgap |
|  | Netty FingerprintTrustManagerFactory |
|  | Squareup CertificatePinner |
|  | Squareup OkHostnameVerifier |
|  | Apache Cordova WebView |
|  | Boye AbstractVerifier |

Tabla 3.1. Resumen de librerías previas y de las librerías nuevas

(apartado 3.5.3) 🡪 Ghidra

## Método de ejecución de múltiples APKs

La finalidad de este método es conseguir ejecutar múltiples APKs sin tener que estar ejecutándolas individualmente. La necesidad de tener que implementar este sistema surge al analizar múltiples APKs para comprobar el funcionamiento de los métodos descritos en el anterior apartado. Lógicamente, es muy costoso tener que ir ejecutando el script para cada APK. El mecanismo implementado lo único que hace es llamar al script *traffic.py* que es el que se encarga de instalar la aplicación en el entorno de ejecución, realizar las fases I y II, la primera fase consiste en capturar tráfico sin realizar ninguna acción en la aplicación. En la segunda fase se captura el tráfico generado en la aplicación realizando ciertas operaciones tales como pulsar botones, etc. Siguiendo con el programa desarrollado, se llama a *traffic.py* en un bucle de forma que se le pasa como argumento cada vez las APKs que estén en un directorio concreto que se debe especificar.

DIAGRAMA DE FLUJO DEL PROGRAMA?

# Resultados

## Introducción

En esta sección se muestran los resultados que se han obtenido a partir de las diferentes implementaciones sobre los casos de inyección de código, para evitar así la fijación de certificados en el terminal. Se ha dividido en subapartados, en función del caso de estudio que se ha considerado en cada momento. En primer lugar, se detallan los resultados obtenidos respecto al caso de la inyección de código que evite el certificate pinning en nuevas librerías descubiertas. A continuación, se analizarán los resultados obtenidos acerca del caso en el que se conocen las librerías que implementan la fijación de certificados, pero, no se conseguía romper el certificado, concretamente en el caso de APKs simples. Finalmente, se analizan los resultados obtenidos del último caso, el cual se parte de las librerías ya conocidas y se diseñan nuevos métodos para romper la fijación de certificados, pero en Bundle APKs.

Un último apartado se dedicará a explicar el sistema de gestión de registros que se ha desarrollado como otro componente, para poder visualizar los diferentes casos comentados previamente.

¿Gráfico con tiempo de ejecución de la búsqueda de bibliotecas ?

## Análisis de los resultados de la inyección de código

En este apartado se exponen y analizan los resultados obtenidos tras haber establecido y probado los mecanismos del capítulo 3, los cuales inyectan código en las aplicaciones utilizando la herramienta Frida.

Para comenzar las pruebas y comprobar la validez de los mecanismos implementados se han seleccionado nueve aplicaciones de las cuales ya se tenían resultados previos, de manera que se comprobó que con las implementaciones previas no se consiguió romper la fijación de las librerías que ya estaban diseñadas. Tres de esas aplicaciones son bundle APKs y el resto son APKs simples.

Hay que añadir que, para facilitar la fase de análisis y comprensión de los resultados, se han utilizado herramientas de gestión de registros que se explican en el siguiente apartado de este mismo capítulo. Se ha considerado necesario diseñar esta fase ya que los scripts empleados para la fase de tráfico e intercepción de comunicaciones muestran sus resultados en logs. Gracias a las herramientas de gestión de registros es más sencillo analizar los resultados obtenidos acerca de los mecanismos implementados.

En la tabla siguiente se recogen las aplicaciones a partir de las cuales se han realizado las pruebas indicando, además, el tipo de APK correspondiente. Como se explicó en capítulos anteriores, la principal diferencia entre las APKs simples y las bundle es, a grandes rasgos para este proyecto, la manera en la que se instala la aplicación en el entorno de ejecución. Las simples no es necesario instalarlas ya que la plataforma cuenta con un script capaz de hacerlo, sin embargo, para las bundle APKs, es necesario instalarlas a mano en el entorno ya que están divididas en varios ficheros.

|  |  |
| --- | --- |
| Aplicación | Tipo de APK |
|  | Simple |
|  | Simple |
|  | Simple |
|  | Simple |
|  | Simple |
|  | Simple |
| Backgrounds | Bundle |
| Zombie Tsunami | Bundle |
| Retrica | Bundle |

Tabla 4.1. Resumen de las aplicaciones analizadas

|  |  |  |  |
| --- | --- | --- | --- |
| Aplicación | Librerías detectadas | Intercepta conexión protegida (resultados previos) | Intercepta conexión protegida (resultados actuales) |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| Backgrounds |  |  |  |
| Zombie Tsunami |  |  |  |
| Retrica |  |  |  |

## Gestión de registros

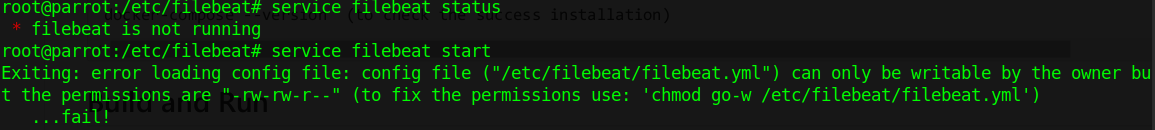
Explicar pq es necesario utilizar herramientas de gestión de logs.

ESQUEMA DE FUNCIONAMIENTO EKF CON AGENTE Y SERVIDOR

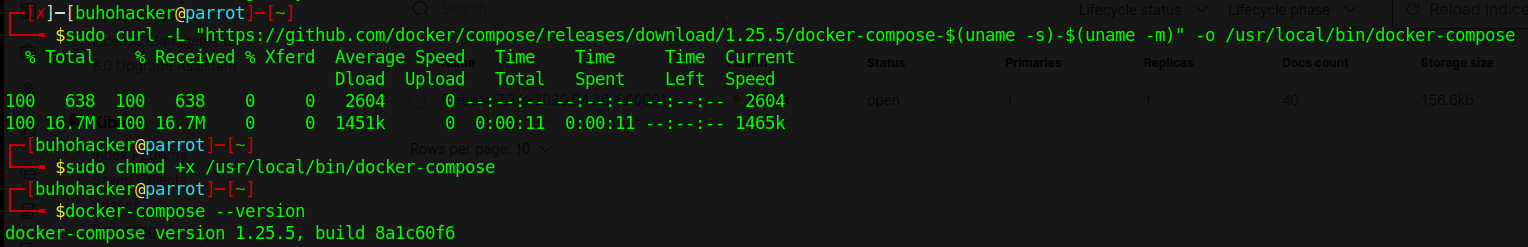
Elasticsearch, Kibana y Filebeat

Para la gestión de los logs se han utiizado as herramientas Elasticsearch, Kibana y Filebeat. Este sistema de gestión se divide en dos secciones, el servidor y el agente. El agente estará formado por Filebeat el cual se deberá instalar en el proxy de intercepción, es decir, en el contenedor que incluye el módulo de tráfico y en el que se encuentran los casos de certificate pinning, Frida, entre otros. Además, en este contenedor se almacenarán los logs de resultados obtenidos al instalar cada APK en el terminal y al ejecutar los casos de fijación de certificados para lograr romperlo. El agente ya está instalado en el contendor por lo que lo único que hay que hacer es modificar la dirección IP del fichero de configuración de Filebeat, este fichero se llama filebeat.yml y se encuentra en la ruta /etc/filebeat.

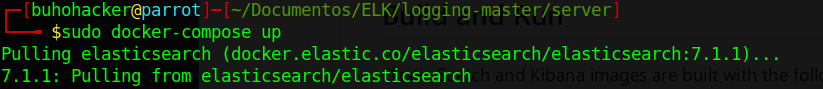
Una vez que se ha verificado que el fichero de configuración es correcto, simplemente hay que iniciar el servicio mediante el comando *service filebeat start*, para comprobar el estado del servicio se puede utilizar el comando *service filebeat status*. Si surge algún problema al tratar de arrancar el servicio puede ser debido a que se escribió el fichero de configuración de filebeat con permisos diferentes. Bastará con cambiar los permisos y volver a ejecutar el comando mencionado anteriormente para que arranque definitivamente el agente. En la imagen que se muestra a continuación, se observa el error mencionado y el funcionamiento del comando para visualizar el estado del agente.



Configuración del servidor

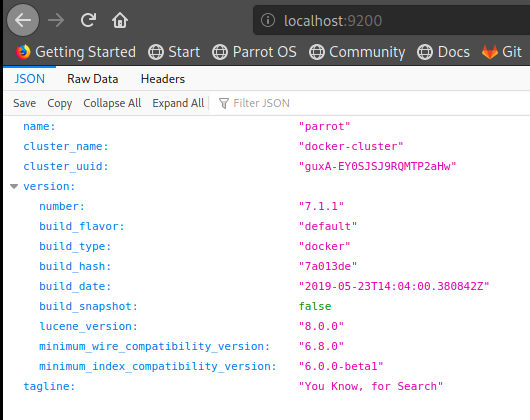
Comandos para instalar Docker compose.

Una vez instalado Docker compose, simplemente hay que levantar el servicio. Para ello, dirigirse al directorio en el que se encuentra la carpeta que contiene el contenedor con elasticsearch y kibana y ejecutar el comando *sudo docker-compose up*.

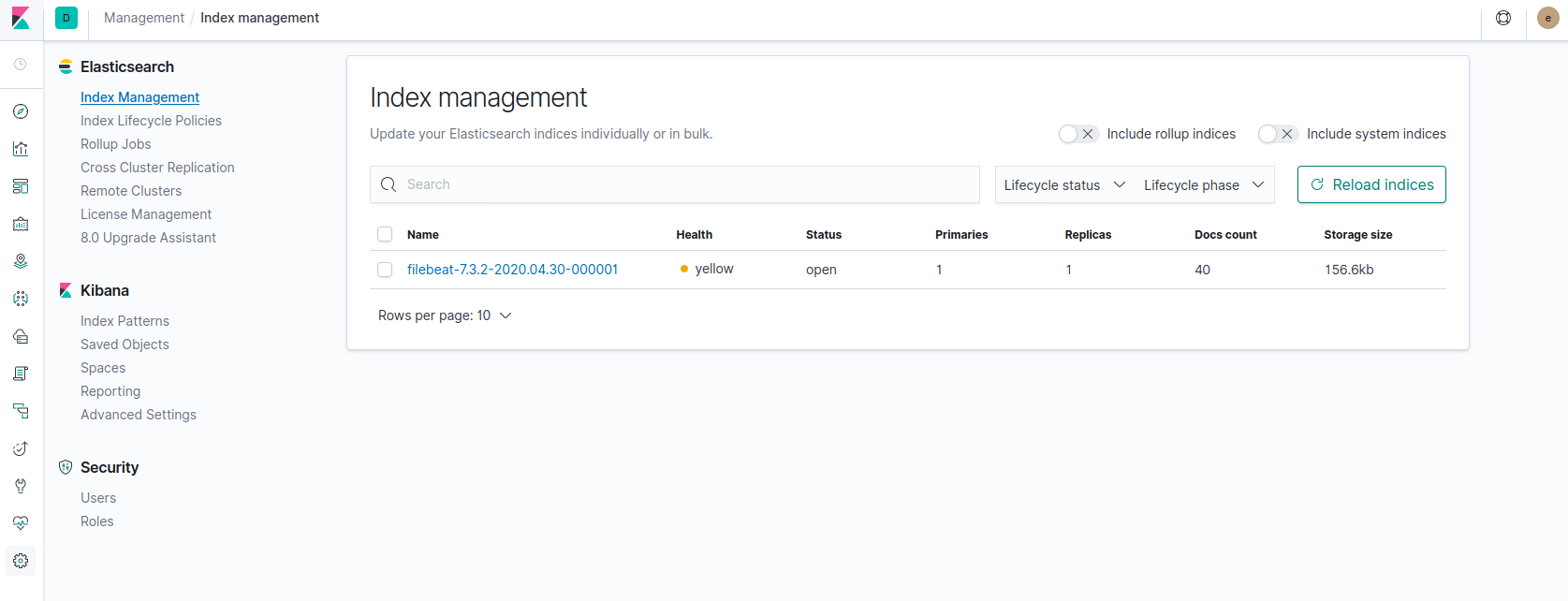


Es posible que, de un error al tratar de lanzar el servidor, si esto ocurre, simplemente será necesario modificar los permisos al directorio esdata01.

Para acceder via web a Elasticsearch, bastará con poner http://localhost:9200 , también es posible indicar la ip de la máquina en vez localhost o el nombre del equipo. En la siguiente imagen se muestra cómo se visualiza Elasticsearch.



Para acceder a Kibana, añadir la dirección <http://localhost:5601>, o con la dirección IP o nombre de la máquina correspondiente. A continuación, se muestra una imagen de cómo se visualizaría Kibana.



# Conclusiones y líneas futuras

Apéndices y/o anexos

Configuración dnsmasq

# Bibliografía

|  |  |
| --- | --- |
| [1] | «Redalia,» [En línea]. Available: https://www.redalia.es/ssl/protocolo-ssl/. [Último acceso: 08 03 2020]. |
| [2] | Wikipedia, «Wikipedia,» 15 02 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Protocolo\_de\_configuraci%C3%B3n\_din%C3%A1mica\_de\_host. [Último acceso: 16 04 2020]. |
| [3] | B. Afonso, «Las aplicaciones que deberían preocuparte son estas y no FaceApp,» Revista GQ, 19 07 2019. [En línea]. Available: https://www.revistagq.com/noticias/articulo/cuales-son-las-aplicaciones-que-venden-mis-datos-a-terceros-faceapp. [Último acceso: 05 11 2020]. |
| [4] | E. Z. Beamud, *Desarrollo de una prueba de concepto para el análisis estático de aplicaciones móviles enfocado a la detección de fugas de información personal,* Madrid, 25/06/2019. |
| [5] | L. M. T. Laverón, *Development of a system for traffic analysis of smartphone apps for private data exfiltration detection,* Madrid, 25/06/2019. |
| [6] | S. F. García, *Desarrollo de una prueba de concepto para la detección y circunvalación de mecanismos de comprobación de certificados digitales en aplicaciones Android,* Madrid, 2019. |
| [7] | W. Commons, «File: Chain of trust.svg,» [En línea]. Available: https://commons.wikimedia.org/w/index.php?title=File:Chain\_of\_trust.svg&lang=es. [Último acceso: 15 03 2020]. |
| [8] | IBM, «Conceptos de Secure Sockets Layer (SSL) y Transport Layer Security (TLS),» 02 04 2020. [En línea]. Available: https://www.ibm.com/support/knowledgecenter/SSFKSJ\_7.1.0/com.ibm.mq.doc/sy10640\_.htm. [Último acceso: 15 04 2020]. |
| [9] | IBM, «Una descripción general del protocolo de enlace SSL o TLS,» 02 04 2020. [En línea]. Available: https://www.ibm.com/support/knowledgecenter/en/SSFKSJ\_7.1.0/com.ibm.mq.doc/sy10660\_.htm. [Último acceso: 15 04 2020]. |
| [10] | OWASP, «Certificate and Public Key Pinning,» [En línea]. Available: https://owasp.org/www-community/controls/Certificate\_and\_Public\_Key\_Pinning. [Último acceso: 12 05 2020]. |
| [11] | akabe1, «GitHub,» [En línea]. Available: https://gist.github.com/akabe1/5632cbc1cd49f0237cbd0a93bc8e4452. [Último acceso: 15 04 2020]. |
| [12] | Portswigger, «Burp Suite,» [En línea]. Available: https://portswigger.net/burp. [Último acceso: 11 05 2020]. |
| [13] | mitmproxy, «mitproxy,» [En línea]. Available: https://mitmproxy.org/. [Último acceso: 13 04 2020]. |
| [14] | apkpure, «apkpure,» [En línea]. Available: https://apkpure.com/es/. [Último acceso: 11 05 2020]. |
| [15] | Apktool, «Apktool,» [En línea]. Available: https://ibotpeaches.github.io/Apktool/. [Último acceso: 13 04 2020]. |
| [16] | A. Developers, «Ofuscación código Android,» [En línea]. Available: https://developer.android.com/studio/build/shrink-code?hl=es-419#obfuscate. [Último acceso: 11 05 2020]. |
| [17] | VirtualBox, «VirtualBox,» [En línea]. Available: https://www.virtualbox.org/wiki/Documentation. [Último acceso: 13 04 2020]. |
| [18] | P. Team, «The Parrot Project,» [En línea]. Available: https://parrotlinux.org/docs/#documentation. [Último acceso: 13 04 2020]. |
| [19] | A. x86, «Android x86,» [En línea]. Available: https://www.android-x86.org/. [Último acceso: 13 04 2020]. |
| [20] | D. Docs, «Docker Documentation,» [En línea]. Available: https://docs.docker.com/. [Último acceso: 13 04 2020]. |
| [21] | A. Studio, «Android developers,» [En línea]. Available: https://developer.android.com/studio/command-line/adb. [Último acceso: 13 04 2020]. |
| [22] | Frida, «Frida,» [En línea]. Available: https://frida.re/. [Último acceso: 16 04 2020]. |
| [23] | J. Decompiler, «Java Decompiler,» [En línea]. Available: http://java-decompiler.github.io/. [Último acceso: 25 04 2020]. |
| [24] | VSCodium, «Free/Libre Open Source Software Binaries of VSCode,» [En línea]. Available: https://vscodium.com/. [Último acceso: 11 04 2020]. |
| [25] | Elastic, «Filebeat overview,» [En línea]. Available: https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-overview.html. [Último acceso: 29 04 2020]. |
| [26] | Wikipedia, «Elasticsearch,» 03 03 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Elasticsearch. [Último acceso: 19 04 2020]. |
| [27] | Elastic, «¿Qué es Kibana?,» [En línea]. Available: https://www.elastic.co/es/what-is/kibana. [Último acceso: 23 04 2020]. |
| [28] | A. Linux, «dnsmasq (Español),» 20 02 2020. [En línea]. Available: https://wiki.archlinux.org/index.php/Dnsmasq\_(Espa%C3%B1ol). [Último acceso: 16 04 2020]. |
| [29] | Python, «os- Interfaces diversas del sistema operativo,» [En línea]. Available: https://docs.python.org/3/library/os.html. [Último acceso: 18 04 2020]. |
| [30] | Python, «csv- Lectura y escritura de archivos CSV,» [En línea]. Available: https://docs.python.org/3/library/csv.html?highlight=csv#module-csv. [Último acceso: 18 04 2020]. |
| [31] | A. Developers, «Acerca de los paquetes Android App Bundle,» [En línea]. Available: https://developer.android.com/guide/app-bundle?hl=es. [Último acceso: 28 04 2020]. |
| [32] | «IBM Mobile Foundation,» 27 02 2020. [En línea]. Available: https://mobilefirstplatform.ibmcloud.com/tutorials/es/foundation/8.0/authentication-and-security/certificate-pinning/. [Último acceso: 08 03 2020]. |
| [33] | IconsMind, «IconsMind,» [En línea]. Available: https://iconsmind.com/. [Último acceso: 13 04 2020]. |
| [34] | IconArchive, «IconArchive,» [En línea]. Available: http://www.iconarchive.com/show/outline-icons-by-iconsmind/Smartphone-4-icon.html. [Último acceso: 13 04 2020]. |
| [35] | IconArchive, «IconArchive,» [En línea]. Available: http://www.iconarchive.com/show/papirus-devices-icons-by-papirus-team/network-server-icon.html. [Último acceso: 13 04 2020]. |
| [36] | Freepik, «Freepik,» [En línea]. Available: https://www.freepik.com/free-icon/login-password\_870212.htm#page=1&query=login&position=0. [Último acceso: 13 04 2020]. |
| [37] | Freepik, «Freepik,» [En línea]. Available: freepik.com/free-icon/login\_863353.htm#page=1&query=login&position=8. [Último acceso: 13 04 2020]. |
| [38] | Freepik, «Freepik,» [En línea]. Available: https://www.freepik.com/free-icon/database\_873831.htm#page=1&query=server&position=13. [Último acceso: 13 04 2020]. |
| [39] | Icons8, «Icons8,» [En línea]. Available: https://icons8.com/icons/set/hi-tech. [Último acceso: 13 04 2020]. |
| [40] | FreeIconsPNG, «FreeIconsPNG,» [En línea]. Available: https://www.freeiconspng.com/images/site-internet-icon. [Último acceso: 13 04 2020]. |
| [41] | L. Download, «Logos Download,» [En línea]. Available: https://logos-download.com/9688-docker-logo-download.html. [Último acceso: 13 04 2020]. |
| [42] | A. Studio, «Android developers adb,» [En línea]. Available: https://developer.android.com/studio/command-line/adb. [Último acceso: 13 04 2020]. |
| [43] | VectorStock, «JSON,» [En línea]. Available: https://www.vectorstock.com/royalty-free-vector/json-file-document-icon-vector-24684476. [Último acceso: 16 04 2020]. |
| [44] | ICONSHOP, «CSV,» [En línea]. Available: https://freeiconshop.com/icon/csv-file-icon-outline/. [Último acceso: 16 04 2020]. |