UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DESARROLLO DE UNA PRUEBA DE CONCEPTO PARA LA DETECCIÓN Y CIRCUNVALACIÓN DE MECANISMOS DE COMPROBACIÓN DE CERTIFICADOS DIGITALES EN APLICACIONES ANDROID

SERGIO FRAMIÑÁN GARCÍA

2019

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Título: Desarrollo de una prueba de concepto para la detección y circunvalación de mecanismos de comprobación de certificados digitales en aplicaciones Android.

Autor: D. Sergio Framiñán García

Tutor: D. José María del Álamo Ramiro

Ponente: D. ……

Departamento: Departamento de Ingeniería de Sistemas Telemáticos (DIT)

MIEMBROS DEL TRIBUNAL

Presidente: D. ……

Vocal: D. ……..

Secretario: D. ……..

Suplente: D. ……..

Los miembros del tribunal arriba nombrados acuerdan otorgar la calificación de: ……

Madrid, a de de 20…

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DESARROLLO DE UNA PRUEBA DE CONCEPTO PARA LA DETECCIÓN Y CIRCUNVALACIÓN DE MECANISMOS DE COMPROBACIÓN DE CERTIFICADOS DIGITALES EN APLICACIONES ANDROID

SERGIO FRAMIÑÁN GARCÍA

2019

Gracias a mis padres, sin los que habría sido imposible llegar hasta aquí. Os quiero.
Gracias a Álvaro, gran compañero de principio a fin sin el que nada habría sido igual.
Gracias a Marta, por acompañarme estos años y aguantar este final de carrera a mi lado.
Gracias a mi familia y amigos, por estar siempre conmigo.

RESUMEN

El objetivo de este trabajo es desarrollar un componente que permita auditar las conexiones https de aplicaciones Android. Para ello se van a estudiar las características del protocolo y los diferentes mecanismos para implementarlas en Android. Una vez estudiado esto, se presentarán de forma teórica los diferentes métodos existentes para poder auditar conexiones cifradas y los componentes desarrollados para llevar a cabo dichos métodos de forma automática. También se mostrarán los resultados del uso de dichos componentes desarrollados. Por último, se expondrán unas conclusiones y líneas de trabajo futuro, acompañadas de una guía de buenas prácticas para el establecimiento de conexiones seguras en el entorno Android.

SUMMARY

The objective of this thesis is to explain the development of a software that allows us to intercept and analyze https connections made by an android application. In order to understand this software, it is necessary to explain the existing protocols used to establish an internet connection and the different mechanisms used to implement them in an Android application. Once this has been reached, we will introduce the different existing methods used to audit encrypted connections, as well as the components that allow to automatize these methods. The results of the use of such developed components will be shown as well. We will end with the conclusions along with a small guide about safe software development.

PALABRAS CLAVE

Certificados, fijación de certificados, SSL, TLS, auditoría, privacidad, protección de datos personales, ciberseguridad, http, https, Android, Frida, instrumentación.

KEYWORDS

Certificate, certificate pinning, SSL, TLS, audit, privacy, protecting personal information, cybersecurity, http, https, Android, Frida, dynamic instrumentation.

ÍNDICE DEL CONTENIDO

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN Y OBJETIVOS

1.1. INTRODUCCIÓN

Hoy en día es habitual el uso del Smartphone y sus aplicaciones de forma cotidiana por la gran mayoría de la población. Ello conlleva que nosotros, como usuarios, cedamos una gran cantidad de datos a los dueños de dichas aplicaciones. A priori, debemos ser informados de qué usos se le dará a nuestra información y aceptar dichos usos. Sin embargo, ¿cumplen las aplicaciones con lo que aceptamos los usuarios o, por el contrario, se exceden en el uso de nuestros datos personales?

Una forma de saber si las aplicaciones realmente hacen lo que declaran es auditar sus comunicaciones. Sin embargo, habitualmente, estas comunicaciones se realizan a través de tecnologías como SSL/TLS, que permiten el establecimiento de comunicaciones cifradas entre un cliente y un servidor, impidiendo el acceso al contenido transmitido. Este protocolo garantiza la integridad y confidencialidad de las comunicaciones mediante el uso de certificados: cuando un cliente se intenta conectar a un servidor mediante SSL, solicita su identidad (un certificado) y, tras recibir una respuesta, verifica la validez del certificado digital del servidor y, si el certificado recibido es aceptable, establece la conexión cifrando los contenidos con la clave recibida en el certificado.

Sin embargo, la seguridad de esta comunicación se puede romper mediante ataques de intercepción (*Man in the middle*) combinados con la suplantación de los certificados que acepta el navegador o aplicación (por ejemplo, añadiendo el certificado de un servidor falso), haciéndole creer al cliente que el tráfico está dirigiéndose a un servidor legítimo cuando, en realidad, está transmitiendo toda la información a un servidor donde el atacante está escuchando.

Realizar este tipo de ataques no siempre es trivial, ya que las aplicaciones se han empezado a proteger empleando métodos como la fijación de certificados (*certificate pinning*). Esta defensa consiste en pre-configurar el cliente de tal forma que solo acepte determinados certificados como válidos. De esta forma, si el certificado recibido por parte del servidor está en la lista de certificados válidos, se permitirán las conexiones. Por el contrario, si no está en la lista, puede haber varias opciones: avisar al usuario, rechazar la conexión o permitir la conexión con funciones limitadas.

1.2. OBJETIVOS

El objetivo de este TFG es contribuir al desarrollo de soluciones para auditoría de aplicaciones móviles que protegen las comunicaciones con métodos como certificate pinning, mediante el desarrollo de un componente para la detección y circunvalación de los mecanismos de comprobación de certificados digitales en la plataforma Android. Para ello, se realizarán las siguientes actividades:

- Estudio de la literatura relacionada, para comprender los mecanismos de comprobación de certificados digitales para el establecimiento de comunicaciones seguras en la plataforma Android.
- Análisis del problema asociado a la interceptación de tráfico en aplicaciones con comprobación de certificados.
- Diseño y desarrollo de un componente que, dada una aplicación, permita circunvalar de forma automática los mecanismos de comprobación de certificados que pudiera usar.
- Desarrollo de un banco de pruebas que comprueben la efectividad de la solución desarrollada.
- Documentación del trabajo realizado.

2. FUNDAMENTOS

A lo largo de esta sección se va a hacer una breve introducción a los conceptos necesarios para poder entender el método posteriormente explicado. El objetivo de este trabajo es desarrollar un componente que sea capaz de interceptar las conexiones que establecen las aplicaciones Android y ser capaces de analizarlas. Por tanto, a continuación, se explicará cómo se establece una conexión http y cómo posteriormente, se mejoró el protocolo implementando un cifrado que dificultara el análisis de las comunicaciones. Dicho cifrado se realiza a través de certificados, un concepto que también es necesario conocer para poder entender el funcionamiento de las comunicaciones seguras. Existen diferentes formas de establecer comunicaciones cifradas en Android, por lo que es necesario conocerlas para saber cómo se pueden auditar. También es necesaria una introducción al análisis dinámico de código, ya que esta técnica está en el núcleo de la solución desarrollada. Por último, es necesario mencionar las diferentes herramientas que se han utilizado para el desarrollo de un componente que automatiza el método diseñado.

2.1. SEGURIDAD DE COMUNICACIONES WEB

Es habitual el uso diario de Internet a través de dispositivos móviles para realizar las más variadas gestiones: desde consultas hasta transacciones bancarias y compras. Gran parte de estas operaciones llevan asociados intercambios de información confidenciales como, por ejemplo, el envío de contraseñas o claves. Es por ello necesario garantizar la seguridad de las conexiones realizadas. La transferencia de datos por internet se realiza gracias al protocolo HyperText Transfer Protocol (HTTP), diseñado a principios de la década de los 90. Este protocolo está basado en operaciones de solicitud/respuesta entre cliente y servidor: el cliente establece conexión con el servidor enviando los datos de la solicitud y este responde con un mensaje similar. [1]

Las peticiones HTTP están formadas por diferentes campos:

- Método: existen varios entre los que destacan GET o POST para solicitar o enviar información, respectivamente.
- Dirección del recurso (URL).
- Versión de HTTP.
- Cabeceras opcionales y cuerpo del mensaje.

Las respuestas también las componen diferentes campos, entre los que resalta el campo del código del estado ya que es donde se indica si la petición ha sido exitosa (200 OK) o hubo errores, como el 404 (no se encuentra el contenido solicitado).

El problema de este protocolo es que la información viaja en claro, por lo que cualquiera que esté escuchando el intercambio de información puede descifrar fácilmente información transmitida. Este supuesto que acabamos de plantear se conoce como ataque *Man in the middle* (MITM) y consiste en lo siguiente: un atacante intercepta las comunicaciones entre un punto A (cliente) y un punto B (servidor), quedando las conexiones de la siguiente forma:

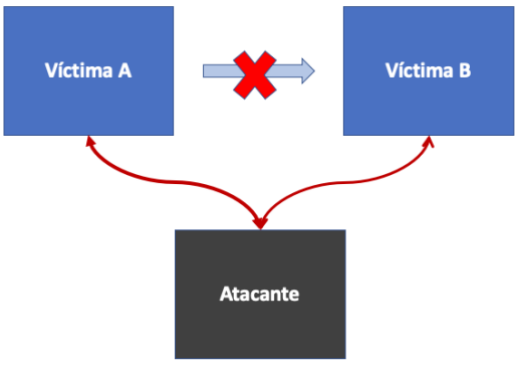

Fig. 1. Ataque Man in the Middle

Si las comunicaciones van en claro, como ocurre en el caso de HTTP, el atacante es capaz de entender la información obtenida. Sin embargo, si las comunicaciones van cifradas, el atacante tan solo obtendrá una cadena ininteligible, por lo que no podrá utilizar ninguna información obtenida.

Un ejemplo de conexión interceptada es el siguiente:

Fig. 2. Obtención de datos mediante un ataque MITM

En esta imagen podemos observar los datos enviados a través de un formulario de acceso: el correo electrónico (sergio****@gmail.com) y la contraseña introducida (123456).

Como solución a este problema surge un nuevo protocolo: HTTPS (HTTP Secure). En este protocolo se crea un canal cifrado para enviar la información basándose en SSL/TLS (Secure Sockets Layer / Transport Layer Security), por lo que una persona que realice un ataque MITM tan solo obtendrá una cadena cifrada, y por lo tanto ininteligible.

Para cifrar la información, se emplean algoritmos de cifrado de clave asimétrica, que funcionan utilizando dos claves que pertenecen a la misma entidad: la clave pública (accesible para todo el mundo) y la clave privada (solo conocida por la entidad poseedora). Estas claves se caracterizan porque si se cifra un mensaje con una de las claves (pública), únicamente puede ser descifrado por la otra (privada), de ahí la importancia de mantener la clave privada a salvo.

Con la implementación de HTTPS el intercambio de información se realiza de la siguiente manera: el cliente solicita al servidor una conexión segura, por lo que el servidor debe mostrar su certificado digital antes de proseguir con la conexión. En el caso de que el cliente acepte el certificado, se establece una conexión segura basada en un cifrado de clave pública/clave privada: con el certificado del servidor (clave pública) el cliente puede cifrar la información que envía de tal forma que tan solo el servidor sea capaz de descifrarlo con su clave privada. [2]

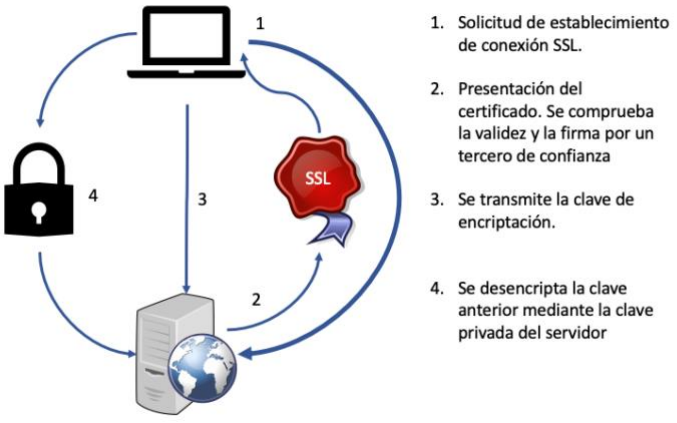

Fig. 3. Establecimiento conexión SSL

No obstante, la seguridad 100% no existe y SSL/TLS no es la excepción. Desde la implementación de este protocolo se han reportado numerosas vulnerabilidades [3] como POODLE o Heartbleed. Estas vulnerabilidades se aprovechan del uso de versiones obsoletas del protocolo, fallos en el algoritmo de encriptación (RC4) o incluso de errores en OpenSSL, un software libre que permite implementar el protocolo SSL/TLS y que es utilizado en la mayoría de los servidores web de Internet. Aunque ya fueron solucionadas con diferentes actualizaciones, conviene mencionar cómo se explotaban las más conocidas.

En el caso de POODLE, el ataque consiste en rebajar la versión de SSL a emplear en las conexiones para reducir el número de bits del cifrado. De esta forma, por fuerza bruta (probar todas las posibilidades) se puede conseguir descifrar la información en un tiempo relativamente corto, ya que la fortaleza del cifrado se ha visto reducida.

Por otra parte, en el caso de Heartbleed, el fallo se encuentra en la gestión de los mensajes "keep-alive" (mensajes para evitar el cierre de la conexión). En el funcionamiento esperado el cliente envía un mensaje con un pequeño contenido que el servidor devuelve automáticamente para indicar que la conexión se mantiene abierta. Sin embargo, puede darse el caso de que el mensaje keep-alive enviado por el cliente puede tener una carga que intente engañar al servidor. Verbigracia: si se manda un mensaje con un contenido de 1 byte al servidor, pero se especifica que tiene un tamaño de 200 bytes, cuando el servidor responda con 200 bytes, la respuesta contendrá el byte recibido y 199 bytes extras que contendrán información privada de la memoria del servidor pudiendo incluir información valiosa como las claves privadas.

2.2. CERTIFICADOS

Con la explicación dada anteriormente podemos concebir el certificado SSL como el documento de autenticación de un servidor. Así como un DNI consta de nombre y apellidos, el equipo de expedición del DNI o fecha de caducidad, en el certificado se pueden encontrar datos como el titular, la caducidad o la autoridad que ha expedido dicho certificado. [4]

Existen varios tipos de certificados: [5]

- **Certificados autofirmados:** certificado generado por una entidad sin respaldo externo. No sirven para autenticar su identidad real.
- **Certificado raíz de CA:** certificado generado por una Autoridad de Certificación (CA – *Certification Authority*), que es una entidad independiente y confiable. Este certificado servirá para firmar y comprobar los certificados de otras entidades, validando de esa forma su identidad.
- **Certificado intermedio o subordinado:** certificado perteneciente a una entidad reconocida por una CA para emitir certificados a terceras partes.
- **Certificado con validación externa:** certificado garantizado por una entidad de confianza externa (la CA o una entidad subordinada, que ha identificado al poseedor del certificado) y una firma encriptada con la clave privada del poseedor (para el establecimiento de las conexiones seguras).

Para garantizar la confiabilidad en el certificado, se establecen cadenas de confianza, generándose así una relación de certificados que permiten asegurar que dicho certificado ha sido emitido por una autoridad (CA). Las autoridades disponen de un certificado raíz, el cual validará cada certificado emitido por la CA. Con él se firman solamente los certificados subordinados y son éstos los que firmarán los certificados finales o de suscriptor, siguiendo el siguiente esquema:

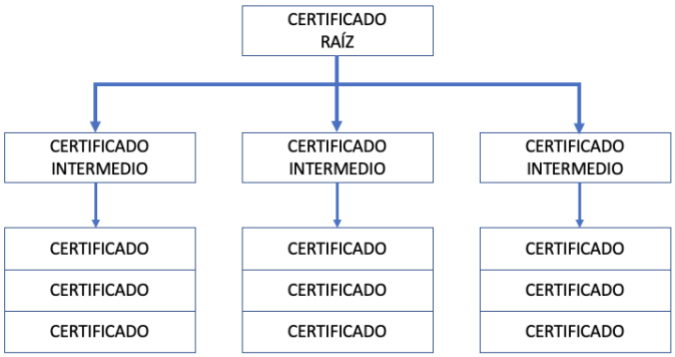

Fig. 4. Cadena de certificados

Al seguir este modelo, es necesario proteger al máximo el certificado raíz, ya que es a partir del cual se generan el resto de los certificados. Si un tercero consiguiera la clave privada del certificado raíz, estarían comprometidos todos los generados a partir de ese certificado raíz. Es por ello que los certificados finales no están firmados por el certificado raíz y las CA emplean grandes medidas de seguridad para proteger la clave privada de dicho certificado. Un ejemplo de esta cadena de confianza se puede encontrar en la página web de la UPM, donde encontramos los siguientes certificados:

- Certificado raíz: emitido por DigiCert.
- Certificado intermedio: emitido por DigiCert para Terena.
- Certificado de suscriptor o final: www.upm.es, emitido por Terena (entidad subordinada).

Fig. 5. Ejemplo de cadena de certificados

Actualmente, las CA tienen dos formas complementarias de ofrecer a los usuarios mecanismos de validación de certificación actualizada: Listas de certificados revocados (CRL, por sus siglas en inglés) y el protocolo OCSP (*Online Certificate Status Protocol*).

Las CRL, como su nombre indica, son listas de certificados revocados firmadas por la CA correspondiente. El inconveniente de este método es que desde la última actualización de la lista hasta la fecha en la que se consulta pueden haberse revocado certificados que aun no estén incluidos en la CRL. Debido a esta problemática, surge OCSP: un protocolo que permite realizar consultas en tiempo real obteniendo como respuestas bueno, revocado o desconocido. El proceso de validación de certificados lo suelen realizar automáticamente los navegadores web, avisando al usuario en el caso de encontrar alguna irregularidad. [5] [6]

Este modelo de cadenas de confianza y servicios de control de validez de certificados es conocido como PKI (*Public Key Infrastructure*), definida en la RFC 4949 como el conjunto de elementos necesarios para crear, almacenar, distribuir y revocar certificados digitales basados en el cifrado de clave asimétrica.

Volviendo al caso del ataque MITM, parece que una vez que la información viaja cifrada es imposible sacar beneficios de este tipo de ataque, salvo por los casos mencionados anteriormente que fueron resueltos una vez detectados.

Sin embargo, el atacante todavía puede situarse entre cliente y servidor antes de que se establezca la conexión segura, es decir, cuando el cliente solicita la conexión y recibe el certificado del servidor. De esta forma, el atacante puede enviar un certificado falso al cliente, haciéndose pasar por el servidor real. Por tanto, las comunicaciones entre cliente y atacante irán cifradas, pero el atacante es capaz de leerlas gracias a su clave privada, ya que el cifrado se ha realizado con la clave pública falsa que ha facilitado él mismo al cliente. Por otra parte, el atacante, al estar situado en el medio, también dispone del certificado del Servidor, por lo que puede cifrar la información en claro del cliente y enviársela al servidor, manteniendo de esta forma la comunicación entre cliente y servidor a la vez que la observa.

Por tanto, volvemos a la situación inicial, donde la información puede estar comprometida sin que prácticamente ni el cliente ni el servidor se enteren. Es aquí donde surge la necesidad de realizar mayores y mejores comprobaciones a los certificados en los que el cliente confía para el establecimiento de conexiones seguras.

2.3. SEGURIDAD DE LAS COMUNICACIONES EN ANDROID

En Android, el proceso de validación de certificados es ligeramente diferente entre versiones del sistema operativo. En Android 6.0 y versiones anteriores, los terminales poseen dos listas de certificados considerados válidos: los preinstalados, que ha ido aumentando conforme se han sucedido las actualizaciones, y los instalados por el usuario. En estas versiones, una aplicación, a la hora de establecer una conexión, comprueba si la CA del certificado que está recibiendo se encuentra en alguna de las dos listas. En caso afirmativo, permite la conexión. Si no está en ninguna de las listas, se arrojará una excepción. Esto conllevaba un gran peligro, ya que cualquier usuario malicioso puede generar su propio certificado e instalarlo en el dispositivo, siendo así admitido un certificado cualquiera como certificado de confianza, permitiendo de esta forma las conexiones a un servidor con dicho certificado.

A partir de Android 6.0 las aplicaciones no admiten los certificados instalados por el usuario como certificados de confianza. A pesar de ello, siguen existiendo vulnerabilidades que pueden comprometer el establecimiento de conexiones seguras.

Por una parte, se puede instalar un certificado y añadirlo a la lista de certificados que tiene Android por defecto. Para realizar esto, es necesario *rootear* el teléfono, es decir, disponer de permisos de superusuario. Esta opción puede parecer interesante, pero muchas aplicaciones no permiten ejecutarse completamente en teléfonos *rooteados*, por lo que no se podrían estudiar ya que no permitirían trabajar con ellas.

Por otra parte, todas las aplicaciones están formadas, entre otros, por un archivo xml (AndroidManifest) en el que se define gran cantidad de información: permisos de la aplicación, idiomas, compatibilidades, versiones… [7]

Una de las formas de evitar las nuevas restricciones en el uso de certificados de Android es modificar este archivo XML para que la aplicación sea soportada por una versión de Android inferior a la 6.0. De esta forma nos situaríamos de nuevo en el primer caso.

También se puede declarar en el manifiesto la existencia de un nuevo archivo: network_security_config.xml. En dicho archivo se definirán las conexiones admitidas, habiendo varias formas de actuar: aceptar sólo los certificados que se encuentran en la lista de certificados válidos de Android, admitir sólo los certificados que se definan en el manifiesto, o añadir ambos, tanto los de Android como los que se añadan en el archivo network_security_config.xml. Por defecto, solo se aceptan los certificados del sistema. Para definir este comportamiento, el contenido del network_security_config.xml debe ser: [8]

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <base-config>
```

```
            <trust-anchors>
                <certificates src="system"/>
            </trust-anchors>
        </base-config>
    </network-security-config>
```

En el fragmento de código anterior observamos que los certificados en los que se confía (trust-anchors) provienen de la fuente (src, del inglés *source*) "system" (sistema). Al ser este el comportamiento por defecto no hace falta incluir un archivo que indique que solo se confía en certificados del sistema.

Gracias a esta forma de actuar, se complica el establecimiento de conexiones SSL fraudulentas, aunque no se imposibilita. Si se añade o modifica el archivo network_security_config.xml, se indica su existencia en el archivo AndroidManifest.xml y se vuelve a compilar la aplicación, se puede añadir un certificado fraudulento, permitiendo así conexiones no deseadas en las que la información se ve comprometida, por ejemplo, de cara a auditar su contenido.

Esta forma de intentar proteger las conexiones SSL mediante pre-determinar o fijar los certificados admitidos se conoce como "certificate pinning" (fijar certificados).

No solo se puede realizar en el archivo xml, sino que existen bibliotecas que nos permiten implementarlo para poder ser más específicos y tener un mayor control sobre la gestión de certificados de una aplicación. De esta forma se puede añadir un certificado propio a una aplicación, no dependiendo así de los certificados que propone Android por defecto. Con cada biblioteca se pueden fijar los certificados de forma diferente, por lo que a continuación se van a analizar las bibliotecas más comunes:

2.3.1. TRUST MANAGER

Los objetos TrustManager son los encargados de administrar el "material de confianza", como, por ejemplo, los certificados. Es una de las formas más empleadas para realizar el pinning ya que es la que viene explicada paso a paso en la documentación de Android. Tal y como se explica en la documentación, funciona de la siguiente manera: [9]

1. Se crea un objeto KeyStore que contiene las autoridades certificadoras en las que vamos a confiar.
2. Se crea un objeto TrustManager en el que se indica que vamos a confiar en las autoridades que hemos indicado previamente en el KeyStore.
3. Se inicia un SSLContext, que se utiliza para evitar los valores por defecto que trae Android y utilizar los configurados manualmente, es decir, confiar en lo que indique el objeto TrustManager que hemos creado previamente.
4. Realizar la conexión mediante HttpsURLConnection. Debemos prestar especial atención en utilizar https y no http URLConnection, ya que lo que queremos establecer es una conexión HTTPS y no HTTP.

En el ejemplo que viene proporcionado en la documentación de Android, la autoridad certificadora la introduce el usuario desde un certificado (*InputStream*) y se almacena de la siguiente forma:

```
    // Carga CAs desde una entrada de Texto InputStream
    // (puede ser desde otro recurso)
    CertificateFactory cf = CertificateFactory.getInstance("X.509");
    // De https://www.washington.edu/itconnect/security/ca/load-der.crt
    InputStream caInput = new BufferedInputStream(new
FileInputStream("load-der.crt"));
    Certificate ca;
    try {
        ca = cf.generateCertificate(caInput);
        System.out.println("ca=" + ((X509Certificate) ca).getSubjectDN());
    } finally {
        caInput.close();
```

```
    }

    // 1. Se crea un KeyStore con las entidades de confianza
    String keyStoreType = KeyStore.getDefaultType();
    KeyStore keyStore = KeyStore.getInstance(keyStoreType);
    keyStore.load(null, null);
    keyStore.setCertificateEntry("ca", ca);

    // 2. Se crea un TrustManager que confía en las CA del KeyStore
    String tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm();
    TrustManagerFactory tmf =
TrustManagerFactory.getInstance(tmfAlgorithm);
    tmf.init(keyStore);

    // 3 Se crea un SSLContext que confía en el TrustManager anterior
    SSLContext context = SSLContext.getInstance("TLS");
    context.init(null, tmf.getTrustManagers(), null);

    // 4 . Se indica al URLConnection que use el SocketFactory del
    // SSLContext
    URL url = new URL("https://certs.cac.washington.edu/CAtest/");
    HttpsURLConnection urlConnection =
        (HttpsURLConnection)url.openConnection();
    urlConnection.setSSLSocketFactory(context.getSocketFactory());
    InputStream in = urlConnection.getInputStream();
    copyInputStreamToOutputStream(in, System.out);
```

2.3.2. OKHTTP3

OkHTTP3 es una biblioteca que se emplea en Android para establecer conexiones HTTP o HTTPS. Entre las diferentes clases que trae esta biblioteca, la utilizada para realizar el pinning es CertificatePinner [10]. Esta clase tiene varias funciones, entre las que destaca la función *check*. Esta función recibe una lista de certificados y comprueba que al menos uno de los certificados está contenido en el objeto *peerCertificates*. También se puede implementar la función *check* pasando como argumento un único certificado, pero esta función está obsoleta, por lo que no es recomendable implementarla. En el caso de uno de los certificados de la lista (o, en el caso obsoleto, el certificado pasado como parámetro) coincida con alguno de los certificados contenidos en el objeto *peerCertificates*, no se realizará ninguna acción. Sin embargo, si ninguno de los certificados que recibe la función coincide con los que debería haber en el objeto *peerCertificates*, la función lanza una excepción, evitando de esta forma que se realice una conexión a un sitio considerado no seguro.

2.3.3. TRUSTKIT

TrustKit es una biblioteca de software libre cuya documentación encontramos en GitHub [11]. El objetivo de esta biblioteca es permitir realizar certificate pinning en versiones de Android inferiores a la 6.0 como si estuviéramos en versiones superiores en las que se implementa cierta comprobación de certificados por defecto.

El funcionamiento de TrustKit es diferente al de bibliotecas anteriores. En este caso, definimos una política de conexiones en el archivo network_security_config.xml (por tanto, la existencia de este archivo debe estar indicada en el manifest.xml como ya se ha mencionado anteriormente). En este archivo vamos a indicar los certificados que aceptamos y el uso de TrustKit. Cogiendo y simplificando el ejemplo de la documentación, el archivo network_security_config.xml quedaría de la siguiente forma:

```
<!-- res/xml/network_security_config.xml -->
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
```

```
  <!-- Fijar el dominio www.datatheorem.com -->
  <!-- Api oficial Android N -->
  <domain-config>
    <domain>www.datatheorem.com</domain>
    <pin-set>
      <pin digest="SHA-
256">k3XnEYQCK79AtL9GYnT/nyhsabas03V+bhRQYHQbpXU=</pin>
      <pin digest="SHA-
256">2kOi4HdYYsvTR1sTIR7RHwlf2SescTrpza9ZrWy7poQ=</pin>
    </pin-set>
    <!-- API Trust kit -->
    <trustkit-config enforcePinning="false">
      <!-- Añadir url para reportar la validación-->
      <report-uri>http://report.datatheorem.com/log_report</report-uri>
    </trustkit-config>
  </domain-config>
</network-security-config>
```

Posteriormente, hay que indicar en el código de la aplicación la existencia de esta política en un proceso conocido como inicialización. Una vez inicializada, se verificará que los certificados empleados en cada conexión cumplen la política indicada. Hay que tener en cuenta que TrustKit solo debe inicializarse una vez en todo el código y no por cada conexión que se cree.

Esta biblioteca no realiza las conexiones, simplemente comprueba la validez del certificado. Por tanto, ha de ser utilizada con otros métodos que realicen las conexiones como por ejemplo los ya mencionados HttpsURLConnection u OKHttp3.

Para ejemplificar este proceso de inicialización y posterior creación de la conexión, se empleará otro fragmento simplificado del código de la documentación. [11]

```
// Usando la ruta por defecto: res/xml/network_security_config.xml
  TrustKit.initializeWithNetworkSecurityConfiguration(this);

  URL url = new URL("https://www.datatheorem.com");
  String serverHostname = url.getHost();

  // HttpsUrlConnection
  HttpsURLConnection connection = (HttpsURLConnection)url.openConnection();

connection.setSSLSocketFactory(TrustKit.getInstance().getSSLSocketFactory(s
erverHostname));
```

2.3.4. APPCELERATOR

Appcelerator [12] consiste en un framework de código libre que permite crear aplicaciones nativas [13] desde un código desarrollado en el lenguaje JavaScript. Las aplicaciones nativas son aquellas desarrolladas específicamente para el sistema operativo en el que van a ser utilizadas, permitiendo una mejor experiencia de uso, pero aumentando el coste de desarrollo si se quiere realizar una aplicación multiplataforma ya que sería necesario desarrollar diferentes códigos para cada sistema operativo.

A la hora de hacer el pinning, el framework Appcelerator emplea el objeto *PinningTrustManager*, que realiza una función similar al TrustManager que comentamos previamente. Ese objeto se comprueba con la función *checkServerTrusted*, que funciona de manera similar a la función *check* vista previamente en OkHttp3.

2.4. ANÁLISIS DE CÓDIGO

El análisis de una aplicación se puede realizar de dos formas: estática y dinámica. La principal diferencia consiste en que en el análisis estático se estudia el código fuente de un programa sin ejecutarlo, mientras que el análisis dinámico consiste en analizar el comportamiento de un proceso mientras se está ejecutando. Dentro del análisis dinámico de software, hay dos tipos de pruebas:

- Caja negra: se comprueba que las salidas son correctas, sin prestar atención a la forma de llegar a dichas salidas.
- Caja blanca: se realizan todas las entradas posibles para obtener una salida concreta.

Para ambos tipos de análisis existen herramientas que pueden facilitar el proceso, ya que ambos procesos pueden resultar largos y costosos. En el caso del análisis estático, algunas herramientas como Jadx permiten obtener el código java de un archivo (como, por ejemplo, un apk), mientras que en el caso del análisis dinámico un ejemplo de herramientas es Introspy, que permite seleccionar qué características de una aplicación se van a monitorizar y posteriormente avisa cada vez que una aplicación haga uso de dichas características previamente configuradas.

En este trabajo aplicaremos ambas formas de análisis para poder llevar a cabo el objetivo de analizar las comunicaciones que establecen las aplicaciones Android.

En primer lugar, se usará el análisis dinámico con el objetivo de variar el comportamiento de las aplicaciones según se están ejecutando. Esto permite, entre otras cosas, modificar el resultado de la función que compruebe la validez de los certificados. De esta forma, cuando una aplicación usa un certificado que debe reconocer como extraño, se puede variar el resultado de la función que debe invalidar dicho certificado, haciendo que lo acepte como válido, estableciendo así una conexión sin problemas. [14]

Un ejemplo práctico es el siguiente: si la función que realiza la comprobación de certificados devuelve *true* cuando un certificado es válido, y prosigue la ejecución realizando una conexión, pero devuelve *false* si el certificado es inválido, deteniendo la ejecución de la conexión y el objetivo del usuario es utilizar un certificado desconocido, lo único que tendrá que hacer es modificar la función cuando se ejecuta para que la variable anteriormente mencionada sea *true* cuando detecta cualquier certificado. De esta forma, se evitaría la detención de la conexión.

Para poder variar el comportamiento de una función según se ejecuta, es necesario conocer qué funciones va a ejecutar la aplicación que se desea instrumentar. Es posible intentar instrumentar las funciones más habituales, que son las que se han mencionado anteriormente, pero esto puede dar lugar a no encontrar dichas funciones en el código de la aplicación que se está estudiando, ya que no son las únicas que existen. Para poder realizar la instrumentación dinámica con un mayor porcentaje de éxito, además del análisis dinámico conviene realizar un análisis estático de código. De esta forma es posible conocer la aplicación y saber qué librerías va a utilizar, qué excepciones va a lanzar y qué código se debe modificar, de cara a alterar el comportamiento de dichas funciones una vez que se estén ejecutando. Sin embargo, el análisis estático no siempre es posible ya que en algunos casos el código está ofuscado para dificultar que cualquier persona pueda entenderlo.

También es necesario el análisis estático de código para conocer el contenido del archivo AndroidManifest.xml de cada aplicación. Este contenido se modificará, al igual que se modificará el contenido del archivo network_security_config.xml, en el caso de que exista, para poder evitar el certificate pinning que realiza Android de manera predeterminada.

El análisis del código del archivo AndroidManifest.xml es de gran importancia, ya que en este archivo se encuentran las funcionalidades de la aplicación, sus permisos y principales características. Debido a la importancia que tiene este archivo, su código siempre deberá estar en claro, ya que es necesario que el resto de los componentes entiendan el contenido de este archivo. Por esta razón, entender este archivo es de vital importancia a la hora de auditar una aplicación Android.

2.5. HERRAMIENTAS DE TRABAJO

En este punto se expondrán brevemente las herramientas que se utilizarán para detectar y auditar las conexiones https realizadas desde aplicaciones Android.

2.5.1. SISTEMA OPERATIVO PARROT

Parrot es una distribución de Linux enfocada a la seguridad informática [15]. Por ello trae instaladas por defecto diferentes herramientas empleadas para análisis forense, auditorías o análisis de redes, entre otros. En primer lugar, se escoge una distribución Linux para el entorno de trabajo porque ser software libre. En segundo lugar, es muy fácil de instalar en un entorno virtualizado, ya que desde la página web se puede descargar el archivo que contiene la máquina virtual ya creada. Otra ventaja que presenta Parrot es que cuenta con un intérprete de Python ya instalado, que es el lenguaje en el que se han desarrollado los componentes que se describirán más adelante, por lo que facilita también la realización de pruebas del script. Además, también trae instaladas otras herramientas necesarias como son Jarsigner y Keytool, cuya utilidad se explica más adelante y es más sencillo llevar a cabo la configuración de la máquina como proxy en Parrot que en otros sistemas operativos (Mac, Windows), lo que hace a este sistema operativo idóneo para utilizarlo como entorno de trabajo para este proyecto.

2.5.2. SISTEMA OPERATIVO ANDROID X86

Android-x86 es un proyecto de software libre que permite emular un dispositivo Android en un ordenador. Será en este emulador donde se instalen las diferentes aplicaciones con las que se llevarán a cabo las pruebas de los componentes desarrollados. Desde su página web se pueden encontrar diferentes versiones de Android. [16]

2.5.3. VIRTUALBOX

VirtualBox es un software de virtualización desarrollado por Oracle. Será en VirtualBox donde se instale el sistema operativo Parrot por una parte y el sistema operativo Android por otra y se cree el entorno de trabajo definido posteriormente. De esta forma se podrá desplegar fácilmente el entorno de validación del componente desarrollado.

2.5.4. APKTOOL Y ADB

Apktool [17] y adb son dos herramientas diferentes que se utilizarán para gestionar los archivos apk de las aplicaciones Android que se estudien. En el caso de disponer de adb, será necesario instalar apktool, no siendo necesario instalar ambas herramientas si la primera que se instala es apktool ya que adb viene incluido en esta última.

En un primer lugar, apktool se utiliza para la obtención del código de una aplicación (descompilar). Es importante mencionar que con Apktool no se obtiene el código Java tal cual, sino que se obtiene una representación en un lenguaje conocido como Smali. Sin embargo, como no se modificará gran parte de código, será útil igualmente. Apktool también puede utilizarse para la generación del archivo apk a partir del código Smali (compilación).

En segundo lugar, adb (*Android Debug Bridge*) es una herramienta que permite realizar operaciones en un sistema operativo Android mediante línea de comandos. Entre las operaciones que permite

realizar destacan: instalar y desinstalar apk, obtención de apk a partir de una aplicación ya instalada en el dispositivo Android conectado, y la transferencia de archivos entre el dispositivo Android y la máquina a la que está conectado.

2.5.5. FRIDA

Frida [18] es un conjunto de herramientas (de software libre) de instrumentación dinámica. Esto quiere decir que permite observar y modificar procesos en ejecución, haciendo de esta forma posible la modificación, en tiempo de ejecución, de los flujos de ejecución previstos en el código de la aplicación. De esta forma se puede variar el comportamiento que presentan las aplicaciones a la hora de realizar una conexión segura para facilitar la auditoría de los contenidos enviados.

Una vez instalado Frida, hay que indicarle cuál es el código que se quiere modificar. Frida es lanzado desde línea de comandos y mediante el parámetro load (-l) permite cargar un programa personalizable. Es en dicho programa en el que hay que especificar qué funciones se van a instrumentar y su correspondiente modificación.

Es importante destacar que Frida dispone de un repositorio (https://codeshare.frida.re/) en el que diferentes personas suben los scripts que emplean indicando sus funcionalidades y objetivos. En este repositorio podemos encontrar varios programas para evitar el pinning en Android. Tal y como se ha mencionado anteriormente, el script que carga Frida es plenamente personalizable, por lo que se pueden reutilizar y combinar elementos de diferentes scripts para poder hacer un programa más completo y que cubra un mayor abanico de posibilidades.

2.5.6. MITMPROXY

Mitmproxy [19] es otra herramienta de código abierto gracias a la cual se pueden interceptar las conexiones HTTP y HTTPS. Con esta herramienta será posible interceptar las conexiones establecidas desde una aplicación, leyendo así el tráfico que transmite y recibe, posibilitando la auditoría del contenido.

Este software tiene varias formas de trabajar. La más interesante para llevar a cabo la auditoría consiste en emplear el comando *mitmdump*, que permite iniciar mitmproxy cargando un fichero de configuración. En dicho fichero se pueden definir diferentes comportamientos como elegir qué conexiones auditar, qué datos enviar a la salida estándar o a un archivo, y multitud de opciones más.

Para este trabajo se utilizará una configuración que registra todas las conexiones que realiza la aplicación que se esté estudiando en cada momento, de cara a ser analizadas posteriormente.

2.5.7. JARSIGNER Y KEYTOOL

Para poder instalar un archivo apk en un dispositivo Android, es necesario que esté firmado. Para ello se hará uso de las herramientas Jarsigner y Keytool, instaladas por defecto en la distribución Parrot. Jarsigner permite firmar un archivo y Keytool permite gestionar los diferentes certificados empleados para firmar.

En la siguiente tabla se ofrece un resumen de las herramientas mencionadas y su función:

Herramienta	Función
S.O. Android	Emular dispositivo Android
S.O. Parrot	Proxy
VirtualBox	Crear entorno de trabajo
Apktool + Adb	Gestión archivos apk
Frida	Instrumentación dinámica
Mitmproxy	Visualización de conexiones
Jarsigner + Keytool	Firma de apk y gestión de certificados

Fig. 6. Resumen de las herramientas empleadas

3. MÉTODO

A lo largo de este trabajo se han expuesto varias formas que existen para hacer que la información de las conexiones que realizan las aplicaciones Android se transmita de forma cifrada.

El objetivo de este trabajo es diseñar y automatizar un método con el que se puedan detectar y deshabilitar los mecanismos con los que las aplicaciones protegen el contenido de sus comunicaciones, de tal forma que se pueda leer en claro la información transmitida. En particular, el método se basará en realizar una suplantación de certificados e intercepción de las comunicaciones mediante un proxy interpuesto, eliminando las protecciones que puedan haber habilitado las aplicaciones para protegerse de esta situación.

Primero se explicará el entorno de trabajo en el que se han realizado las diferentes pruebas y se han desarrollado los diferentes métodos, posteriormente se desarrollará la metodología seguida en cada escenario y se terminará con una explicación de los componentes software realizados para facilitar el trabajo que se ha de realizar.

3.1. ENTORNO DE TRABAJO

Para auditar las conexiones que realiza una aplicación se desplegará un entorno compuesto por dos máquinas virtuales creadas con VirtualBox, con las siguientes características:

- Una máquina virtual (Android) con sistema operativo Android86 versión 7.1. En esta máquina se instalarán y utilizarán las aplicaciones Android auditadas. También será necesario que en esta máquina esté el servidor de Frida, cuya instalación se explica más adelante.
- Una máquina virtual (Proxy):
  - Sistema operativo Parrot que actuará como proxy y permitirá monitorizar las conexiones que establecen las aplicaciones Android.
  - Apktool y Adb para manejar los archivos apk de las aplicaciones que se quieren auditar.
  - Mitmproxy para ser capaces de leer las conexiones que establecen las aplicaciones.
  - Frida para instrumentar las aplicaciones dinámicamente, intentando evitar así los mecanismos de seguridad que implementan a la hora de establecer conexiones.
  - Jarsigner y Keytool para firmar los archivos apk cuyo código se haya modificado para intentar evitar el certificate pinning.

Dicho escenario se configurará de la siguiente manera: la máquina virtual Android se conectará a internet a través del Proxy, el cual dispondrá tanto de conexión a internet y podrá conectarse a la máquina Android mediante adb. Además, el Proxy será capaz de interceptar las comunicaciones de Android gracias a mitmproxy. El esquema es el siguiente:

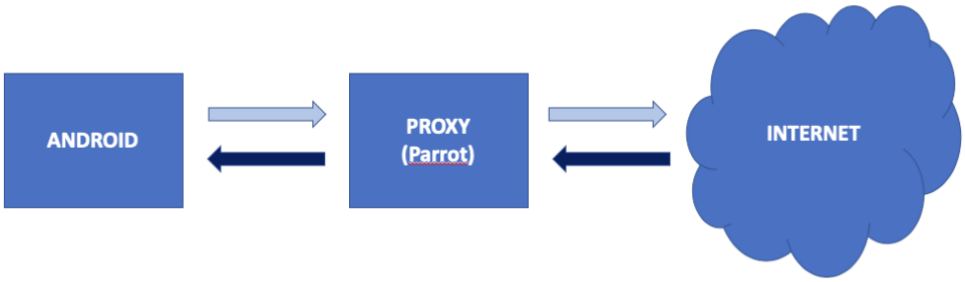

Fig. 7. Escenario de trabajo

Para que la máquina virtual Android tenga conexión y para ser capaces de ver el tráfico cursado es necesario tener abierto en el Proxy el programa Mitmproxy y configurar la red para redirigir el tráfico de Android por el proxy. Para configurar la intercepción en el Proxy es posible desarrollar un script siguiendo la documentación [20] de Mitmproxy.

Tras la instalación de mitmproxy y la configuración de la red, para poder terminar de configurar el entorno de trabajo es necesario instalar el certificado de mitmproxy en la máquina virtual de Android, es decir, realizar la suplantación de certificados. Para ello, con las configuraciones anteriormente realizadas, se accede en la máquina Android a través un navegador (Chrome, por ejemplo) a la página web mitm.it/cert/pem y, si la configuración de la red es correcta, el certificado se descargará automáticamente. Una vez descargado e instalado el certificado de mitmproxy, si se consigue que las aplicaciones usen este certificado para establecer las conexiones seguras el tráfico generado se verá en claro desde la consola de mitmproxy en la máquina Parrot. Es decir, se podrá auditar el contenido de las comunicaciones.

Para comprobar la instalación del certificado se puede buscar en la pestaña de "certificados de usuario" de la máquina Android. Cabe mencionar que para añadir un certificado propio a la lista de certificados de usuario en la máquina Android es necesario definir una forma de desbloqueo segura ya sea por patrón o clave.

Por último, también es necesario instalar las herramientas de Frida en la máquina Proxy. Esto se realiza mediante el comando:

```
pip install frida-tools
```

3.2. ESCENARIO I: PINNING POR DEFECTO

A partir de Android 6.0 las aplicaciones bloquean cualquier conexión que se realiza a un servidor cuyo certificado no está en la lista de certificados pre-instalados en la distribución de Android. En versiones anteriores las aplicaciones admitían certificados instalados por el usuario, sin embargo, ahora es necesario especificar en el código que se pueden utilizar certificados instalados manualmente para realizar conexiones https. [21]

Esta forma de realizar certificate pinning es bastante insegura: si en una aplicación no se implementan mecanismos extra, bastará con modificar levemente el código permitiendo así que se tomen como válidos los certificados instalados por el usuario. Esto se puede realizar con la aplicación apktool, que ya se mencionó en el apartado anterior. El uso de adb permite facilitar el trabajo, pero no hace falta instalarlo una vez que se instala apktool ya que viene incluido.

Por lo tanto, el método a seguir en este escenario consistirá en obtener el código de la aplicación con la que deseamos trabajar, modificarlo y volver a instalar la aplicación en un dispositivo Android. [22]

El archivo con el que se empezará a trabajar es el apk de la aplicación en cuestión (es decir, el código ya compilado y preparado para que la aplicación se pueda utilizar desde un terminal Android) y el entorno de trabajo es el definido en el punto anterior. Una vez se disponga de dicho entorno, los pasos a seguir son los siguientes:

1. Se extrae el archivo apk de la aplicación instalada en Android. Para ello hay que obtener el id de la aplicación, la ruta en la que está instalada y posteriormente sacarlo. Los comandos que se deben ejecutar desde la máquina Proxy son los siguientes:

```
adb connect IPMaquinaAndroid
adb shell pm list packages | grep nombreAPP
adb shel pm path com.ejemplo.miaplicacion
adb pull /data/app/com.ejemplo.miaplicacion.apk
```

2. Una vez obtenido el archivo apk, procedemos a obtener el código de la aplicación mediante apktool. Para descompilar hay que lanzar apktool pasándole como parámetro el archivo apk. También se puede indicar una ruta para la salida, pero en caso de no hacerlo se creará un directorio con el mismo nombre que la aplicación en la misma ruta en la que está el apk. El comando es el siguiente:

```
apktool d archivo.apk
```

Al acceder al nuevo directorio se pueden observar diferentes archivos y subcarpetas. Los más importantes son el AndroidManifes.xml y el directorio de recursos "res". Para la modificación de código se deberá añadir el archivo network_security_config.xml en la ruta /res/xml. El contenido de dicho archivo para permitir todos los certificados viene detallado en la documentación de Android y es el siguiente.

```
<?xml version="1.0" enconding="utf-8"?>
<network-security-config
xmlns:android="http://schemas.android.com/apk/res/Android">
        <base config>
                <trust-anchors>
                        <certificates src="system"/>
                        <certificates src="user"/>
                </trust-anchors>
        </base-config>
</network-security-config>
```

3. Una vez añadido el archivo network_security_config.xml, es necesario indicar a la aplicación la existencia de dicho archivo. Para ello hay que modificar el archivo AndroidManifest.xml, localizado en el directorio raíz. La línea que se debe añadir en el apartado <application> es la siguiente:

```
android:networkSecurityConfig="@xml/network_security_config"
```

4. Una vez terminadas las modificaciones a realizar en el código, es necesario obtener un nuevo archivo apk en el que estén reflejadas las modificaciones que se han realizado. Para ello se vuelve a utilizar apktool. En esta ocasión sí es recomendable indicarle la ruta en la que se desea obtener el archivo apk. Simplemente indicando un nombre en el comando, la ruta de salida será el mismo directorio en el que se lanza. El comando requerido para construir el archivo es:

```
apktool b carpetaCodigoModificado -o NuevoArchivo.apk
```

Para que este nuevo archivo apk se pueda instalar y utilizar en un dispositivo Android, es necesario que esté firmado con un certificado. En este caso, no hay problema en que sea un certificado autofirmado. Para crear una clave con la que firmar el certificado se puede utilizar la herramienta keytool, que viene por defecto en algunas distribuciones Linux como Parrot. Una vez creada la clave, se firma el apk utilizando jarsigner, que también se encuentra disponible en Parrot. Los dos comandos necesarios para la creación de claves y firma del apk son: [23]

```
keytool -genkey -v -keystore mi-clave.keystore -alias
alias_clave -keyalg RSA -keysize 2048 -validity 10000
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -
keystore mi-clave.keystore NuevoArchivo.apk alias_clave
```

Posteriormente, se instala el apk en la máquina virtual Android. Para ello se debe utilizar otra vez adb mediante el comando:

```
adb install NuevoArchivo.apk
```

El último paso es abrir la aplicación en la máquina virtual Android mientras está abierto mitmproxy en el Proxy e intentar navegar por ella. Si la aplicación funciona con normalidad y en el Proxy se pueden visualizar las conexiones que realiza, quiere decir que en la aplicación que se modificó no existía ningún mecanismo para realizar certificate pinning más allá del que facilita Android de forma automática.

Este proceso manual es fácil de llevar a cabo cuando se quiere analizar una única aplicación. Sin embargo, puede resultar imposible de realizar para un gran número de archivos apk. Por ello se ha desarrollado un componente) que realiza de forma automática todo el trabajo. El componente consta de dos archivos:

1. Programa desarrollado en Python que recibe como parámetro un archivo de texto. En dicho archivo de texto se puede incluir un apk por cada línea y cada línea debe contener los siguientes parámetros separados por punto y coma:

   ArchivoAPK;KeyStore;AliasKeyStore

Cabe mencionar que los archivos apk y el archivo KeyStore que se indican como parámetro deben estar en la misma ruta que el componente desarrollado.

2. Un archivo network_security_config.xml configurado de tal forma que se confíe en los certificados instalados por el usuario. El contenido de este archivo es el visto en el punto 2 anterior.

Para preparar el archivo de texto de forma sencilla, la recomendación es utilizar una hoja de cálculo y un procesador de texto. Teniendo en cuenta que tanto KeyStore y AliasKeyStore se pueden repetir para todos los archivos, es fácil de preparar una hoja de cálculo de tres columnas con todos los datos, copiarlo en un procesador de texto y reemplazar las tabulaciones por puntos y comas.

Una vez llamado al componente y pasándole el archivo de texto que contendrá todos los parámetros, el propio componente se encargará de:

1. Descompilar apk en la carpeta de salida indicada
2. Copiar archivo network_security_config.xml en la carpeta /res/xml.
3. Modificar AndroidManifest.xml
4. Compilar apk en un nuevo archivo que se llama ArchivoApkMOD.apk
5. Firmar el nuevo apk.

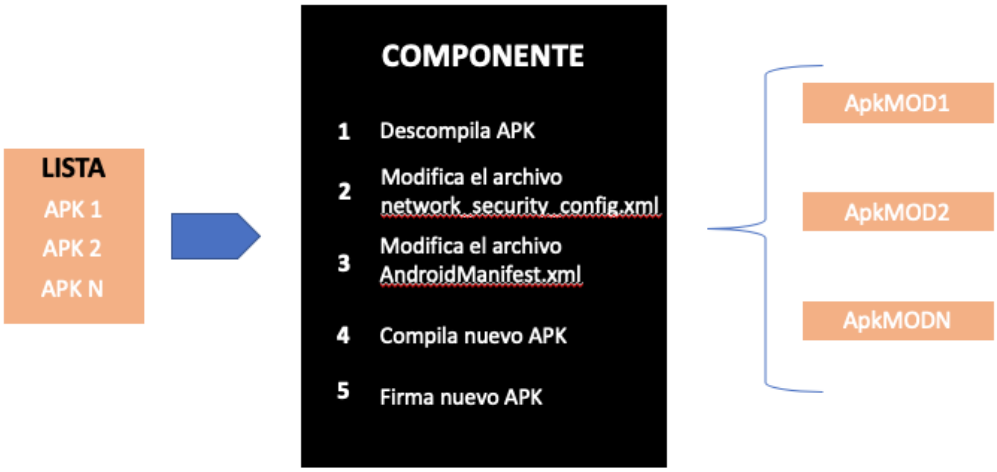

Fig. 8. Funciones del componente desarrollado

Gracias a este componente, el proceso para intentar inhabilitar el pinning por defecto se reduce a la generación de un certificado para realizar la firma, creación de una lista que contenga el nombre de los archivos a modificar y a la posterior instalación del apk modificado en la máquina Android, reduciendo notablemente el tiempo de trabajo. Más adelante, se ofrecerá una explicación más detallada del trabajo que realiza este componente, así como una guía con los comandos que se deben invocar para que funcione correctamente.

3.3. ESCENARIO II: PINNING CON BIBLIOTECAS

Anteriormente se menciona la posibilidad de modificar el archivo network_security_config.xml permitiendo todos los certificados. De esta forma no se impedirá ninguna conexión, siempre y cuando el pinning no esté implementado con una biblioteca. Una vez que el pinning está implementado a nivel de código, es necesario modificar el código para intentar permitir las conexiones a un certificado

diferente a los incluidos en la aplicación, permitiendo de esta forma monitorizar el contenido de las conexiones cifradas de una aplicación.

En un primer momento parece que no hay mayor solución que revisar el código aplicación por aplicación y modificar las funciones en las que se establece el pinning. Una tarea difícil, ya que cada aplicación cuenta con miles de líneas de código, posiblemente ofuscado y en cada una se puede realizar de manera diferente, como se expuso en el apartado previo. Sin embargo, teniendo en cuenta los patrones y el uso mayoritario de bibliotecas que existen, se puede automatizar el intentar detectar la forma utilizada en cada aplicación. Para ello haremos uso de la herramienta de instrumentación dinámica Frida. Esta herramienta, como ya se mencionó anteriormente, permite instrumentar y manipular el código de una aplicación según se está ejecutando.

Sabiendo que es posible modificar el código según se está ejecutando, aparecen diferentes soluciones para permitir que las aplicaciones acepten cualquier certificado, entra las que destacaremos dos:

1. Añadir nuestro certificado a la lista de certificados permitidos por la aplicación.
2. Modificar las funciones que realizan la comprobación de certificados.

Para empezar, se explicará cómo se modificarán las funciones comentadas en el apartado anterior:

- La primera forma de realizar certificate pinning explicada previamente es mediante el objeto Trust Manager. Lo que se realizará con Frida es crear un Trust Manager con los datos propios. Recordando lo explicado anteriormente, el objeto Trust Manager se utilizaba al crear el SSLContext, de tal forma que también hay que sobrescribir esa función, creando un contexto SSL que utilice el Trust Manager anterior y que muestre un mensaje por pantalla que indique que se ha interceptado un intento de realizar certificate pinning mediante Trust Manager.

- En el caso de OkHttp3, se comentó anteriormente que la función check que comprueba los certificados válidos es una función que tiene como argumento una lista de certificados. En el caso de que el certificado de la conexión que se está intentando establecer no coincida con ninguno de la lista, lanza una excepción. Lo que se conseguirá gracias a Frida es que en el momento en el que se vaya a comprobar el certificado, lo único que hará la función es imprimir un mensaje por pantalla sin llegar a lanzar la excepción que impide que la conexión que se está intentando establecer siga adelante.

- El tercer caso que analizamos previamente fue el de TrustKit. En este caso se establecían unas políticas en el archivo network_security_config.xml y se comprobaban una vez que se iniciaba TrustKit en el código. Lo que se hará con Frida es modificar la función que comprueba el nombre del Servidor y el certificado de tal forma que lo único que realice sea imprimir un mensaje por pantalla indicando que ha sido interceptado un intento de conexión https mediante TrustKit.

- El último caso que analizamos en el punto anterior fue el caso de Appcelerator. Como se comentó previamente, este caso es similar al de OkHttp3. Como consecuencia, la forma de inhabilitar el pinning será igual: reescribiremos el código de la función que comprueba los certificados de tal forma que nunca lance una excepción para interrumpir las conexiones, sino que lo que hará una vez detectada será mostrar un mensaje que indique la existencia y detección de esta función en el código.

Todos estos supuestos se pueden incluir en un mismo programa, de tal forma que al utilizar Frida con una aplicación compruebe todos estos diferentes casos. En el caso de que una aplicación emplee una forma diferente para el pinning, este script que hemos creado no será capaz de detectarla. Sin embargo, si mediante un análisis de código somos capaces de detectar la función que lleva a cabo la comprobación de certificados, podemos añadir esa función a nuestro script, aumentando así las diferentes maneras que seremos capaces de detectar y evitar.

Una vez que ya se conoce Frida y las posibilidades que nos proporciona, es el momento de explicar cómo instalar y utilizar Frida en la máquina Android desde el Proxy [24]. Para ello bastarán los siguientes pasos:

1. Se descarga y descomprime la última versión del servidor de Frida de Android en la máquina Proxy.

```
wget https://github.com/frida/frida/releases/frida-server-12.6.5-android-x86_64.xz
xz -d frida-server-12.6.5-android-x86_64.xz
```

2. Se instala el servidor de Frida en la máquina Android mediante adb. Es importante comentar que esta instalación ha de hacerse con privilegios de administrador y los comandos deben lanzarse en la máquina Proxy. Cabe mencionar que una vez conectados a la máquina Android y obtenidos los privilegios de administrador (root), es necesario volver a conectarse. Para que Frida disponga del certificado del proxy que se va a emplear (mitmproxy en este caso), también es necesario pasar el certificado a la máquina Android.

```
adb connect IPMaquinaAndroid
adb root
adb connect IPMaquinaAndroid
adb push frida-server-12.6.5-android-x86_64 /data/local/frida-server
adb shell "chmod 755 /data/local/frida-server"
adb push ~/.mitmproxy/mitmproxy-ca-cert.cer /data/local/tmp/cert-der.cr
```

3. Se deja el servidor a la escucha de nuevos eventos. Este comando se lanza también desde la máquina Proxy.

```
adb shell "/data/local/frida-server -listen 0.0.0.0&
```

Otro parámetro que admite Frida es la aplicación sobre la que se actúa. De esta forma, al lanzar Frida, se lanza automáticamente la aplicación en la máquina Android. Para ello hay que saber el id de aplicación, que suele ser de la forma *com.ejemplo.app*. Este id se obtiene si desde nuestra máquina Proxy se ejecuta un comando que imprime la lista de todas las aplicaciones instaladas junto al comando grep, que permite buscar una palabra dentro de esa lista. Por tanto, el comando a lanzar es el siguiente:

```
adb shell pm list packages | grep nombreAPP
```

(sustituyendo *nombreAPP* por el nombre de la aplicación cuyo id queremos conocer).

Si se dispone del apk, otra forma de obtener el nombre es con el comando:

```
aapt dump badging nombreApp | grep "package: name"
```

(sustituyendo *nombreAPP* por el nombre de la aplicación cuyo id queremos conocer).

Una vez que el servidor de Frida está instalado en la máquina Android y a la escucha, el script con que se utilizará para modificar el código de las aplicaciones está preparado y el id de la aplicación que se va a auditar es conocido, el último paso a realizar es lanzar Frida desde la máquina Proxy. El comando queda de la siguiente manera:

```
frida -U -f com.ejemplo.app -l programa.js --no-pause
```

Tras lanzar este comando en el Proxy, en Android debería aparecer la aplicación especificada como si se hubiera abierto de forma manual.

El script utilizado para cargar en Frida es una modificación del script en el GitHub de Jamie Holding. La explicación del funcionamiento de dicho script y el enlace para descargarlo se puede encontrar en su blog [25].

A este script se le ha añadido la comprobación de la función obsoleta *check* de la biblioteca OkHttp3, ya que pese a estar obsoleta, algunas aplicaciones siguen implementándola.

Para automatizar este método, se ha modificado un componente previo desarrollado en el Departamento de Ingeniería de Sistemas Telemáticos[26]. Este componente se utiliza para instalar una aplicación, abrirla y navegar por ella mientras se recogen todas las conexiones que realiza dicha aplicación para analizarlas posteriormente. Para ello, necesita recibir como parámetros la dirección IP de la máquina Android y la aplicación que se desea instalar y lanzar.

Tras las modificaciones, el componente recibe los parámetros anteriores y el script con el que se trabaja (se proporciona el script ya mencionado anteriormente). El componente se conectará como *root* a la máquina Android, pondrá el servidor a la escucha y lanzará tanto la aplicación como el script que se pasan como parámetros. Además, también configura la red (siguiendo los pasos de la documentación de mitmproxy mencionados anteriormente) para que el tráfico vaya a través de mitmproxy. Una vez realizada la configuración de la red y la instalación, el componente pregunta al usuario si desea que se lance Frida y, cuando recibe la confirmación, lanza la aplicación en el terminal Android.

Para que funcione este nuevo componente, previamente hay que instalar de forma manual el servidor de Frida y el certificado en la máquina virtual Android (es decir, los pasos 1 y 2 mencionados anteriormente). Una vez se lanza este componente, se abre la aplicación en la máquina Android y se guardan todas las conexiones que realiza en un archivo de salida llamado output.log, con el objetivo una vez más de ser analizadas posteriormente. Además, una vez se ejecuta este componente, gracias al script utilizado por Frida se puede visualizar por pantalla si está siendo utilizada alguna de las bibliotecas que se contemplan en el script y, en algunos casos, a qué dominio se está intentando conectar la aplicación con cada biblioteca. Esta información se puede almacenar también en un archivo de texto y puede ser muy útil de cara a un análisis estático de código.

Para analizar las conexiones recogidas, se dispone de otro componente, también desarrollado en el Departamento de Ingeniería de Sistemas Telemáticos y modificado para adaptarlo a las necesidades de este trabajo. Este programa recibe como parámetro de entrada la salida generada por el componente anterior, que es el archivo donde se recogen todas las conexiones establecidas. El componente de análisis permite imprimir información de cada conexión guardada. En el caso de que se haya conseguido ver en claro la conexión, se imprime la dirección IP, la petición (get, post, etc) y el contenido. En caso contrario, si no se consigue evitar el certificate pinning de una conexión, se imprime un mensaje que indica que esa conexión se realizó con certificate pinning y el dominio o dirección IP a la que se intentaba conectar la aplicación, no siendo posible ver el contenido de dicha conexión debido al cifrado.

3.3.1. ESCENARIO III: PINNING DESCONOCIDO

Hay que tener en cuenta que existen innumerables formas de realizar certificate pinning y por cada una de ellas se puede intentar encontrar una manera de evitarlo. Hay casos en los que la solución pasa por realizar un análisis exhaustivo del código, de los procesos y de las rutinas que invoca la aplicación según está siendo utilizada.

Existen herramientas muy potentes de ingeniería inversa que nos permiten analizar las aplicaciones a nivel de ensamblador, como es el caso de IDA (Interactive Disassembler).

Con IDA se pueden estudiar y modificar las subrutinas de las aplicaciones. De esta forma, si se localiza la excepción que lanza una aplicación que recibe un certificado inválido y se cambia la rutina que debe seguir, es posible evitar el pinning. Dicha rutina suele ser un salto condicional al que se llega cuando el certificado utilizado no es igual al esperado (JNZ en ensamblador).

Para desactivar dicha rutina puede ser suficiente con cambiar la condición mediante la cual se llega a esa excepción. Cambiando la orden de JNZ a JZ (es decir, de saltar si el certificado NO es igual a saltar si el certificado es igual), se consigue que solo salte la excepción en el caso de que el certificado sea igual al que la aplicación espera y, por tanto, solo en ese caso se interrumpirían las conexiones. Como se está utilizando un certificado desconocido para la aplicación, esa subrutina no se debe llegar a lanzar y, por tanto, la aplicación debe permitir las conexiones. [27]

En este método no se va a profundizar más ya que las posibilidades existentes son infinitas y se requiere de mucho tiempo para analizar cada aplicación que se desee auditar, además de unos buenos conocimientos de programación en ensamblador.

En el siguiente cuadro se muestra un resumen de las diferentes formas de implementar conexiones seguras desde Android y el método que se seguirá para poder realizar las auditorías:

Tipo de pinning		Metodología
Por defecto		Modificar XML
Mediante librerías	Trust Manager	Instrumentación dinámica
	OkHttp3	
	TrustKit	
	Appcelerator	

Fig. 9. Formas de realizar certificate pinning

3.4. MANUAL DE COMPONENTES DESARROLLADOS

Para llevar a cabo las tareas anteriores de forma semiautomática se han desarrollado dos componentes que se pueden encontrar en mi GitHub (https://github.com/framirccv/TFG/). La idea principal es que se puedan realizar estas tareas para un gran número de aplicaciones con el mínimo esfuerzo posible y sin tener que diferenciar el escenario en el que se encuentra cada aplicación. Para ello, es necesario emplear los dos componentes desarrollados ya que el primero modificará el código de una lista de archivos apk y con el segundo se ejecutará la aplicación, aplicando la instrumentación dinámica.
A continuación, se va a explicar cada componente por separado para, posteriormente, mostrar cómo sería el flujo de trabajo mezclando los dos componentes.

3.4.1. MODIFICACIÓN DE CÓDIGO DE FORMA MASIVA

El primer componente se llama *listaApk.py* y consiste en un programa desarrollado en Python que recibe como parámetro un archivo. Este archivo de texto contiene el nombre de una aplicación por cada línea, además del keystore y el alias de dicho keystore. Cada parámetro debe ir separado por punto y coma y en el archivo de texto no debe estar indicada la extensión (todo lo que viene a partir del punto en el nombre). Tanto el archivo apk como el keystore deben estar en el mismo directorio en el que se ejecuta el script, al igual que el archivo network_security_config.xml que se proporciona con el programa. En nuestro ejemplo, la carpeta de la que disponemos es la siguiente:

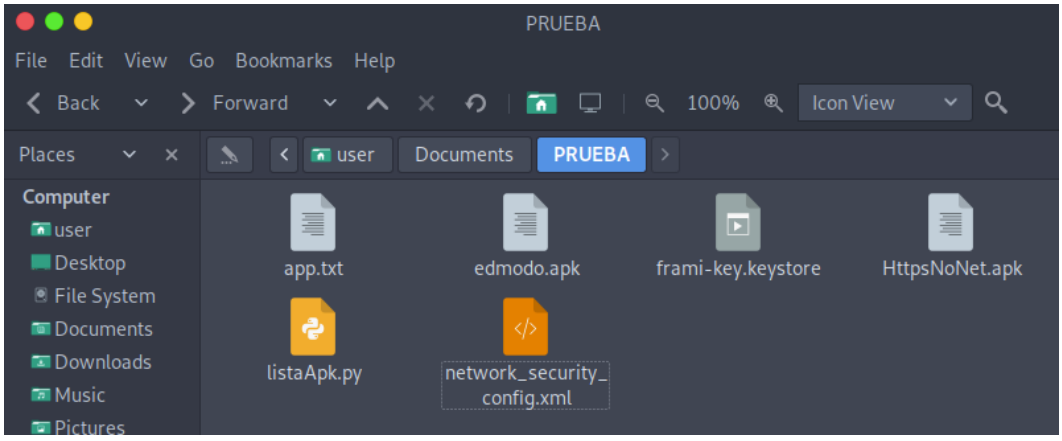

Fig. 10. Elementos para modificación masiva

Disponemos de dos archivos apk como se puede observar: HttpsNoNet.apk (aplicación desarrollada para la realización de este trabajo) y edmodo.apk. (aplicación de terceros). Para utilizar el script *listaApk.py* con estas dos aplicaciones, el contenido del archivo *app.txt* debe ser el siguiente:

HttpsNoNet;frami-key;frami

edmodo;frami-key;frami

Una vez que el archivo de texto está preparado, para ejecutar el componente el comando es el siguiente:

```
python listApk.py app.txt
```

Se debe cambiar el nombre "app.txt" por el nombre del archivo de texto que se haya utilizado.

Una vez lanzado, el componente recorre cada línea del archivo de texto que recibió como parámetro, separando los parámetros por punto y coma, guardando cada parámetro como una variable. Después, descompila el archivo apk en una carpeta de salida del mismo nombre que la aplicación y que se encontrará en la misma ruta donde se haya ejecutado. Una vez descompilado, se copia el archivo network_security_config.xml preconfigurado en el directorio /res/xml y posteriormente se recorre el archivo AndroidManifest.xml y se añade como último parámetro del atributo <application> la línea:

```
android:networkSecurityConfig="@xml/network_security_config"> '
```

De esta forma, se termina con las modificaciones del código y el componente pasa a volver a compilar la aplicación y firmarla con el keyStore que se ha indicado como parámetro.

Una vez terminado este proceso, los nuevos archivos que se pueden encontrar son:

- Una carpeta por cada aplicación que contiene el código descompilado y ya modificado. En el caso del ejemplo tendremos una carpeta HttpsNoNet y otra edmodo.
- Un nuevo archivo apk que se llama "archivoAnteriorMod.apk". En nuestro caso (tras haber hecho limpieza), nos encontramos con los nuevos archivos HttpsNoNetMod.apk y edmodoMod.apk

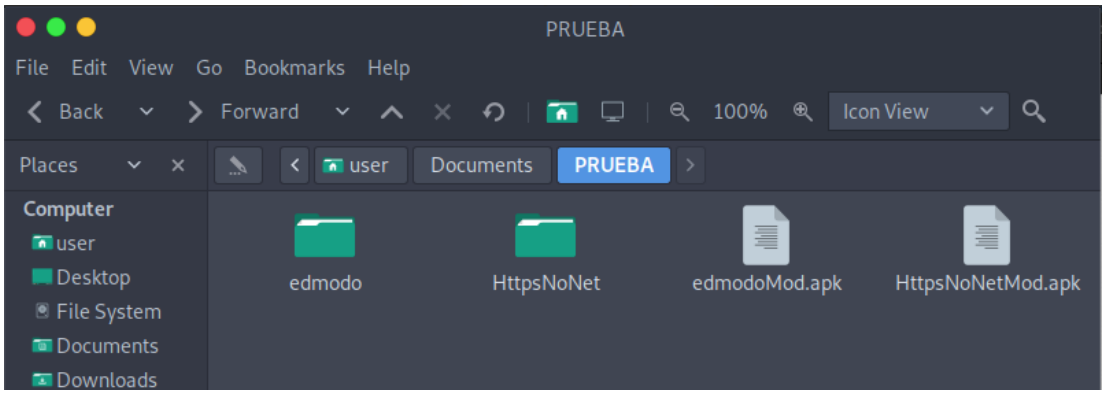

Fig. 11. Elementos resultantes de la modificación masiva

Por tanto, la interacción del usuario con este componente quedaría de la siguiente forma:

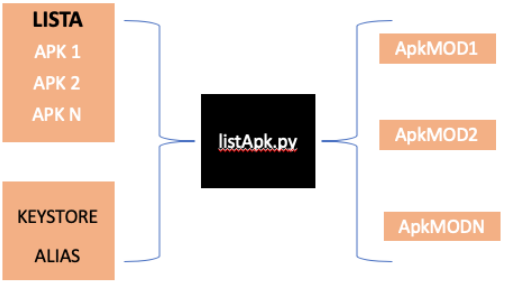

Fig. 12. Interacción del usuario con listApk.py

3.4.2. INSTRUMENTACIÓN DINÁMICA

Una vez se han obtenido los archivos apk modificados, se puede emplear el segundo componente desarrollado, de tal forma se realice la ejecución de la aplicación permitiendo la instrumentación dinámica. Este segundo componente consta de dos módulos diferentes:

- Interceptación de peticiones (*Intercept*):
  - proxy.sh: script que el usuario tiene que lanzar. Genera un archivo de salida llamado output.log, que se analizará en el módulo de análisis.
  - inspect_request.py: script que configura mitmproxy para guardar las conexiones de las aplicaciones. El usuario no tiene que interaccionar con este componente, ya que se lanza desde proxy.
  - Config.yaml: archivo de configuración que debe copiarse en el directorio: /home/${user}/.mitmproxy antes de la primera ejecución del programa.
  - Pinning.js: script proporcionado para la instrumentación dinámica.
  - README.md: archivo de texto donde se comenta el funcionamiento del programa.
- Análisis de peticiones (*Analyze*):
  - Analyze.py: script que el usuario utilizará. Cargará como parámetro el archivo Output.log generado anteriormente.
  - Data.py: archivo en el que se tratan los diferentes datos del archivo Output.log
  - Utils.py: permite que el programa entienda el formato del archivo Output.log

Para interceptar peticiones, el archivo que el usuario debe conocer es el proxy.sh. Este script admite como parámetros la dirección ip de la máquina Android a la que debe conectarse (-i), el archivo apk que se va a estudiar (-a) y el script con el que se va a instrumentar la aplicación una vez que se lance (-s), quedando el comando de ejecución de la siguiente manera:

```
./proxy.sh -i IPMaquinaAndroid -a ArchivoAPK -s ScriptInstrumentación
```

Proxy.sh lo primero que hace es configurar la red para que el tráfico pase a través de la máquina en la que se lanza (máquina Proxy en el entorno de trabajo). Posteriormente, se conecta a la máquina Android como root, instala la aplicación indicada y abre un nuevo terminal en el que ejecuta el comando visto anteriormente para dejar a la escucha el servidor de Frida situado en la máquina Android tal y como se explicó en el punto anterior. También se crea otro terminal en el que se abrirá la consola de mitmproxy, permitiendo así monitorizar las conexiones en tiempo real.

Una vez instalada la aplicación, se pregunta al usuario si desea lanzar Frida. Cuando el usuario acepta, se lanza la aplicación instalada en la máquina Android, instrumentándola con el script que indicó como parámetro. En este instante, empieza la ejecución de la aplicación en la máquina Android y en la máquina Proxy hay tres consolas abiertas en las que se muestra la siguiente información:

- La primera consola abierta es en la que se lanzó el script proxy.sh. En esta ventana se podrán ver las bibliotecas interceptadas.
- La segunda consola es la ventana de mitmproxy, en la que se observarán las conexiones que está realizando la aplicación
- Por último, existirá una tercera ventana en la que no se podrá escribir. En esta ventana es en la que se ejecutó el comando que puso el servidor a la escucha.

Para salir de la ejecución de Frida, basta con escribir “quit” y pulsar intro en la ventana de terminal en la que se está ejecutando. Para salir de las otras dos ventanas, basta con pulsar las teclas Control+C.

Una vez interrumpida la ejecución del script (bien cerrando la aplicación instrumentada en la máquina Android o bien escribiendo “quit” en el terminal), se crea un archivo de salida llamado Output.log

Dicho archivo será el que se pase como parámetro al módulo de análisis. Una vez se dispone del archivo Output.log y el script analyze.py en el mismo directorio, el comando a ejecutar es:

```
python3 analyze.py Output.log fasen > resultado.txt
```

Como se mencionó anteriormente, este componente es una modificación de un componente que se desarrolló para otro proyecto del Departamento de Ingeniería de Sistemas Telemáticos. Por este

motivo, el parámetro fasen es necesario para el funcionamiento, si bien no es relevante para el trabajo aquí expuesto. La salida se redirige a un archivo de texto que contiene los datos del análisis de tal forma que el usuario sea capaz de entenderlo. Un ejemplo de los resultados obtenidos es la siguiente captura:

```
(True, 'api.edmodo.com', Request(GET 34.217.245.35:443/social_signals))
EL REQ [0] ES False
(False, 'app-measurement.com', ('172.217.16.238', 443))
EL REQ [0] ES True
EL REQ [0] ES True
{"properties":{"stream_index_seen":5},"product_group":"stream","visual_element":
"stream","tracking_action":"display","context":"android_home"}
(True, 'api.edmodo.com', Request(POST 52.41.221.213:443/action_trackings))
EL REQ [0] ES False
(False, 'eu.ime.cootek.com', ('3.120.101.104', 443))
EL REQ [0] ES FalseEL REQ [0] ES False
(False, 'app-measurement.com', ('172.217.16.238', 443))

(False, 'www.directly.com', ('100.25.75.186', 443))
```

Fig. 13. Archivo resultante del análisis de conexiones

Se interpreta de la siguiente una forma muy intuitiva ya que los resultados *True* son aquellos en los que sí se puede ver el contenido, mientras que en los resultados *False* no se ha sido capaz de descifrar la comunicación. En ambos casos aparecen tanto el dominio, como la IP como el puerto. Además, en el caso de que el resultado sea *True*, se puede ver la información de la petición.

En resumen, la interacción del usuario con el componente que intercepta y analiza las peticiones se recoge en el siguiente esquema:

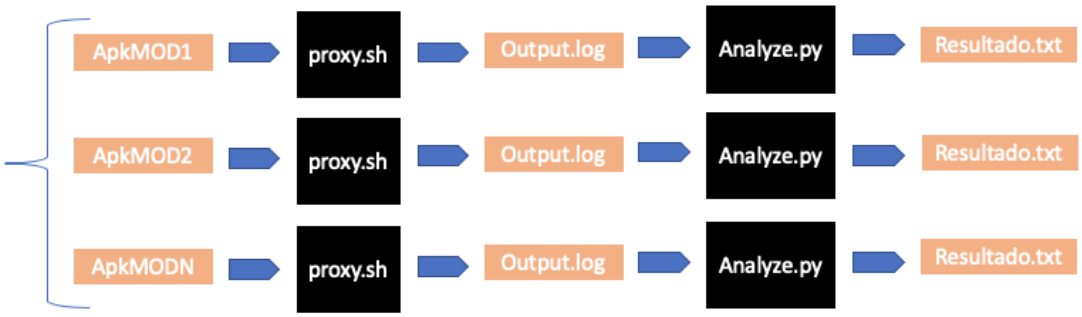

Fig. 14. Interacción del usuario para interceptar y analizar

3.4.3. FLUJO DE TRABAJO

Para ayudar a clarificar el funcionamiento de los diferentes componentes, se va a exponer cómo sería el flujo de trabajo con una aplicación empleando los componentes desarrollados. De esta forma se pretende recalcar qué elementos realiza de forma automática el software, qué requisitos se necesitan para que este funcione y demostrar cómo disminuye la cantidad de trabajo realizado por cada aplicación. Para esta demostración se seguirá el ejemplo con la aplicación edmodo anteriormente mencionada.

En primer lugar, se necesita levantar el entorno de trabajo anteriormente expuesto en el apartado 3.1. Esto incluye también la instalación de Frida en la máquina Android, para ello hay que seguir los puntos 1 y 2 del apartado 3.3. Una vez realizado esto, es necesario crear un archivo KeyStore que contenga el certificado que se empleará para firmar las aplicaciones modificadas. El comando para

ello está mencionado también en apartados anteriores. También es necesario copiar el archivo Config.yaml a la ruta indicada previamente.

Estos requisitos perduran en el tiempo, por lo que tan solo habrá que realizarlos la primera vez que se utilice el entorno. Una vez se dispone el entorno y de los componentes desarrollados, el trabajo para la aplicación edmodo (suponiendo el archivo app.txt que se mostró anteriormente) se realizaría de la siguiente forma:

```
python listApk.py app.txt
./proxy.sh -i IPMáquinaAndroid -a edmodoMod.apk -s script
python3  analyze.py Output.log fasen > resultado.txt
```

Como se puede observar, la cantidad de comandos a introducir se ha reducido considerablemente. Además, la modificación del código de las aplicaciones se puede realizar de forma masiva, reduciendo toda la cadena de comandos que se realizaba anteriormente por cada aplicación a la realización de una lista de aplicaciones a modificar.

En definitiva, gracias a estos componentes se ahorra mucho tiempo y, además, se evitan también errores humanos, como por ejemplo teclear mal una línea en el archivo network_security_config.xml. Por otra parte, es importante destacar que, gracias a los componentes desarrollados, cualquier usuario debería ser capaz de obtener unos resultados de una aplicación siguiendo solamente este pequeño manual, sin la necesidad de conocer qué procesos se están realizando por detrás ni poseer unos extensos conocimientos en comunicaciones seguras o programación.

4. VALIDACIÓN Y RESULTADOS

En los puntos anteriores se han expuesto diversas maneras de realizar certificate pinning en una aplicación Android, se ha propuesto un método para saltárselo en una aplicación cualquiera, y se ha descrito un componente que permite automatizar este método.

En este apartado se van a exponer y analizar los resultados obtenidos tras aplicar los métodos y componentes desarrollados a un conjunto de aplicaciones.

4.1. PINNING POR DEFECTO

En un primer lugar, la forma más sencilla es probar a saltar el pinning que utiliza Android por defecto. Para ello se ha desarrollado una simple aplicación para una versión de Android superior a la 6.0. Dicha aplicación tan solo tiene un cometido: realizar una petición https a una página web nada más abrir el programa. La conexión https se ha realizado con la biblioteca OkHttp sin fijar ningún certificado, simplemente definiendo la conexión. Para ser capaces de conocer si la petición Https se realiza correctamente, se ha incluido un cuadro de texto en la pantalla que muestra el código resultante de la petición (si todo va bien debe ser un código 200).

```
package com.example.frami.http3;

import ...

public class MainActivity extends AppCompatActivity {

    private TextView mTextViewResult;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mTextViewResult = findViewById(R.id.text_view_result);

        OkHttpClient client = new OkHttpClient();
        String url = "https://                    ";

        Request request = new Request.Builder()
                .url(url)
                .build();

        client.newCall(request).enqueue(new Callback() {
            @Override
            public void onFailure(Call call, IOException e) { e.printStackTrace(); }

            @Override
            public void onResponse(Call call, Response response) throws IOException {
                if (response.isSuccessful()) {
                    final String myResponse = response.networkResponse().toString();

                    MainActivity.this.runOnUiThread(() -> {
                            mTextViewResult.setText(myResponse);
                        });
                }
            }
        });
    }
}
```

Fig. 15. Código de ejemplo

Una vez comprobado el funcionamiento de la aplicación en el simulador de Android Studio, se genera el apk y se instala en la máquina virtual Android (la instalación se puede realizar mediante adb). En dicha máquina virtual se puede comprobar que, cuando se conecta a Internet directamente, la aplicación realiza la petición correctamente. Sin embargo, una vez que activamos el escenario descrito anteriormente y Android se conecta a Internet a través del Proxy, la aplicación no muestra ningún mensaje en la pantalla ya que la petición no llega a realizarse puesto que la máquina virtual Android se está encontrando con un certificado (mitmproxy) que no está en la lista de certificados que se incluye por defecto en el dispositivo.

Suponiendo que el certificado de mitmproxy ya está instalado en Android (explicado con anterioridad al definir el escenario de trabajo), el trabajo que queda hacer es obtener el código de la aplicación, modificarla, firmarla y volverla a instalar. Todo este trabajo se realizará desde la máquina Proxy. Para comenzar, podemos o bien obtener el apk de la aplicación desde la Android mediante adb como ya se comentó anteriormente o bien como es un apk propio trabajar sobre el mismo fichero que se generó con AndroidStudio y se instaló con adb en la máquina.

En un primer momento, se siguen de forma manual los pasos anteriormente mencionados: se descompila la aplicación, se añade el archivo network_security_config.xml y se refleja su existencia en el manifest, se vuelve a compilar y firmar la aplicación y se instala en el dispositivo de pruebas. Una vez instalada, antes de abrir la aplicación es necesario tener abierto mitmproxy en la máquina Proxy para ser capaces de ver la conexión. Ahora, una vez se abre la aplicación, se observa tanto en la pantalla de la máquina virtual Android como en el Proxy la conexión realizada y su código 200 que indica que se ha realizado correctamente.

Posteriormente, se ha comprobado que el componente desarrollado automatiza este proceso obteniendo los mismos resultados. Partiendo otra vez del apk generado con Android Studio, se crea un archivo de texto como el definido en la explicación del componente, quedando de la siguiente forma:

ArchivoAPK;KeyStore;AliasKeyStore

Posteriormente, se lanza el programa creado pasándole dicho archivo como parámetro:

```
python listApk.py app.txt
```

El resultado de este programa es un nuevo apk que contiene las modificaciones necesarias para confiar en los certificados de usuario. Esta nueva aplicación se instala en la máquina virtual Android. Abriendo primero mitmproxy en la máquina Proxy y la aplicación modificada en la Android, se observa que la aplicación vuelve a realizar correctamente la petición, demostrando así que el componente funciona.

Gracias a este proceso, se consigue que una aplicación que de primeras es incapaz de admitir un certificado fuera de los indicados en el sistema operativo, una vez modificada sea capaz de realizar conexiones con cualquier certificado instalado en el dispositivo Android desde el que se utiliza.

Análisis de Tráfico	Apk sin modificar	Apk tras realizar modificación manual	Apk tras realizar modificación a través de componente desarrollado
App propia	✘	✓	✓

Fig. 16. Resultados de la aplicación del método para pinning nativo

4.2. PINNING MEDIANTE BIBLIOTECAS

En esta ocasión es algo más complicado realizar una aplicación y sortear el certificate pinning posteriormente. Esto se debe a que, si se utiliza la lógica por defecto de las bibliotecas más comunes, la forma teórica de evitar la comprobación de certificados es la expuesta en puntos anteriores. En el caso de realizar una aplicación propia en la que se realice el pinning con otras funciones diferentes, también sería relativamente fácil ya que se conoce la lógica interna de la aplicación. Es por ello que para poner a prueba el componente desarrollado se han empleado diferentes aplicaciones de terceros disponibles en el mercado de aplicaciones de Android.

La primera aplicación puesta a prueba fue la aplicación del periódico español As. Tras instalar dicha aplicación del sitio oficial de Google (Play Store), se conecta la máquina Android al Proxy y se abre la aplicación, intentando navegar a través de ella. Al no reconocer el certificado de mitmproxy, la aplicación no muestra contenido y se queda cargando la página de inicio.

Posteriormente, se cierra la aplicación y se vuelve a lanzar desde la máquina Proxy a través de Frida como se ha expuesto en puntos anteriores. Una vez lanzada a través de Frida, la aplicación carga el contenido perfectamente, pudiéndose observar en la consola de mitmproxy las conexiones realizadas y en la pantalla mostrada por Frida las bibliotecas a través de las cuales está implementado el certificate pinning.

Después de comprobar el correcto comportamiento del componente con esta primera aplicación, se pasa a validar con un conjunto de aplicaciones mayor que ya han sido analizadas en el grupo de

investigación, de cara a comparar los resultados obtenidos con el componente aquí desarrollado con los obtenidos en dicho proyecto.

Las aplicaciones están divididas en dos ámbitos y son las siguientes:

ÁMBITO EDUCATIVO	ÁMBITO DEPORTE Y TIEMPO LIBRE
Agenda Escolar	Entrenamiento en 7 minutos
ClassDojo	Google Fit
Duolingo	Mi Fit
Edmodo	Runtastic
Flipgrid	Runtastic PRO
Kahoot	Samsung Health
Moodle	Step Tracker
Remind: Comunicación Segura	Strava
Socrative	SyncMyTracks
TokApp School	Tools & Mi Band

Fig. 17. Aplicaciones analizadas

Sin embargo, en este trabajo no se van a probar todas ellas ya que algunas requieren un terminal físico para ejecutarse y, por tanto, no se pueden estudiar en un entorno virtual como es el caso de este trabajo. Además, también se omitirán las aplicaciones en las que no se detectaron conexiones que emplearan certificate pinning, ya que no se podría observar ninguna mejora.

A continuación, se incluye una tabla en la que para cada aplicación analizada se comparan los resultados previos y los obtenidos al usar el componente aquí desarrollado, mostrando también qué bibliotecas se han detectado.

Aplicación	Conexiones protegidas (resultados previos)	Conexiones protegidas (resultados actuales)	Librerías detectadas
Class-Dojo	pstatic.classdojo.com	✓	OkHttp3
Edmodo	172.217.16.234	25%	TrustManager
	172.217.16.238	✗	?
	172.217.168.164	✗	?
	cdnjs.cloudflare.com	50%	TrustManager
	fonts.gstatic.com	66%	TrustManager
	p14.zdassets.com	✗	TrustManager
	static.zdassets.com	✓	TrustManager
Flipgrid	ruby.flipgrid.com	✓	OkHttp3
	secure.flipgrid.com	✗	?
Kahoot	create.kahoot.it	✓	TrustManager
	ssl.gstatic.om	✓	TrustManager
Socrative	b.socrative.com	✓	TrustManager
Step Tracker	googleads.g.doubleclick.net	44%	?

Fig. 18. Resultados de aplicar instrumentación dinámica

El significado de la tabla anterior es el siguiente:

- ✓ Las comunicaciones se descifraron correctamente.
- ✗ Las comunicaciones no se consiguieron descifrar.
- **%** Porcentaje de conexiones con la dirección correspondiente descifradas.

Analizando los resultados, se puede observar que de las 14 conexiones en las que no se podía auditar el contenido debido al uso de mecanismos de protección, 10 se han conseguido descifrar total (6) o parcialmente (4).

De las 4 conexiones que no se consiguen procesar, en 3 ocasiones no se obtiene información sobre la librería que se está empleando para realizar dicha conexión, por lo que es posible que se esté usando un método desconocido para realizar la comprobación de certificado y, por tanto, habría que analizar dichas conexiones de otra manera.

En la conexión restante que no se consigue descifrar, pero sí se sabe que usa la biblioteca *TrustManager*, es posible que utilice otro mecanismo extra de verificación, por lo que se impide saltar el certificate pinning con el script que se utilizó a la hora de realizar estas pruebas. Como se comentó en puntos anteriores, para evitar esta nueva forma que presenta la aplicación de proteger sus conexiones, convendría realizar un análisis estático de la aplicación.

En el siguiente gráfico se puede encontrar un resumen del análisis realizado:

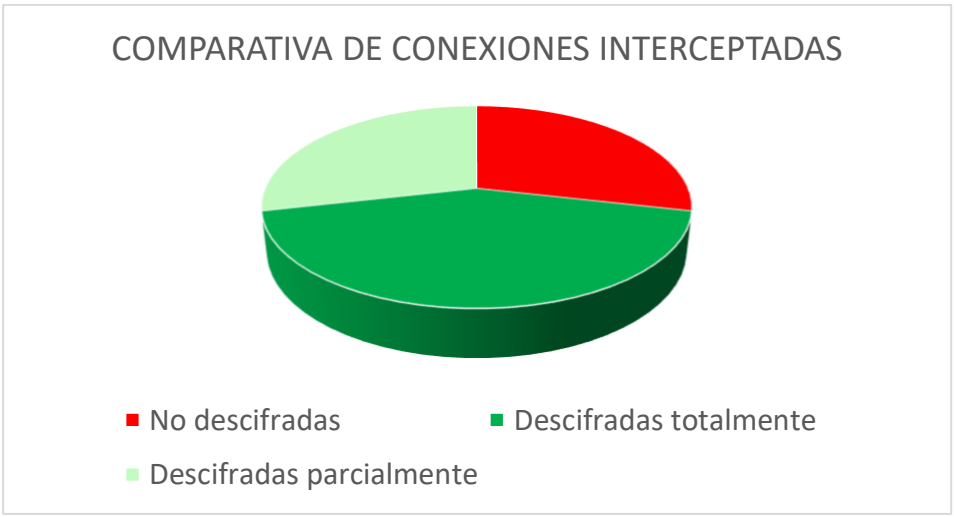

Fig. 19. Comparativa de las conexiones interceptadas

Cabe mencionar que no solo se han detectado las conexiones recogidas en la tabla. Se detectan multitud de conexiones, tanto cifradas como en claro, no reflejadas en este resumen, ya que el objetivo de esta comparativa es indicar la mejora que supone el desarrollo de este nuevo componente. Entre las conexiones detectadas cuya información se puede ver en claro destacan llamadas de las aplicaciones a sus APIs para transmitir diferente información (inicios de sesión, envío de mensajes) como peticiones para descargar hojas de estilos (css).

5. CONCLUSIONES Y LÍNEAS FUTURAS

5.1. CONCLUSIONES

A lo largo de todo este documento se han expuesto diferentes formas de poder auditar el contenido de aplicaciones de terceros, incluso cuando están cifradas. Estos conocimientos se pueden emplear académicamente o para realizar auditorías solicitadas por las autoridades de supervisión, cuerpos de seguridad, desarrolladores o dueños de la aplicación, e incluso los usuarios finales.

Sin embargo, la realización del trabajo también ha permitido conocer las vulnerabilidades más habituales de las aplicaciones en cuanto a establecimiento de conexiones seguras. Debido a esto, es necesario desarrollar cualquier aplicación de la forma más segura posible, protegiendo al máximo los datos tratados en ellas. Para ello, a modo de conclusión se realizarán unas recomendaciones de buenas prácticas.

En primer lugar, a la hora de protegerse de la ingeniería inversa hay que ser especialmente cuidadoso. Un atacante que dedica tiempo y recursos a una aplicación puede llegar a conocer más de ella que los miembros de un equipo de desarrollo que se repartieron el trabajo a la hora de escribir el código. Aunque no existe una manera de imposibilitar la ingeniería inversa por completo, la misión del equipo de desarrollo debe ser dificultarla al máximo posible. Para ello existen diferentes formas que pueden complicar mucho el trabajo de un atacante o de un auditor. La solución que ofrece Android se llama ProGuard, herramienta explicada en la documentación de Android. En dicha documentación se definen las funcionalidades herramienta ProGuard (eliminar el código que no se utiliza, ocultar las clases, archivos, métodos) También se resalta la importancia de personalizar correctamente la configuración de ProGuard para que no se eliminen por error líneas de código que sí son necesarias, por lo que hay que tener cuidado al utilizar esta herramienta y realizar pruebas suficientes con la aplicación final. [28]

También hay que evitar almacenar en claro credenciales de autenticación de usuarios, de acceso a una API, variables. Estos datos deben estar siempre protegidos y no ser entendibles a simple vista, por lo que es conveniente encriptarlos de cara a dificultar el entendimiento de la lógica de la aplicación y, sobre todo, dificultar que se pueda variar el comportamiento mediante nuevas inyecciones de código.

Además, siempre hay que tener presente las librerías empleadas pueden presentar vulnerabilidades por lo que hay que tener el código actualizado y debe revisarse frecuentemente para evitar que un atacante se aproveche de un fallo conocido.

Por último, siempre hay que prever que el comportamiento del usuario puede salirse de los márgenes establecidos. Es por ello por lo que hay que limitar el control que el usuario puede ejercer sobre una aplicación. Si se permite al usuario introducir cualquier tipo de dato, es posible que intente ejecutar código o enviar información que altere el funcionamiento de nuestro sistema.

Estas directrices se deben seguir a lo largo del desarrollo de toda la aplicación para conseguir un código seguro, ya que los problemas asociados pueden ir desde un fallo simple que haga que se cierre la aplicación hasta alterar el comportamiento del sistema haciendo que quede información sensible al descubierto y a disposición de un atacante.

Desde el punto de vista del usuario, es importante destacar la importancia que tiene conocer siempre el origen de las aplicaciones descargadas y tener cuidado con aquellas descargadas de sitios no oficiales (fuera de la PlayStore en el caso de Android) ya que, como se ha visto en puntos anteriores, cualquier persona puede modificar el código de una aplicación con fines maliciosos y subirla a internet con el objetivo de que miles de personas la ejecuten en sus terminales.

5.2. LIMITACIONES

El desarrollo de este software se ha realizado con el objetivo de ofrecer un componente que facilite el trabajo auditar una aplicación Android. También se puede utilizar con fines académicos para demostrar las vulnerabilidades y funcionamiento tanto de los protocolos http como https o de las

diferentes librerías de código empleadas para establecer conexiones en una aplicación Android y que se han comentado anteriormente.

Por otra parte, en términos de uso, este software está limitado simplemente al método más básico de protección de las comunicaciones (pinning nativo) y a las librerías más habituales dentro del mundo de Android. Actualmente cubre gran cantidad de conexiones tal y como se vio en el análisis, pero a medida que se vaya avanzando y se desarrollen nuevas librerías (o incluso, nuevos protocolos) para establecer las conexiones https en las aplicaciones, será necesario actualizar el contenido de este software de acuerdo con las nuevas prácticas que se pongan en uso en un futuro.

5.3. LÍNEAS FUTURAS

Al igual que se ha mencionado la existencia de infinidad de métodos para fijar certificados en Android, también se ha mencionado que el script empleado en el método de instrumentación dinámica se puede hacer tan extenso como el usuario necesite. Por esta razón, a medida que avance el análisis estático de diferentes aplicaciones o se vayan popularizando nuevas formas de realizar certificate pinning, se puede ir ampliando y mejorando el programa que se carga en Frida. De esta forma, se aumentará la efectividad del componente desarrollado, mejorando así sus prestaciones y aumentando el número de conexiones descifradas con éxito.

Actualmente existe una opción para navegar de forma automática por aplicaciones Android conocida como *monkey*. Sin embargo, esta forma de automatizar la interacción humana con las aplicaciones es algo ineficaz, ya que se atasca en pantallas en las que el usuario tiene que introducir datos (como, por ejemplo, inicios de sesión). Si se desarrollara un componente que permita la navegación por la aplicación de forma automática, se podrían integrar los dos componentes desarrollados para reducir aun más la interacción humana con el componente, ahorrando de esta forma tiempo de trabajo. De esta forma, el componente modificaría el código de una aplicación, la instalaría directamente en el dispositivo Android, navegaría por ella recogiendo sus conexiones y posteriormente cerraría la aplicación, pasando a la siguiente de la lista.

BIBLIOGRAFÍA

[1] KUROSE, Jim y KEITH, Ross. *Computer Networking. A top down approach*. 6ª edición. [en línea]. [Consulta 04/03/2019] http://ce.sharif.edu/courses/94-95/2/ce443-3/resources/root/Book/fqo47.Computer.Networking.A.TopDown.Approach.6th.Edition.pdf

[2] THOMAS, Stephen. *SSL & TLS Essentials.* [en línea]. [Consulta: 20/02/2019] http://index-of.co.uk/Hacking-Coleccion/SSL%20&%20TLS%20Essentials%20-%20Securing%20the%20Web.pdf

[3] CERTSUPERIOR. *Los 5 vectores de Ataque más comunes de la tecnología SSL.* [en línea]. [Consulta: 1/03/2019] https://www.certsuperior.com/Blog/ataques-tecnologia-ssl

[4] STALLINGS, William. *Cryptography and network security*. 7ª Edición. Pearson. ISBN 9789332585225

[5] SYMANTECH. *Certificate Pinning.* [en línea]. [Consulta: 02/02/2019] https://www.symantec.com/content/dam/symantec/docs/white-papers/certificate-pinning-en.pdf

[6] PRABHARKARAN, Prabhin. *Certificate Revocation – CRL VS OCSP.* [en línea] [Consulta: 4/03/2019] https://www.techrunnr.com/certificate-revocation-crl-vs-ocsp/

[7] WASS, Cody. *Four Ways to Bypass Android SSL Verification and Certificate Pinning.* [en línea]. [Consulta: 01/03/2019] https://blog.netspi.com/four-ways-bypass-android-ssl-verification-certificate-pinning/

[8] DOCUMENTACIÓN ANDROID. *Configuración de seguridad de la red*. [en línea]. [Consulta: 23/02/2019] https://developer.android.com/training/articles/security-ssl?hl=es-419

[9] DOCUMENTACIÓN ANDROID. *Seguridad con HTTPS y SSL.* [en línea]. [Consulta: 23/02/2019] https://developer.android.com/training/articles/security-config

[10] DOCUMENTACIÓN OKHTTP3. Class CertificatePinner. [en línea]. [Consulta 4/05/2019] https://square.github.io/okhttp/3.x/okhttp/okhttp3/CertificatePinner.html#check-java.lang.String-java.util.List-

[11] DATATHEOREM. *TrustKit-Android*. [en línea]. [Consulta: 06/06/2019] https://github.com/datatheorem/TrustKit-Android

[12] WIKIPEDIA. *Appcelerator Titanium.* [en línea]. [Consulta 06/06/2019] https://en.wikipedia.org/wiki/Appcelerator_Titanium

[13] RAONA. *¿App nativa, web o híbrida?* [en línea]. [Consulta: 06/06/2019] https://www.raona.com/aplicacion-nativa-web-hibrida/

[14] GO4ITSOLUTIONS. *Análisis dinámico de código vs análisis estático.* [en línea]. [Consulta: 07/06/2019] https://go4it.solutions/es/blog/analisis-dinamico-de-codigo-vs-analisis-estatico

[15] PARROTSEC. *Parrot Documentation*. [en línea]. [Consulta: 05/03/2019] https://www.parrotsec.org/docs/

[16] ANDROIDx86. *Run Android on your pc.* [en línea]. [Consulta: 5/03/2019] https://www.android-x86.org/

[17] APKTOOL. *A tool for reverse engineering Android apk files.* [en línea]. [Consulta: 01/04/2019] https://ibotpeaches.github.io/Apktool/

[18] FRIDA. *Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers.* [en línea] [consulta: 5/03/2019] https://www.frida.re/docs/home/

[19] MITMPROXY. *Introduction.* [en línea] [consulta: 5/03/2019] https://mitmproxy.readthedocs.io/en/v2.0.2/introduction.html

[20] MIMPROXY DOCS. *Transparaently proxify virtual machines*. [en línea]. [Consulta: 15/03/2019] https://docs.mitmproxy.org/stable/howto-transparent-vms/

[21] SCAZZOSI, Jacobo. *Inspecting Android´s HTTP traffic with mitmproxy*. [en línea]. [Consulta: 03/04/2019] https://treesandrobots.com/2018/03/debug-http-android-app-mitm.html

[22] PINDROP BLOG. *Bypassing the CA Restrictions in Android Nougat.* [en línea]. [Consulta 29/03/2019] https://www.pindrop.com/blog/bypassing-the-ca-restrictions-in-android-nougat/

[23] STACKOVERFLOW. *How to sign an already compiled apk*. [en línea]. [Consulta 02/04/2019] https://stackoverflow.com/questions/10930331/how-to-sign-an-already-compiled-apk

[24] V0X. *Bypassing SSL Certificate Pinning on Android for MITM attacks*. [en línea]. [Consulta: 17/04/2019] https://v0x.nl/articles/bypass-ssl-pinning-android/

[25] HOLDING, Jamie. *Advanced Certificate Bypassing in Android with Frida.* [en línea]. [Consulta: 19/04/2019] https://blog.jamie.holdings/2019/01/19/advanced-certificate-bypassing-in-android-with-frida/

[26] MARÍA-TOMÉ, Lucas, *Development of a system for traffic analysis of smartphone apps for private data exfiltration detection*. Trabajo Fin de Máster. ETSI Telecomunicación. Universidad Politécnica de Madrid. 2019.

[27] GENOVESE, Marco. *How to bypass Instagram SSL Pinning on Android v78*. [en línea]. [Consulta: 01/04/2019] https://plainsec.org/how-to-bypass-instagram-ssl-pinning-on-android-v78/

[28] DOCUMENTACIÓN ANDROID, *Reducir tu código y tus recursos.* [en línea] [Consulta: 06/06/2019] https://developer.android.com/studio/build/shrink-code?hl=es-419

ANEXO A: ASPECTOS ÉTICOS, ECONÓMICOS, SOCIALES Y AMBIENTALES

A.1 INTRODUCCIÓN

El Smartphone se ha convertido en un elemento imprescindible en la vida de prácticamente cada persona. Para prácticamente cada problema existe una aplicación que facilita la resolución de dicho problema. Sin embargo, lo que a priori parece de forma gratuita o inofensivo, en la época en la que predomina el big data, no lo es. En infinidad de aplicaciones los usuarios introducen sus correos, aceptan recibir publicidad constante, dan permisos a servicios del teléfono que la aplicación no necesita para funcionar, que simplemente necesita para recaudar datos. En definitiva, miles de usuarios están permitiendo que las compañías dueñas de las aplicaciones tengan acceso a sus gustos, hábitos, rutinas y elaboren perfiles en función de ellos.

Además, está más demostrado que, lo que en un primer lugar parece inofensivo (ceder unos cuantos datos sobre la vida personal), es determinante en procesos de los cuales el usuario no es consciente y que abarcan desde la publicidad que se recibe diariamente hasta influenciar en unas elecciones generales. Por ello es necesario conocer y controlar qué actividades realizan las compañías con nuestros datos para así intentar conservar la privacidad de cada persona.

¿Son realmente seguras las aplicaciones? ¿Utilizan los datos solo para acciones que el usuario consiente o se exceden en el uso de información privada?

Gracias al desarrollo del componente visto en este trabajo, se podrán auditar más fácilmente las comunicaciones que establecen las aplicaciones Android. De esta forma, se puede controlar que las aplicaciones no se exceden en la recolección de datos y simplemente guardan aquellos datos a los que acceden tras haber solicitado permiso al usuario.

A.2 DESCRIPCIÓN DE IMPACTOS RELEVANTES RELACIONADOS CON EL PROYECTO

A lo largo de este proyecto se han mostrado diferentes formas de modificar el código de las aplicaciones Android y de evitar las medidas de seguridad que ocultan el contenido de las conexiones con el objetivo de poder auditarlas. En definitiva, gracias a esto se puede tener acceso a la información personal de cada usuario que las aplicaciones están transmitiendo.

El desarrollo de este software y la exposición de estos conocimientos conllevan a un dilema ético, ya que, aunque el objetivo de este estudio es facilitar el trabajo que conlleva una auditoría, estos procedimientos se pueden emplear con fines maliciosos, obteniendo información sensible de usuarios de forma ilegal.

Debido al posible trato con datos personales de usuarios, todas las pruebas realizadas en este trabajo se han realizado con datos ficticios, sin llegar a invadir en ningún momento la privacidad de ninguna persona real.

Por otra parte, el uso malicioso de estos conocimientos también puede desembocar en un problema social, debido a que una persona con conocimientos suficientes puede modificar una aplicación para sus propios intereses y subirla a internet. Si esa aplicación modificada obtuviera un gran número de descargas, la seguridad de la información de los usuarios estaría comprometida. Por tanto, es importante siempre descargar aplicaciones oficiales y no cualquier archivo apk encontrado por Internet.

Para comprobar la modificación del código de una aplicación y realizar las pruebas de este trabajo se ha desarrollado una aplicación propia como se ha comentado a lo largo del trabajo, para así salvaguardar la integridad de una aplicación de terceros.

A.3 ANÁLISIS DETALLADO DE ALGUNO DE LOS PRINCIPALES IMPACTOS

Tal y como se mencionó en apartados anteriores, el único fin de este software es el de facilitar el trabajo realizado en una auditoría de seguridad. A la hora de realizar una auditoría de seguridad en una aplicación, se buscan fallos y vulnerabilidades en la aplicación a auditar. Esto puede dar lugar a encontrar fallos de diferentes tipos como errores de funcionamiento de la aplicación o el desarrollo de código no seguro. En el caso de este trabajo, se ha expuesto cómo las comunicaciones y la información que transmite una aplicación puede no realizarse de forma completamente segura y, por tanto, una aplicación puede estar dejando información sensible de un usuario a la vista de un atacante.

Cabe recalcar que para realizar las pruebas en las aplicaciones anteriormente expuestas se han utilizado siempre datos ficticios, de cara a conservar la privacidad de personas reales y no tratar con su información sensible.

Gracias a este software, se podrán detectar más fácilmente fallos en la seguridad de las aplicaciones a la hora de transmitir información personal de un usuario, por lo que será más fácil localizar el error en la aplicación y solventarlo, ayudando así a proteger la privacidad de la persona que utiliza la aplicación, evitando que su información personal caiga en manos de un atacante.

Sin embargo, los conocimientos teóricos aquí expuestos también pueden dar lugar al desarrollo de prácticas con intereses maliciosos que comprometan la información de los usuarios de los teléfonos móviles de última generación. Por ello se enfrenta un dilema ético a la hora de publicar esta información y desarrollar un software que facilite a cualquier persona acceder a las comunicaciones de una aplicación.

Pese a la existencia de este dilema, se considera que es más importante publicar la información. Al comienzo de este trabajo se mencionó que la seguridad 100% no existe, por lo que es importante comunicar los fallos y vulnerabilidades a los desarrolladores para que así sean capaces de solucionarlos, haciendo más segura la experiencia del usuario. De hecho, debido a la importancia que poseen los fallos de seguridad, las grandes compañías ofrecen programas de recompensa (*bug bounty*) a personas particulares que informan sobre fallos de seguridad en sus aplicaciones, contribuyendo a la seguridad de las mismas. Por tanto, el desarrollo de este software y posterior publicación, se pueden ver del mismo modo, ya que la intención es facilitar que un desarrollador pueda localizar ciertos puntos débiles en su aplicación y pueda solventarlos o que un auditor pueda encontrar más fácilmente las vulnerabilidades en el establecimiento de conexiones para posteriormente comunicarlas.

Por último, debido al posible uso indeseado que se puede dar a este software, se presentan como conclusiones en este documento unas guías de buen uso tanto como para desarrolladores como para usuarios, para generar conciencia y conseguir dificultar el trabajo a un atacante.

A.4 CONCLUSIONES

La finalidad de los componentes desarrollados en este proyecto es contribuir a la seguridad de la sociedad, asegurando un ambiente tecnológico en el que la privacidad de cada persona es respetada por las compañías que facilitan las diversas aplicaciones y herramientas.

Gracias a este software, los usuarios podrán confiar en que existe y se emplea un método para intentar limitar la recolección abusiva de datos personales e intentar controlar el cumplimiento del Reglamento General de Protección de Datos.

Por otra parte, tanto los desarrolladores como los auditores podrán contar con una herramienta que les facilite su trabajo y permita aumentar la seguridad en las conexiones que establecen las aplicaciones.

ANEXO B: PRESUPUESTO ECONÓMICO

Para realizar el presupuesto económico se ha tenido en cuenta la mano de obra (precio por hora y horas trabajadas) y el coste de un ordenador que tenga 16 GB de memoria Ram y permita levantar el entorno de trabajo sin problemas.

	horas	Precio/hora	TOTAL
COSTE DE MANO DE OBRA (coste directo)	250	15 €	**3.750 €**

COSTE DE RECURSOS MATERIALES (coste directo)

	Precio de compra	Uso en meses	Amortiz. en años	TOTAL
Ordenador personal (Software incluido).....	859,00 €	6	5	85,90 €

TOTAL	**85,90 €**

GASTOS GENERALES (costes indirectos)	15%	sobre CD	**575,39 €**
BENEFICIO INDUSTRIAL	6%	sobre CD+CI	**264,68 €**

SUBTOTAL PRESUPUESTO		**4.675,96 €**
IVA APLICABLE	21%	**981,95 €**
TOTAL PRESUPUESTO		**5.657,91 €**