universidad de león

Departamento de Matemáticas

MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN CIBERSEGURIDAD

Trabajo de Fin de Máster

Análisis estático y comparativa de apps iOS provenientes de diferentes mercados de apps

Static analysis and comparison of iOS apps coming from different app markets

Autor: Adrián Campazas Vega

Tutor: Ricardo Julio Rodríguez Fernández

(Septiembre, 2019)

UNIVERSIDAD DE LEÓN

Departamento de Matemáticas

MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN CIBERSEGURIDAD

Trabajo de Fin de Máster

ALUMNO: Adrián Campazas Vega

TUTOR: Ricardo Julio Rodríguez Fernández

TÍTULO: Análisis estático y comparativa de apps iOS provenientes de diferentes mercados de apps

TITLE: Static analysis and comparison of iOS apps coming from different app markets

CONVOCATORIA: Septiembre, 2019

RESUMEN:

Con la popularización de iOS, el número de cibercriminales interesados en obtener beneficios con la intrusión de malware en los dispositivos móviles de Apple ha aumentado considerablemente. El afán de la compañía por crear un sistema cerrado y con numerosas restricciones tanto a nivel de sistema operativo como en su tienda oficial de aplicaciones ha derivado en el surgimiento de una gran cantidad de mercados alternativos al App Store, el mercado oficial de aplicaciones de Apple. En este trabajo se ha realizado un estudio y clasificación de los diferentes mercados alternativos de aplicaciones iOS. De los 23 mercados estudiados, destaca que tan solo uno de ellos no ofrece réplicas de aplicaciones disponibles en el App Store. Asimismo, 13 de los mercados analizados no necesitan jailbreak, indicando que están dedicados en exclusiva a ofrecer aplicaciones de forma gratuita que en el mercado oficial son de pago. Además, se ha desarrollado un sistema para el análisis estático de dichas aplicaciones. Este sistema permite a un analista de seguridad visualizar la información más relevante de una aplicación iOS (apps) de forma automática. Además, se ha implementado también un comparador de aplicaciones de iOS utilizado para evaluar la similitud entre apps obtenidas del mercado oficial de Apple y apps descargadas de mercados alternativos, con el objetivo de determinar si dichas aplicaciones han sido modificadas y contienen código malicioso. El sistema desarrollado se ha evaluado con 14 aplicaciones procedentes de la App Store y del mercado alternativo AppCake, concluyendo que tan solo 4 aplicaciones no han sido modificadas. Además, gracias al comparador se comprobó que una de las aplicaciones analizadas incorporaba un sistema de cobro por publicidad no existente en la aplicación original.

Palabras clave: iOS, extracción de características, mercados alternativos, análisis estático.

Firma del alumno:

VºBº Tutor:

Digitally signed by RODRIGUEZ FERNANDEZ RICARDO JULIO - [redacted]

Date: 2019.09.05 22:23:25 +02'00'

Agradecimientos

Gracias a Ricardo J. Rodríguez, tanto por su aportación y seguimiento como por su comprensión a lo largo de todo el proyecto. Muchas gracias a mi familia, pareja y amigos por la paciencia y apoyo demostrado durante estos dos duros años de máster.

Abstract

With the popularization of iOS, the number of cybercriminals interested in getting benefits from infecting Apple's mobile devices with malware has increased considerably. The desire of Apple to create a closed system with numerous restrictions, both at the operating system level and at its official app market level, led to the birth of a large number of alternative markets to the App Store, the Apple's official app market. In this work, we have performed a study and classification of the different alternative markets of iOS applications. Only one of the 23 alternative markets analyzed in this work does not offer free replicas of the non-free applications available in the App Store. Moreover, 13 alternative markets do not need jailbreak, meaning that they are exclusively dedicated to offer for free applications which are licensed software in the official market. In addition, we have developed a system for static analysis that allows a security analyst to review the most relevant information of an iOS application (app) automatically. We have also implemented a similarity tool of iOS apps that enables us to compare apps obtained from the App Store and apps downloaded from alternative markets. As a result, we can easily determine whether the latter applications have been altered and contain malicious code. The developed system was evaluated in 14 applications downloaded from the App Store and from the alternative market AppCake. The results show that only 4 applications are unmodified. Furthermore, the similarity tool succeeded in detecting a pay-per-click system which was added into one of the applications under study.

Índice general

Índice de figuras

Índice de tablas

Glosario de términos

Ciberseguridad: Área relacionada con la informática que se enfoca en la protección de la información, de los sistemas informáticos que contienen dicha información y de los usuarios.

Vulnerabilidad: Debilidad o fallo de seguridad en un sistema o aplicación informática que puede permitir a un atacante comprometer la confidencialidad, integridad o disponibilidad de los datos contenidos o de los propios sistemas vulnerables.

iOS: Sistema operativo móvil desarrollado por Apple Inc.

Android: Sistema operativo móvil desarrollado por Google y basado en el núcleo de Linux. Android es software de código abierto.

Jailbreak: Es el proceso de suprimir algunas de las limitaciones de seguridad que Apple impone en sus dispositivos iOS. Para poder realizar un Jailbreak, se han tenido que encontrar vulnerabilidades en el sistema que permitan liberar el dispositivo.

Malware: En castellano programa malicioso, hace referencia a cualquier software maligno que trata de afectar a cualquier dispositivo electrónico con el objetivo en la gran mayoría de los casos de obtener rédito económico.

Protocolo SSH: El protocolo SSH permite acceso remoto a un servidor a través de canal seguro, toda la información entre el cliente y el servidor viaja cifrada.

Botnet: Una botnet hace referencia a un conjunto de robots o bots que se ejecutan de forma automática. El propietario de una botnet puede utilizarla para numerosos finaes como realizar ataques de denegación de servicio o minar criptomonedas entre otros usos.

Debugger: En castellano depurador, es un programa informático utilizado para probar y depurar errores de otros programas.

Spam: El spam hace referencia al correo que se considera basura. Este tipo de correo es no solicitado o deseado por el usuario que lo recibe. Habitualmente el spam es de tipo publicitario y quien lo enviar suele hacerlo de forma masiva.

Exploit: En castellano significa explotar o aprovechar. Un exploit es un fragmento de código o secuencia de comandos utilizada con el objetivo de aprovecharse de una vulnerabilidad de seguridad.

XNU: Es un núcleo desarrollado por NeXT e implementado por Apple en 1996 en su sistema operativo de escritorio macOS.

Objective-c: Lenguaje de programación orientado a objetos utilizado por Apple tanto en su sistema operativo móvil iOS como en el de escritorio macOS.

Python: Lenguaje de programación interpretado y multiparadigma, ya que suporta tanto programación orienta a objetos como programación imperativa.

Capítulo 1

Introducción

Según un estudio realizado el 21 de febrero de 2019, Apple es la segunda empresa con mayor capital bursátil del mundo [1]. La historia de la compañía se remonta al año 1976 cuando sale al mercado el dispositivo "Apple I". A día de hoy la compañía ofrece al mundo una amplia gama de productos que engloba desde sus famosos Mac Book hasta sus nuevos AirPods, sin olvidar por su puesto de su mayor baluarte, el dispositivo móvil iPhone.

Hasta el año 2015 Apple ha vendido más de 900 millones de unidades de su dispositivo móvil [2], como se muestra en la Figura 1.1. Gran parte del éxito del dispositivo radica en su sistema operativo, denominado iOS. Este sistema operativo móvil desarrollado por Apple vio la luz en enero del año 2007. Desde entonces, el sistema ha recibido numerosas mejoras y actualizaciones.

Desde su origen, iOS se concibió como un sistema operativo diseñado exclusivamente para los dispositivos de Apple, un sistema gestionado y limitado por dicha compañía. Esto posiblemente sea el motivo por el cual la penetración de mercado de iOS sea muy inferior a la de Android, un 11.1 % frente a un 88 % según datos del 2018 como se aprecia en la Figura 1.2. Además, para usar iOS es necesario disponer de un dispositivo Apple, ya sea iPhone o iPad, con el coste económico que esto supone. Otro factor que afecta a los usuarios a la hora de elegir un sistema u otro es la cantidad de restricciones que iOS tiene implementadas en lo que a personalización del sistema se refiere.

El modelo de seguridad diseñado por Apple para su sistema operativo iOS y la gran cantidad de restricciones impuestas a los desarrolladores a la hora de subir

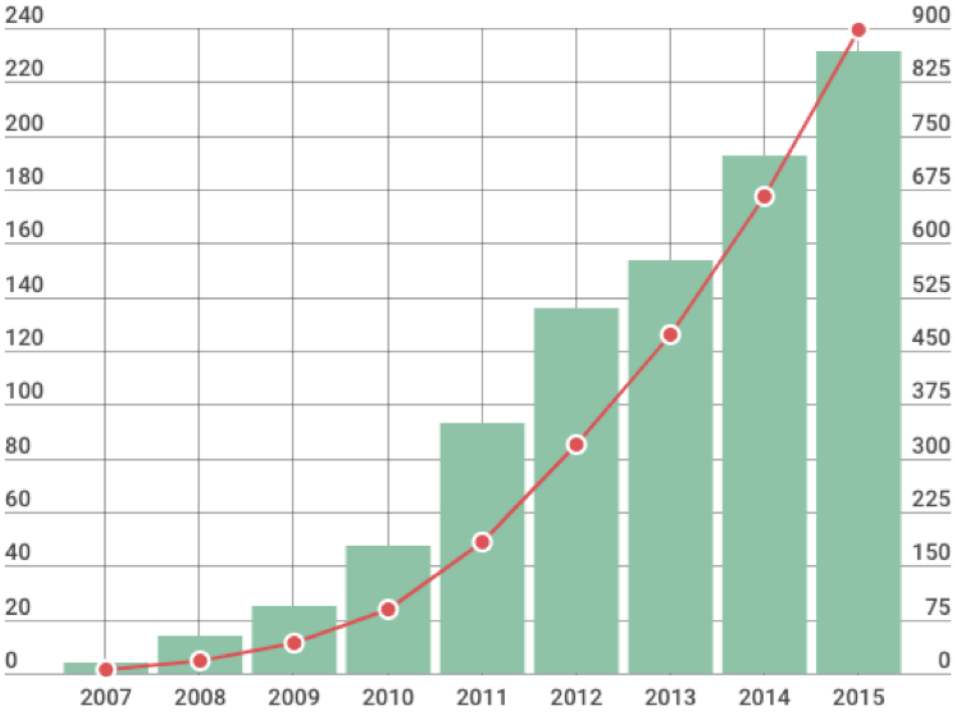
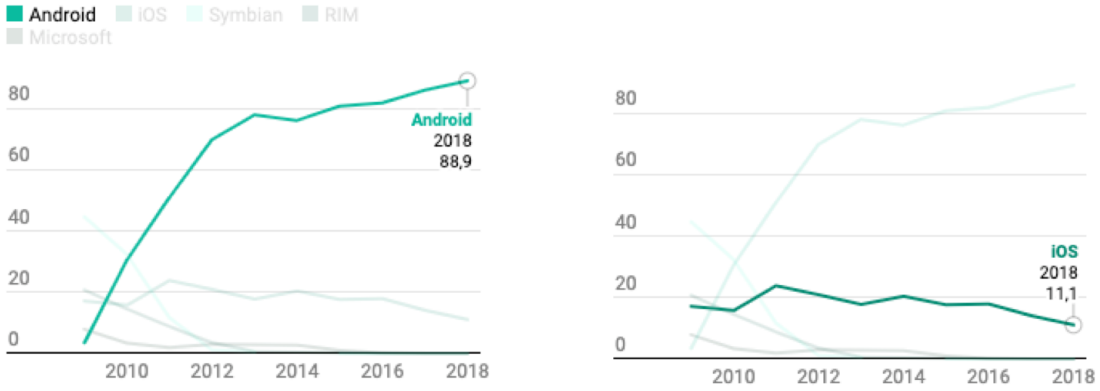


Figura 1.1: Crecimiento de ventas de Iphone
Fuente: *https://www.revistagq.com/noticias/tecnologia/articulos/cuantos-iphone-vendido-modelo/23617*

aplicaciones al App Store (mercado oficial de Apple para proveer de aplicaciones a sus dispositivos) otorga a iOS un nivel de seguridad elevado.

Además de censurar diferentes temas dentro del App Store (como violencia, xenofobia, racismo, etc), uno de los objetivos principales de las restricciones impuestas por Apple a la hora de publicar aplicaciones en su mercado oficial es evitar la presencia de aplicaciones maliciosas. Si bien es cierto que históricamente se ha demostrado que el App Store ha contenido muestras de malware debido a errores en el sistema de veto implementado, es mucho más sencillo para los desarrolladores de este tipo de aplicaciones subirlas a mercados alternativos que no cuentan con las defensas que el App Store implementa.

Un mercado alternativo es una tienda de aplicaciones no controlada por Apple y que por lo tanto, no cuenta con las medidas de seguridad implementadas por la compañía. Los mercados alternativos inicialmente tenían como objetivo albergar las aplicaciones que Apple no permitía subir a su mercado oficial. Rápidamente, además de este tipo de aplicaciones, comenzaron a llegar copias gratuitas de aplicaciones de pago del App Store. El número de aplicaciones de este tipo que son distribuidas gratuitamente, sin control y sin pasar medidas de seguridad, suponen un posible



(a) (b)

Figura 1.2: Cuota de mercado desde 2010 hasta 2018 de (a) Android e (b) iOS.

riesgo para la seguridad de los usuarios que consumen aplicaciones de estos mercados. Los mercados alternativos de aplicaciones iOS se convierten por tanto en plataformas perfectas para albergar aplicaciones maliciosas.

En este trabajo, en primer lugar se va a realizar un estudio de los diferentes mercados alternativos que albergan aplicaciones iOS (también llamadas apps). Después, se van a analizar las aplicaciones disponibles en estos mercados. Para ello, se ha implementado un sistema de análisis estático para aplicaciones iOS. El sistema implementado obtiene las características más relevantes desde el punto de vista de la ciberseguridad. Además, se ha implementado un comparador que permite analizar diferentes aplicaciones y compararlas entre sí, pudiendo así estudiar las diferencias entre una app legítima del App Store y sus homólogas procedentes de mercados alternativos. Esta herramienta comparativa también se puede utilizar para comparar versiones diferentes de la misma aplicación y obtener así los cambios entre versiones.

El resto de la memoria se estructura como sigue: El Capítulo 2 hace referencia a los conocimientos previos necesarios para entender este trabajo de fin de máster. Después, el Capítulo 3 presenta un estudio de los diferentes mercados alternativos de aplicaciones iOS y una clasificación de los mismos. En el Capítulo 4 se presentan las diferentes implementaciones llevadas a cabo para obtener el objetivo del proyecto. Además, el Capítulo 5 presenta los resultados obtenidos por las herramientas elaboradas, aplicándolas sobre aplicaciones provenientes de diferentes mercados alternativos. En el Capítulo 6 se detallan los trabajos relacionados con el proyecto. Por último, el Capítulo 7 corresponde a las conclusiones obtenidas y trabajos futuros. Además, se ha incluido el Anexo A en el que se desarrollan las tareas llevadas a cabo a lo largo del proyecto y su planificación.

Capítulo 2

Conocimientos previos

Este capítulo describe los conocimientos previos necesarios para entender el proyecto desarrollado. En primer lugar, se presenta el modelo de seguridad de iOS. Después, se expone la revisión de aplicaciones que realiza el App Store. Posteriormente, se introduce el concepto de jailbreak. Además, se define la estructura básica de una aplicación desarrollada para iOS. Por último, se muestra la estructura que tiene una aplicación instalada en iOS.

2.1. iOS y su modelo de seguridad

iOS está basado en Darwin, un sistema operativo de código abierto desarrollado por Apple. El núcleo del sistema operativo de Darwin es XNU, un núcleo híbrido que combina componentes de los núcleos de Mach y FreeBSD [3].

El esquema de seguridad de iOS se divide en dos partes claramente diferenciadas, el sistema de ficheros y el núcleo. El sistema de ficheros cuenta con dos capas, por un lado la capa de aplicación, en la que todas las aplicaciones se encuentran encapsuladas en una sandbox (explicada más adelante), y por otro lado, la partición de iOS o capa de sistema. La segunda capa es el núcleo que utiliza el co-procesador Secure Enclave (explicado más adelante). Además, como se puede ver en la Figura 2.1, el co-procesador Secure Enclave se encuentra totalmente aislado del sistema de ficheros.

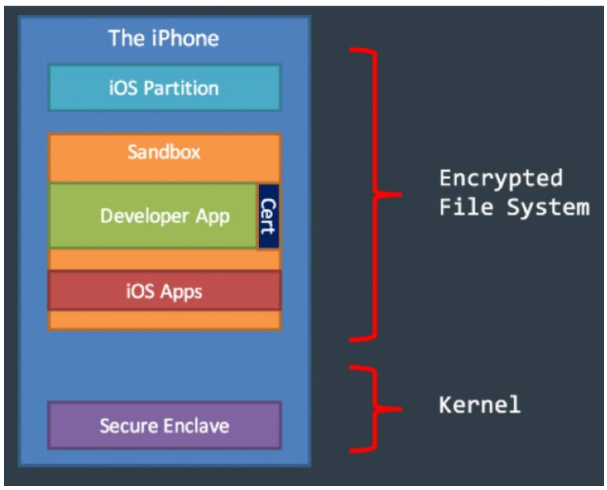


Figura 2.1: Esquema de seguidad de iOS
Fuente: *https://www.slideshare.net/InfoQ/modern-ios-application-security*

Las medidas de seguridad que Apple implementa en el conjunto del dispositivo se pueden dividir en los siguientes grandes grupos, descritos a continuación: *Hardware Security*, *Secure Boot*, *Code Signing*, *Sandbox*, *Encryption and Data Protection* y Mitigaciones de Explotación de Código Nativo.

2.1.1. Hardware Security

En el año 2013 Apple lanzó al mercado el procesador A7. La gran novedad de este chip es la incorporación de un segundo procesador dedicado exclusivamente a la seguridad. Este procesador es conocido como Secure Enclave [4].

Este co-procesador utiliza un micro-núcleo propio al cual no se puede acceder desde el sistema operativo ni desde ninguna aplicación. Cuenta con 4MB de almacenamiento y es utilizado únicamente para guardar claves de curva elíptica de 256 bits.

El co-procesador almacena 2 claves diferentes: una clave conocida como GID, que se carga en el chip en el momento de la fabricación del dispositivo y es compartida por todos los procesadores de una clase de dispositivos. Esta clave se utiliza para evitar la manipulación del *firmware* y para otras tareas no relacionadas directamente con los datos privados del usuario. La segunda clave que almacena es conocida como UID. Esta clave es exclusiva de cada dispositivo y se utiliza para proteger la jerarquía de claves y el sistema de ficheros a nivel de dispositivo [3].

2.1.2. Secure Boot

Cuando se enciende un dispositivo, iOS lee las instrucciones iniciales de una memoria tipo ROM (de solo lectura) para arrancar el sistema. Esta ROM de inicio contiene la clave pública de autoridad de certificación de Apple. Una vez que se ha verificado que el cargador de arranque de bajo nivel (*low-level bootloader*, LLB) ha sido firmado por Apple, lo inicia. El LLB realiza algunas tareas básicas y da comienzo a la segunda fase del arranque del sistema denominada iBoot. Cuando se inicia iBoot, el dispositivo puede entrar en modo de recuperación o iniciar el núcleo. Después de que iBoot verifica que el núcleo también se encuentra firmado por Apple, comienza el proceso de arranque del sistema cargándose los controladores e iniciándose los demonios del sistema [5]. Todo este proceso se puede apreciar de forma esquematizada en la Figura 2.2.

El propósito de Apple con este procedimiento es garantizar que todos los componentes del sistema han sido firmados y distribuidos por Apple y no por terceros que hayan podido añadir código malicioso.

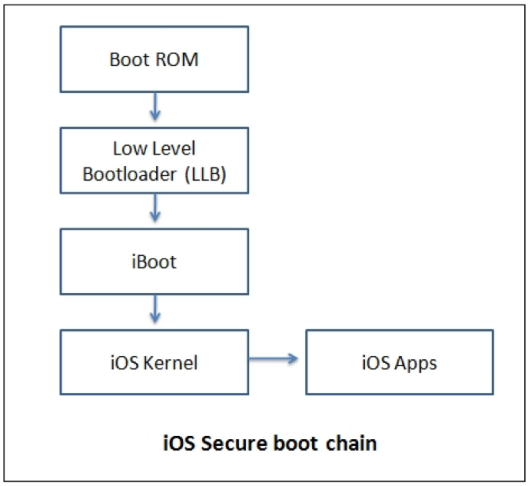


Figura 2.2: Cadena de arranque de iOS
Fuente: *https://subscription.packtpub.com/book/networking_and_servers/9781785883255/1/ch01lvl1sec13/ios-secure-boot-chain*

2.1.3. Code Signing

Code signing es una de las principales razones por lo que muchos investigadores y atacantes han tratado a lo largo de la historia de romper la seguridad de los dispositivos iOS. Apple ha implementado un sistema de cifrado con el objetivo de que solo el código aprobado por Apple a través del App Store pueda ejecutarse en

un dispositivo iOS. Por lo tanto, los usuarios finales solo pueden usar aplicaciones del mercado oficial.

Además de que el código se encuentre firmado, iOS solo ejecuta las páginas de memoria que provengan de fuentes firmadas. Esto significa que las aplicaciones no pueden modificar su comportamiento dinámicamente o actualizarse en tiempo de ejecución. Esta protección complica notablemente la ejecución de *exploits* en el sistema. Muchos *exploits* tratan de descargar, instalar y ejecutar nuevas aplicaciones o funcionalidades adicionales que son bloqueadas por el sistema al no encontrarse firmadas [6].

Cabe comentar que en febrero de 2019 Apple ha sufrido un fallo en los certificados empresariales que permiten instalar aplicaciones sin cifrar en dispositivos iOS. Estos certificados van más allá de los certificados estándar de desarrollador (requisito necesario para publicar aplicaciones en el App Store) y permiten distribuir aplicaciones fuera del App Store [7].

2.1.4. Sandbox

El concepto de *sandbox* se encuentra muy extendido en sistemas operativos móviles. Una *sandbox*, también llamada caja de arena o contenedor, consiste en la ejecución de aplicaciones dentro del sistema de forma aislada. Estas "cajas de arena" permiten un control más preciso de las acciones que pueden realizar los procesos dentro del sistema. En el caso de iOS, se usa un sistema basado en permisos. Por ejemplo, la cámara de un dispositivo podrá acceder a la galería de imágenes o escribir en la tarjeta de memoria, pero no podrá acceder a los mensajes de texto almacenados en el dispositivo. Todas las aplicaciones de terceros se ejecutan bajo el mismo usuario (`mobile`, en el caso de iOS), y solo unas pocas aplicaciones y servicios del sistema se ejecutan como `root`. Las aplicaciones de iOS se limitan a su propio contenedor que permite únicamente el acceso a los propios archivos de la aplicación y a un número muy limitado de APIs del sistema para interaccionar con algunas de las funcionalidades del dispositivo.

El uso de sandbox tiene dos efectos importantes a nivel de seguridad en el sistema. Por un lado, limita el daño que una aplicación maliciosa puede causar en el dispositivo. Un *malware* que haya conseguido pasar los controles de seguridad del App Store y haya sido instalado correctamente en un dispositivo podrá robar las imágenes o enviar mensajes de texto, pero no podrá realizar ambas funciones gracias al sistema de permisos mencionado anteriormente, ya que a la hora de subir

una aplicación al App Store el desarrollador tiene que definir con qué objetivo concreto solicita permisos al usuario. Si esta solicitud no está debidamente justificada, el App Store rechazará la publicación de la aplicación. Por otro lado, si un atacante encuentra una vulnerabilidad dentro de una aplicación que se ejecuta dentro de su "caja de arena", la explotación estará limitada a la "caja de arena" de la aplicación y no afectará al resto del sistema.

2.1.5. Encryption and Data Protection

Apple lideró el cifrado de los datos dentro de los dispositivos móviles. iOS cifra el disco por completo y además proporciona a los desarrolladores la API de protección de datos (denominada *Data Protection*) para proteger aún más sus archivos. Estos dos mecanismos protegen los datos del usuario permitiendo borrar el dispositivo en remoto en el caso de pérdida o robo. Estos mecanismos, además, evitan que un atacante pueda desmontar físicamente el disco de un dispositivo y montarlo en otro para obtener los datos. Obsérvese que sin embargo esta medida de seguridad no evita el robo de datos si el dispositivo se encuentra en ejecución, ya que los archivos se encuentran descifrados durante la ejecución de las aplicaciones. Por lo tanto, un atacante que consiga autenticarse en el sistema podrá acceder a los datos del dispositivo sin impedimentos [6].

Todos los dispositivos iOS modernos utilizan discos de estado sólido. Estos discos implementan numerosos mecanismos de reducción de desgaste. Estos mecanismos eliminan todas las garantías de que cuando se sobrescribe un dato dentro del sistema, este dato también se esté sobrescribiendo en la ubicación física que le corresponde. Con el objetivo de asegurarse de que si se elimina un dato en el sistema este dato también se esté eliminando físicamente en el disco, iOS cifra los archivos con claves de cifrado. De este modo, cuando un archivo es eliminado también se elimina su clave de cifrado. Así, el sistema se asegura de que el dato eliminado no va a poder ser recuperado aunque físicamente siga en el disco. [6].

2.1.6. Mitigaciones de Explotación de Código Nativo

iOS implementa dos mecanismos estándar para ayudar a evitar ataques de ejecución de código.

El primer mecanismo es la aleatorización del espacio de direcciones de memoria, denominado ASLR (*Address Space Layout Randomization*). ASLR asigna al azar la ubicación de la memoria del ejecutable y los datos del programa en cada ejecución.

Las librerías del sistema que son utilizadas por múltiples procesos son asignadas cuando se inicia el dispositivo y se mantienen en el mismo lugar hasta que este se reinicia o se apaga. Esta protección hace que las direcciones de memoria específicas de cada función sean difíciles de predecir, dificultando la explotación de vulnerabilidades.

El segundo mecanismo es el bit XN (*Execute Never*). Este mecanismo permite al sistema operativo marcar segmentos de memoria como no ejecutables. En iOS este bit se aplica por defecto tanto al heap como a la pila, las dos estructuras de memoria normalmente usadas por las aplicaciones durante su ejecución para almacenar datos. De esta forma, si un atacante ha conseguido escribir código en la pila o bien en el heap no podrá redirigir la ejecución del programa al código insertado.

2.2. Revisión de Aplicaciones en App Store

Además de las protecciones de seguridad que iOS implementa a nivel de sistema, cabe comentar las medidas de seguridad que la App Store establece a la hora de subir una aplicación al mercado oficial de Apple.

Apple no divulga públicamente las técnicas que utiliza para evaluar las aplicaciones que son candidatas a ser distribuidas en su App Store. Un artículo reciente en la CNBC detalla que, además de las pruebas automáticas, todas las aplicaciones que tratan de subirse al mercado oficial de Apple y las actualizaciones de las aplicaciones ya subidas son analizadas manualmente por un empleado de la compañía. De esta forma, además de los controles de seguridad establecidos previamente, Apple controla qué tipo de contenido se encuentra disponible en la App Store y rechaza las aplicaciones que no cumplen con su estándar tanto a nivel de seguridad como de calidad y contenidos [8].

2.3. Jailbreak

Hacer un jailbreak a un dispositivo supone atacar la cadena de confianza establecida por Apple en el arranque del dispositivo, deshabilitando los mecanismos de firma de código de iOS y permitiendo que un dispositivo ejecute aplicaciones no aprobadas por Apple o lo que es lo mismo, permitiendo a un dispositivo iOS ejecutar aplicaciones que no han sido obtenidas a través del App Store.

Es importante tener en cuenta que al realizar un jailbreak a un dispositivo no necesariamente se desactiva la "caja de arena" proporcionada por iOS, si no que per-

mite disponer de aplicaciones fuera de esa caja de arena. Las aplicaciones instaladas a través del App Store se siguen instalando en la misma ubicación que si el dispositivo no tiene jailbreak, y por lo tanto están sujetas a la *sandbox* proporcionada por Apple. Las aplicaciones de terceros que no se adquieren a través del mercado oficial se instalan junto a las aplicaciones nativas pre-instaladas por Apple, obteniendo así un mayor número de permisos dentro del sistema [6].

Hay que tener en cuenta que con la actualización de Xcode a su versión 7, Apple por primera vez ha permitido a los desarrolladores instalar sus aplicaciones en dispositivos reales de forma gratuita sin que estas tengan que pasar por el App Store. Estas aplicaciones tienen ciertas limitaciones como que a los 7 días después de su instalación dejan de funcionar. Por descontado, las aplicaciones instaladas a través de Xcode se instalan dentro de su *sandbox* [9].

La historia del jailbreak está estrechamente relacionada con la historia del sistema operativo móvil de Apple, ya que desde los inicios de iOS con su versión 1 ha estado acompañado de un jailbreak. Realizar un jailbreak a un dispositivo iOS proporciona al usuario gran cantidad de utilidades que por defecto el sistema operativo de Apple no permite, como por ejemplo instalar un intérprete de Python o un servidor SSH.

Actualmente, la última versión de iOS disponible en el momento de escribirse este documento es la 12.4, pudiéndose realizar jailbreak a dispositivos iOS que monten el procesador A7 hasta la versión 12.2 y hasta la versión 12.1.2 para los dispositivos con procesador desde el A7 hasta el A12.

A medida que ha evolucionado la seguridad de los dispositivos móviles de Apple también ha evolucionado las diferentes técnicas de jailbreak. Se pueden diferenciar tres tipos de jailbreak: *Tethered*, *Semi-Tethered* y *Untethered*.

Jailbreak tethered: Es el jailbreak menos recomendado, ya que cada vez que el dispositivo se reinicia o se apaga este no volverá a arrancar hasta que se conecte a un ordenador y se vuelva a realizar el proceso de jailbreak.

Jailbreak semi-tethered: El jaibreak se pierde cuando el dispositivo se apaga o se reinicia. La diferencia con el jailbreak tethered es que no se mantiene en un bucle al arrancar si no que inicia normalmente con el sistema sin jailbreak. Este tipo de jailbreak es el más común en la actualidad.

Jailbreak untethered: Este tipo de jailbreak persiste aunque el dispositivo se apague o se reinicie.

2.4. Aplicaciones iOS: formato IPA

Al igual que en Android, iOS empaqueta sus aplicaciones en un formato propio, denominado IPA (*iOS App Store Package*). Este tipo de archivos contiene todos los ficheros necesarios para que una app pueda ser instalada en un dispositivo iOS.

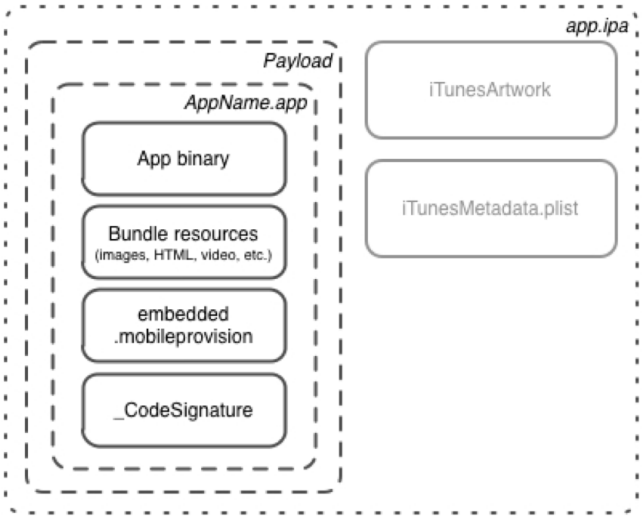


Figura 2.3: Estructura de una aplicación iOS
Fuente: *http://blog.dornea.nu/2014/10/29/howto-ios-apps-static-analysis/*

En la Figura 2.3 se puede apreciar la estructura que tiene una aplicación en iOS. Los ficheros que contiene un archivo IPA son los siguientes:

- `/Payload/`: Esta carpeta contiene todos los datos de la aplicación.
- `/Payload/Application.app`: Esta carpeta contiene los archivos propios de la aplicación. Entre ellos se encuentra el código compilado o los diferentes recursos de la aplicación.
- `/iTunesArtwork`: Contiene la imagen que se utiliza como icono de la aplicación.
- `/iTunesMetadata.plist`: Contiene información acerca del desarrollador,el nombre de la aplicación, el identificador de la aplicación en el App Store y la clasificación de la aplicación en función de su temática, entre otros datos [3].

Todos los ficheros IPA tienen la misma estructura si provienen del App Store. Sin embargo, si son descargados de mercados alternativos es posible que los ficheros `iTunesArtwork` y `iTunesMetadata.plist` no se encuentren presentes.

Si se profundiza dentro del directorio `Application.app`, los ficheros más relevantes son:

- Binario: Contiene el código compilado de la aplicación. Hay que tener en cuenta que si la aplicación proviene del App Store estará cifrada por Apple.
- `Application`: Contiene los iconos de la aplicación.
- `Info.plist`: Contiene la información de configuración de la aplicación. Desde el punto de vista de la ciberseguridad, contiene información relevante como los permisos que la aplicación solicita al usuario.
- `Launch images`: Contiene las imágenes que la aplicación cargará al iniciarse.
- `Settings.bundle`: Contiene las preferencias específicas de la aplicación.
- `Mobileprovision`: Contiene información del certificado de desarrollador, dispositivos compatibles y un identificador proporcionado por Apple.
- `_CodeSignature`: Almacena las firmas de los ficheros de la app. La firma de código consta de tres partes: un sello, una firma digital y los requerimientos en la firma, que tienen como objetivo evaluar la firma de código. [10].
- `Language.lproj`: Esta carpeta contiene dos tipos de ficheros. Por un lado se encuentran los *storyboards*. Un *storyboard* contiene los elementos gráficos que la aplicación muestra al usuario. Por otro lado, contiene archivos de cadenas para los diferentes idiomas que son compatibles con la aplicación.

2.5. Estructura de una app instalada en iOS

Hasta la versión 7 del sistema operativo móvil de Apple las aplicaciones instaladas a través del mercado oficial se encontraban en la carpeta `/var/Applications/`. Con la llegada de iOS 8, se modificó cómo se almacenaban estas aplicaciones.

Las aplicaciones comenzaron a almacenarse dentro de una carpeta cuyo nombre es un identificador aleatorio de 128 bits UUID (*Universal Unique Identifier*). El contenido estático de la aplicación y los datos que la aplicación guarda en el sistema se almacenan en ubicaciones diferentes [6]:

- `/var/mobile/Containers/Bundle/Application/[UUID]/Application.app`: contiene los datos de `Application.app` mencionados anteriormente, y almacena el contenido estático y el binario compilado de la aplicación. El contenido

de esta carpeta se utiliza para validar la firma del código. A partir de la versión 9.3 de iOS, la ruta ha cambiado a `/var/containers/Bundle/Application/`.

- `/var/mobile/Containers/Data/Application/[UUID]/Document`: Contiene todos los datos generados por el usuario al utilizar la aplicación.
- `/var/mobile/Containers/Data/Application/[UUID]/Library`: Contiene todos los archivos que no son propios del usuario como *cookies* o archivos de configuración.
- `/var/mobile/Containers/Data/Application/[UUID]/tmp`: Contiene los archivos temporales de la aplicación.

En la Figura 2.4 se puede apreciar la distribución de los archivos de una aplicación iOS instalada en el sistema.

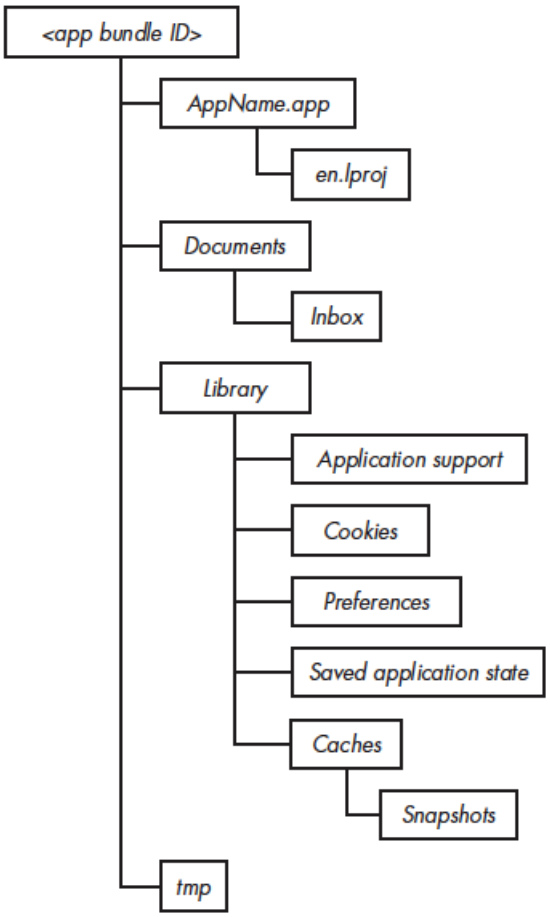


Figura 2.4: Distribución de archivos en una aplicación en iOS
Fuente: *[5]*

Capítulo 3

Estudio de mercados alternativos

En este capítulo se ha realizado la investigación, el estudio y la clasificación de los diferentes mercados alternativos que contienen aplicaciones iOS en la actualidad.

A medida que el sistema operativo móvil iOS ha alcanzado popularidad entre los usuarios, han comenzado a surgir mercados de aplicaciones alternativos al App Store. Estos mercados nacieron para cubrir ciertas características que el App Store no proporciona a los usuarios de Apple. Entre esas características cabe destacar que los mercados alternativos ofrecen aplicaciones gratuitas que son de pago en el mercado oficial. Este aspecto va en contra de la propiedad intelectual tanto de desarrolladores independientes como de las empresas profesionales del sector, pero al mismo tiempo poder obtener aplicaciones o juegos de pago de forma gratuita es sin duda uno de los mayores reclamos de este tipo de mercados. Por otro lado, es muy común encontrar aplicaciones o juegos "vitaminados", es decir, aplicaciones con mejores características que las aplicaciones originales. Por ejemplo, en el caso de videojuegos, tener cartas legendarias de forma gratuita o monedas infinitas; en el caso de aplicaciones como Spotify, tener características de usuarios premium con cuentas gratuitas. Además de aplicaciones, este tipo de mercados proporcionan a los usuarios gran cantidad de herramientas para aumentar la utilidad de su iPhone o iPad con jailbreak instalado. Este tipo de mejoras son conocidas como *tweaks*.

A la hora de clasificar los diferentes mercados alternativos de aplicaciones para dispositivos iOS se han de tener en cuenta los siguientes criterios: *¿requiere jailbreak?*, *¿contiene réplicas de aplicaciones publicadas en el App Store?*, *¿requiere registro?* y *¿es gratuito?*. Los criterios establecidos se han seleccionado con el objetivo de comprobar cuál es el comportamiento de los mercados alternativos de aplicaciones y se pueden responder mediante una respuesta afirmativa o negativa. En la Tabla 3.1

se describe la clasificación de los 23 mercados alternativos estudiados, atendiendo a estos criterios establecidos.

Tabla 3.1: Clasificación de los mercados alternativos estudiados

Nombre	**Requiere Jailbreak**	**Contiene réplicas App Store gratuitas**	**Requiere registro**	**Es gratuito**
Tuto App [11]	No	Sí	En función de la versión	En función de la versión
VShare [12]	No	Sí	No	Sí
Zestia Extender [13]	No	Sí	No	Sí
Apps4iPhone [14]	Sí	Sí	No	Sí
AppValley [15]	No	Sí	No	Sí
Tweakbox [16]	No	Sí	No	Sí
Appcake [17]	Sí	Sí	No	Sí
51ipa [18]	Sí	Sí	No	Sí
25 pp [19]	Sí	Sí	No	Sí
Tongbu [20]	Sí	Sí	Sí	Sí
App 704 [21]	No	No	Sí	Sí
Mojo installer [22]	Sí	Sí	No	Sí
Ipa Box [23]	No	Sí	No	Sí
App China [24]	No	Sí	No	Sí
AppEven [25]	No	Sí	No	Sí
HipStore [26]	Sí	Sí	No	Sí
TopStore [27]	No	Sí	No	Sí
iNoJB [28]	No	Sí	No	Sí
Emus4U [29]	No	Sí	No	Sí
IpaStore [30]	En función de la versión	Sí	No	En función de la versión
Lee890 [31]	Sí	Sí	No	Sí
Asterix installer [32]	No	Sí	No	Sí
Taigone Tikiri [33]	Sí	Sí	No	Sí

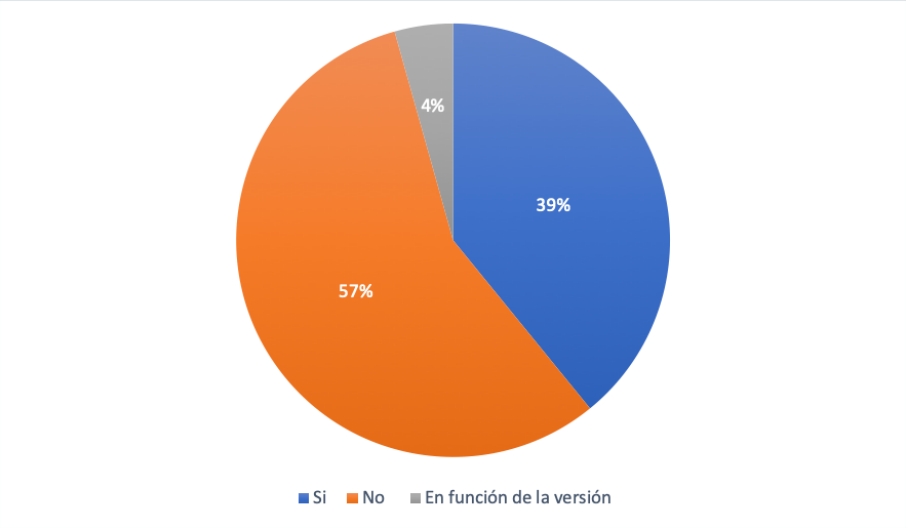


Figura 3.1: Mercados alternativos que necesitan jailbreak

Como se puede ver en la Figura 3.1, solo requieren de jailbreak un 43 % frente al 57 % de los mercados alternativos estudiados. En la Figura 3.2, se puede apreciar como el 96 % de los mercados analizados ofrecen aplicaciones que ya se encuentran en el App Store. Además, la Figura 3.3 refleja que solo el 9 % de los mercados alternativos requieren que el usuario se registre.

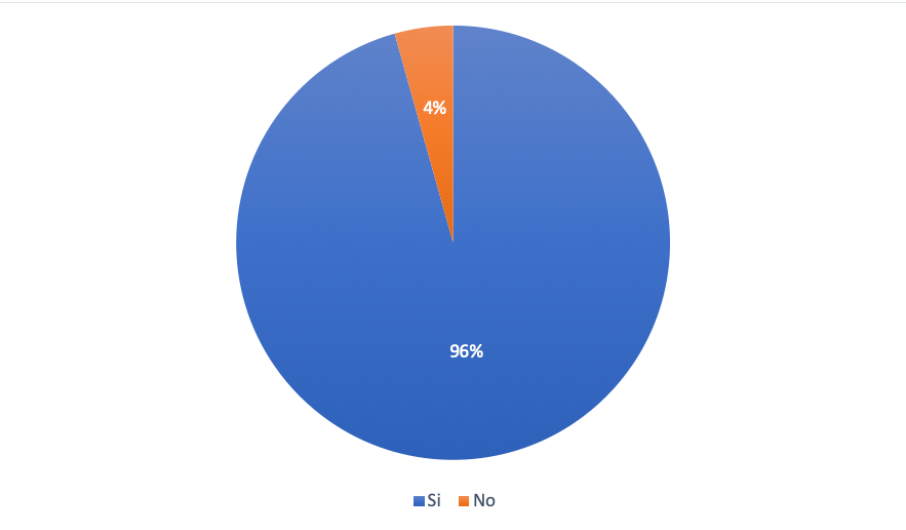


Figura 3.2: Mercados alternativos que ofrecen aplicaciones que se encuentran en el App Store

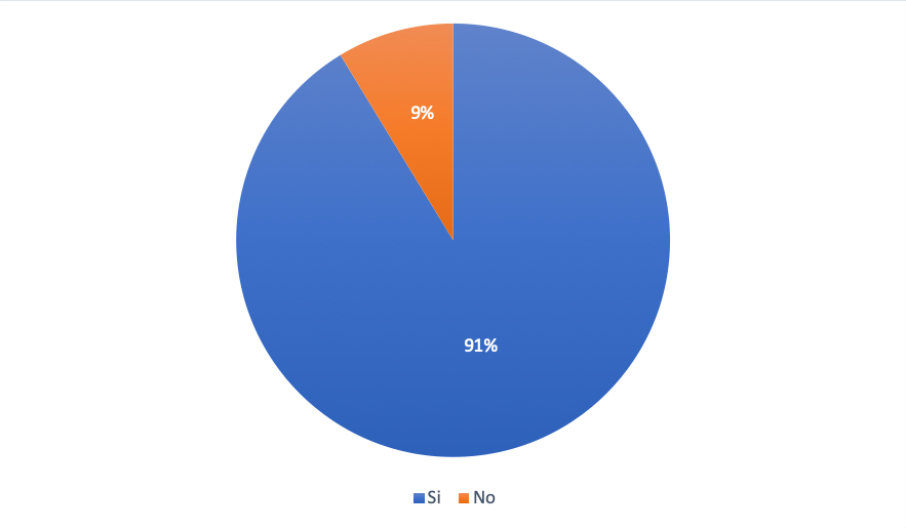


Figura 3.3: Mercados alternativos que requieren registro de usuario

De los datos obtenidos anteriormente se puede extraer dos conclusiones generales. La primera de ellas es que de los 23 mercados analizados tan solo uno de ellos no ofrece réplicas de aplicaciones disponibles en el App Store, bien sea debido a que las aplicaciones que proporciona han sido rechazadas por la restrictivas políticas del App Store o bien porque el propio desarrollador no ha deseado publicarlas en el mercado oficial. La segunda conclusión es que 13 de los 23 mercados no necesitan jailbreak. Estos 13 mercados están dedicados en exclusiva a ofrecer aplicaciones de forma gratuita que en el mercado oficial son de pago. Por lo tanto, un usuario que utilice esos mercados no podrá instalar nuevas funcionalidades en el sistema, que es uno de los motivos por los que el jailbreak surgió inicialmente.

Capítulo 4

Sistema de análisis y de comparación

En este capítulo se describen los desarrollos implementados. En primer lugar, se describe el sistema de análisis estático y las características que obtiene, mostrando además un caso de uso sobre una aplicación real. Por último, se presenta el comparador y su funcionamiento.

4.1. Sistema de análisis

Debido a la dificultad intrínseca que conlleva analizar una aplicación desarrollada en iOS, se ha implementado un sistema de análisis estático semi-automático, dado que es necesario proporcionar al sistema el archivo IPA con el binario sin cifrar. En las fases iniciales de este trabajo, se intentó implementar un sistema completamente automático que fuera capaz de obtener el fichero IPA con el código descifrado y realizara el análisis de forma automática. Se descartó esta opción debido a la escasa compatibilidad que el proyecto ofrecería, ya que para poder ejecutar la aplicación, se necesitaría una versión de iOS específica con un jailbreak concreto. Se tomó entonces la decisión de desarrollar el sistema de análisis de forma que pueda ser usado tanto en una versión móvil que se ejecute directamente en un dispositivo iOS, como en una versión de escritorio que se pueda ejecutar en un dispositivo con macOS instalado.

4.1.1. Descripción del sistema

La implementación del sistema de análisis se ha llevado a cabo con la intención de desarrollar un sistema escalable. Por este motivo, cualquiera de las características que la herramienta extrae es totalmente independiente de las otras, facilitando

así la opción de añadir, eliminar o modificar cualquiera de los módulos incluidos actualmente en la aplicación.

En la Figura 4.1 se puede ver el diagrama de sistemas de la aplicación. El sistema recibe un fichero IPA el cual descomprime para obtener el binario y el fichero `Info.plist` en la carpeta de trabajo. En este punto de la ejecución comienzan a ejecutarse los 5 subsistemas encargados de extraer las características de la aplicación analizada. Una vez finalizado el análisis, se genera el reporte y se escribe la información a un fichero de texto. El proyecto se ha desarrollado en Python 3 utilizando orientación a objetos. El código desarrollado se ha publicado de forma libre y gratuita con licencia GNU/GPLv3 en https://github.com/gripapc/Static-analysis-and-comparator-of-iOS-Applications-.

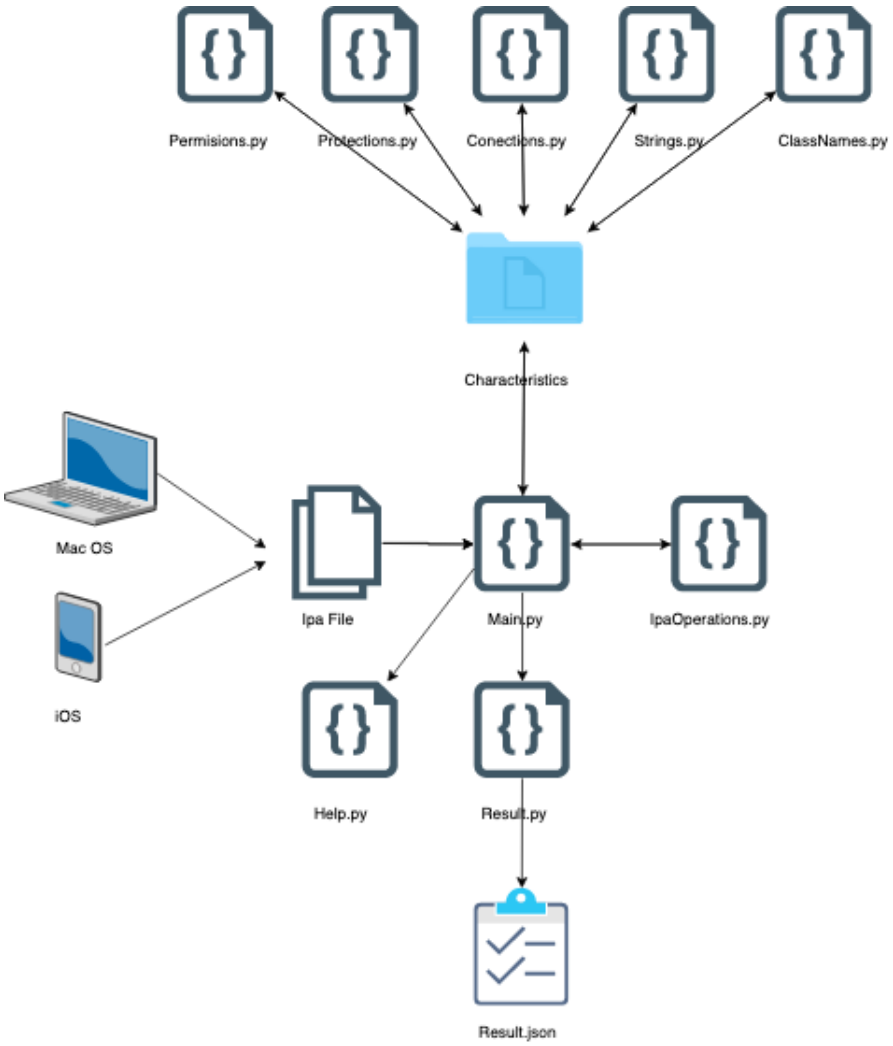


Figura 4.1: Diagrama de sistemas de la aplicación

Una vez finalizado el análisis de la aplicación seleccionada, el analizador generará un archivo en formato JSON con los resultados del análisis. JSON es un formato ligero de intercambio de datos. Se ha elegido JSON como formato de salida de los datos debido a que en los últimos años se ha convertido en un estándar en el mundo de la informática, más aún si cabe en el ámbito de la ciberseguridad ya que multitud de herramientas aceptan este tipo de formato como entrada de datos [34].

4.1.2. Características extraídas

Se han seleccionado un conjunto de características que pueden ayudar a un analista a determinar si una aplicación iOS puede tener un comportamiento malicioso. Las características seleccionadas se pueden dividir en 6 grupos que se explican a continuación.

Protecciones

Se han seleccionado tres características que muestran las protecciones que tiene el binario:

- *Canary*: Es una protección de la pila frente a desbordamientos de buffer. El sistema aloja un número aleatorio entero justo antes del retorno del puntero de retorno de la pila para detectar si se ha sobrescrito durante la ejecución del programa, abortando la ejecución del proceso si así sucede [35] .
- *Bit NX*: Esta medida de seguridad garantiza que ciertas áreas de la memoria como la pila o el heap no se puedan ejecutar (explicada anteriormente en la Sección 2.1.6).
- *Crypto*: Muestra si el código del binario se encuentra cifrado.

Esta información se ha extraído con la herramienta rabin2, de la suite de herramientas proporcionadas por radare2 [36]. En la Figura 4.2 se puede apreciar toda la información mostrada por rabin2 de un binario iOS. Se han remarcado las tres características recogidas.

```
[MacBook-Pro-de-gripapc:Ipas y JSON Actuales gripapc$ rabin2 -I netflix.ipa
baddr    0x0
binsz    33570495
bits     64
canary   false
crypto   false
endian   little
havecode false
laddr    0x0
linenum  false
lsyms    false
maxopsz  16
minopsz  1
nx       false
pcalign  0
pic      false
relocs   false
sanitiz  false
static   true
stripped false
va       false
```

Figura 4.2: Información mostrada por rabin2 de un binario iOS

Permisos

Uno de los aspectos en los que tanto Apple como Google han puesto mucho hincapié es en los permisos que tienen las aplicaciones dentro del sistema. Los desarrolladores usan permisos para solicitar al sistema acceso a determinadas operaciones como, por ejemplo, utilizar el micrófono o escribir en la memoria del dispositivo.

En el caso concreto de iOS, los desarrolladores deben argumentar en el `Info.plist` (fichero que contiene la configuración de una aplicación iOS) el motivo concreto por el que se está solicitando al usuario cierto permiso. Además, a partir de la versión 6 de iOS los usuarios pueden gestionar de forma individual cada permiso como se puede ver en la Figura 4.3.

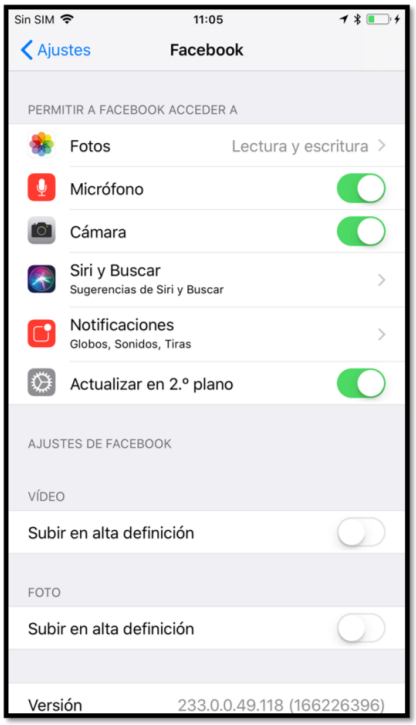


Figura 4.3: Ejemplo de gestión de permisos de la app de Facebook

La gestión de los permisos es fundamental en lo que a ciberseguridad se refiere. Una aplicación maliciosa que envíe mensajes de texto premium necesitará acceso al envío de SMS en el dispositivo, mientras que una aplicación que se encargue de robar datos necesitará acceso a la galería de imágenes, al micrófono y conexión a Internet. En la Tabla 4.1, se puede apreciar la lista completa de permisos que iOS utiliza y las claves definidas en el fichero `Info.plist`.

Para obtener los permisos de la aplicación se ha utilizado la información contenida en el fichero `Info.plist`. En la Figura 4.4 se puede apreciar cómo se definen los permisos dentro del fichero `Info.plist`.

```
<key>NSMotionUsageDescription</key>
<string>Used to control your tempo for Spotify Running.</string>
<key>NSBluetoothPeripheralUsageDescription</key>
<string>This lets you use Spotify with external accessories.</string>
<key>NSAppleMusicUsageDescription</key>
<string>This will let you play your imported music on Spotify.</string>
  <key>NSMicrophoneUsageDescription</key>
<string>This lets you control Spotify using your voice.</string>
```

Figura 4.4: Definición de permisos en un fichero `Info.plist`

Conexiones

La mayoría de las aplicaciones que se instalan en dispositivos móviles requieren de conexión a Internet para funcionar correctamente. Las aplicaciones maliciosas no son una excepción. Por lo tanto, analizando las conexiones que una aplicación iOS realiza, se puede determinar si dicha aplicación se está conectando a servidores de baja reputación con el objetivo de exfiltrar información del usuario o cualquier otra actividad perniciosa.

Para obtener las conexiones que realiza una aplicación se ha utilizado la herramienta Strings, que extrae las cadenas de caracteres presentes en el binario, buscando tanto direcciones URL (*Uniform Resource Locators*) como direcciones IP (*Internet Protocol*).

Nombre de las clases

Las aplicaciones iOS se encuentran desarrolladas actualmente en Swift 2. Este lenguaje de programación es orientado a objetos y por lo tanto el código fuente se divide en clases. El nombre de las clases, que se suele trasladar del código fuente al código binario, habitualmente proporciona mucha información acerca del código que contienen y del papel que desarrollan dentro de la aplicación. Esta información puede ser de gran utilidad para un analista de seguridad.

Para obtener el nombre de las clases de una aplicación, se ha utilizado la herramienta otool. Otool es un analizador de binarios disponible tanto para dispositivos iOS con jailbreak como para macOS. Otool entre otras funcionalidades permite obtener información acerca de las clases y los objetos de la aplicación. En la Figura 4.5 se puede ver remarcado en la salida de otool el nombre de una de las clases del binario de Spotify desarrollado para iOS, así como los nombres de los métodos de la clase.

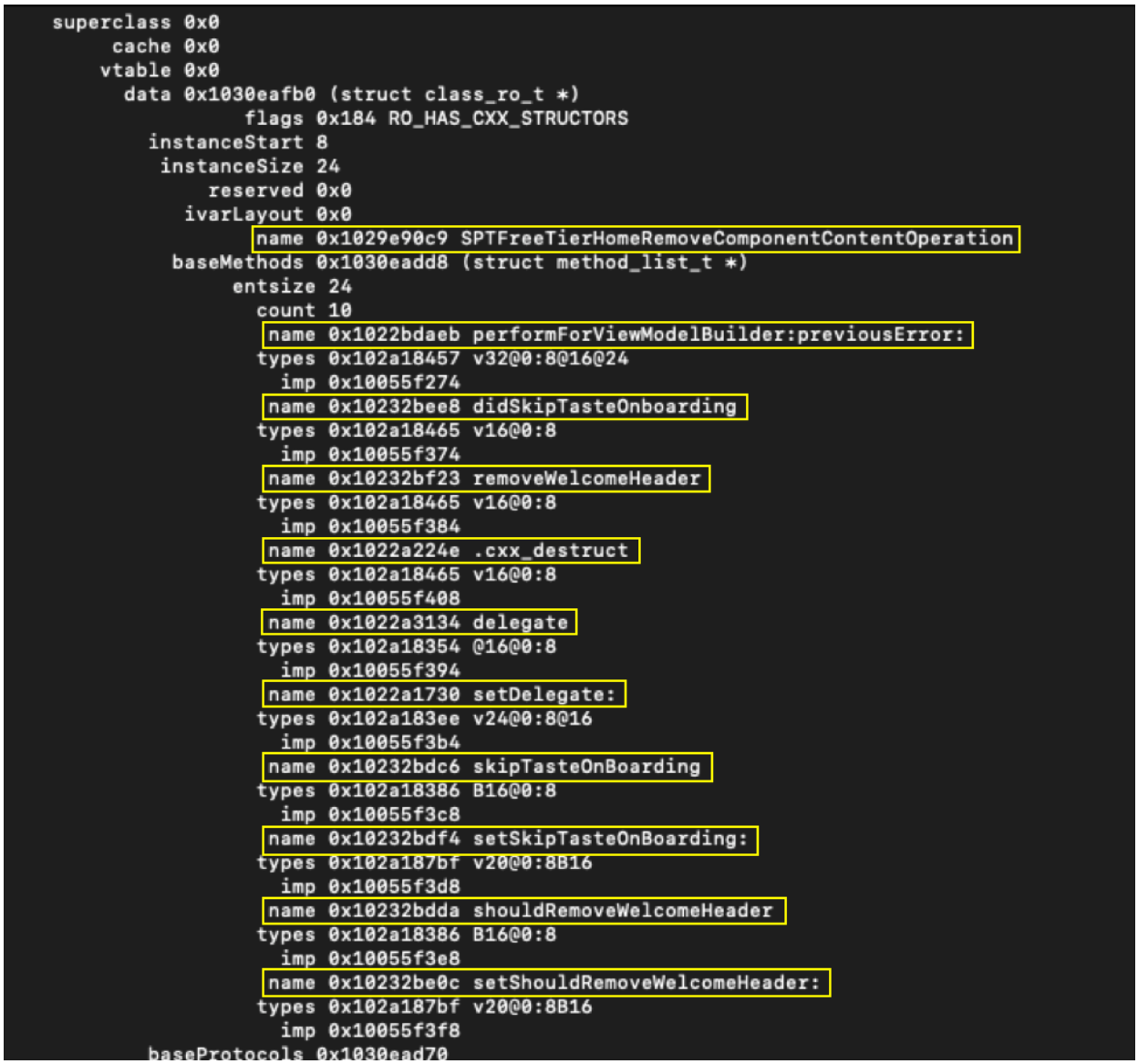


Figura 4.5: Salida de otool sobre un binario iOS descifrado

Directivas del fichero IPA

Además de los permisos, el fichero `Info.plist` contiene más información relevante. A la hora de realizar el análisis de una aplicación, es fundamental conocer la versión de la misma ya que de una versión a otra es posible que se vean añadidas o eliminadas conexiones, nombres de clases o incluso permisos. Como información general de la aplicación, dentro de este apartado se incluye también el nombre del paquete `Bundle Identifier` y la versión mínima de iOS que soporta.

Desde el punto de vista de la ciberseguridad, una de las directivas más importantes que contiene el fichero `info.plist` es `NSAllowsArbitraryLoads`. Si esta direc-

tiva se encuentra activa, significa que el desarrollador deshabilita las restricciones de seguridad de transporte de aplicación y por lo tanto la aplicación puede realizar conexiones sin utilizar un protocolo seguro como HTTPS. Todo desarrollador que desee desactivar las restricciones de seguridad en el transporte deberá indicar a Apple el motivo de esa decisión y finalmente Apple determinará si permite su publicación en el mercado oficial [37].

Por el contrario, si `NSAllowsArbitraryLoads` se encuentra deshabilitada y el desarrollador requiere que la aplicación se conecte a algún dominio que no utilice el protocolo HTTPS, deberá indicarlo en la directiva `NSExceptionDomains`. Si no es así, el sistema bloqueará dicha conexión.

La última directiva importante para el analizador es `LSApplicationQueriesSchemes`. Esta directiva registra con qué aplicaciones dentro del dispositivo puede interactuar la aplicación en cuestión. Es importante tener en cuenta que muy probablemente una app maliciosa tratará de interactuar con la mayoría de apps instaladas en el sistema con el objetivo de obtener toda la información posible del usuario.

Todas las cadenas

El analizador obtiene todas las cadenas de caracteres del binario. Individualmente estas cadenas no son de gran ayuda para un analista, pero pueden serlo a la hora de conocer qué diferencias existen entre aplicaciones.

Estadísticas

Por último, se muestra información numérica de la aplicación. Los datos proporcionados son: número de permisos, número de conexiones, número de clases y número de cadenas.

4.1.3. Caso de uso: aplicación Spotify

Para poder utilizar el analizador, se necesita introducir tan solo dos parámetros. Por un lado, la carpeta donde se realizarán las operaciones necesarias para obtener el binario y el fichero `Info.plist`, y por otro lado el fichero IPA que va a ser analizado. En la Figura 4.6 se muestra la entrada del analizador para la aplicación de Spotify.

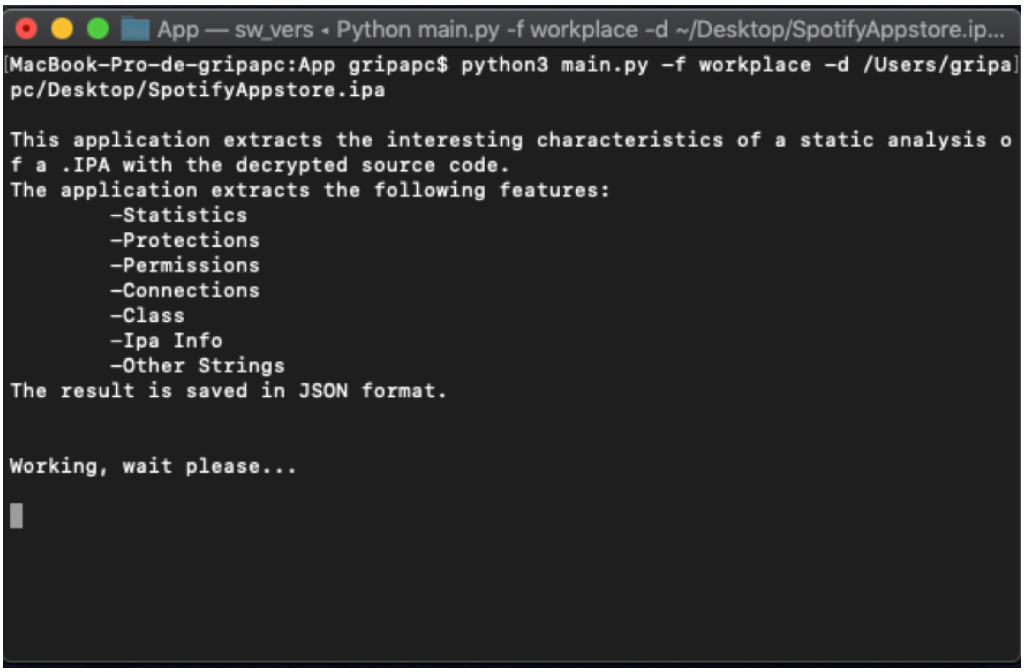


Figura 4.6: Ejecución del sistema de análisis con la aplicación Spotify

Las siguientes figuras muestran la salida del sistema de análisis correspondientes al análisis de la aplicación Spotify en su versión 8.4.96 descargada del mercado oficial de aplicaciones de Apple y cuyo código binario fue descifrado previamente. En la Figura 4.7 se puede apreciar la estructura que devuelve el analizador con todas sus características organizadas en grupos. En la Figura 4.8 se pueden observar las estadísticas de la aplicación analizada. Como se puede ver la aplicación solicita al usuario 7 permisos, realiza 139 conexiones diferentes, implementa 6164 clases (entre las que se incluyen las librerías externas que la aplicación utiliza) y por último, se puede ver que el binario contiene 154767 cadenas diferentes.

Figura 4.7: Estructura del analizador

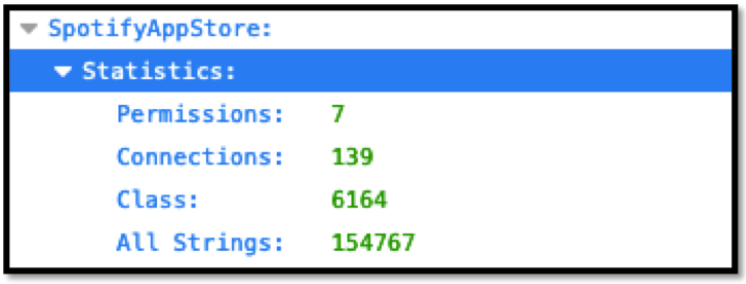


Figura 4.8: Estadísticas de la aplicación Spotify analizada

En la Figura 4.9 se pueden ver las diferentes protecciones que tiene la aplicación. En este caso, se puede apreciar como `crypto` se encuentra al valor `false` ya que se descifró el código compilado de la aplicación antes de realizar el análisis. También es importante destacar que la protección `canary` se encuentra activada.

En la Figura 4.10 se pueden observar los diferentes permisos que la aplicación solicita al sistema. Como se puede observar, esta versión de Spotify solicita permisos para acceder a la cámara y a la galería del dispositivo para añadir imágenes. Spotify también solicita al usuario poder utilizar el *Bluetooth* y el micrófono del dispositivo, así como utilizar AppleMusic (servicio de música en *streaming* de Apple).

Figura 4.9: Protecciones de la aplicación Spotify analizada

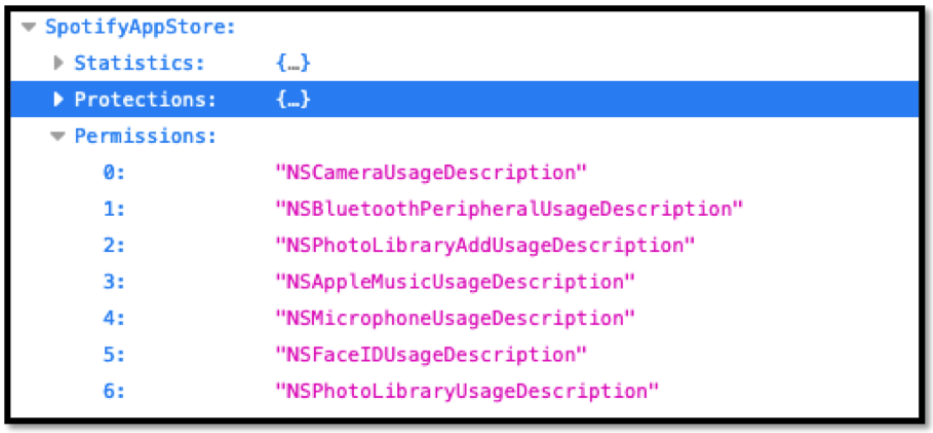


Figura 4.10: Permisos que solicita al sistema la aplicación Spotify analizada

La Figura 4.11 muestra algunas de las direcciones a las que la aplicación se conecta. Como se puede observar, la mayoría de las URLs a las que la aplicación se conecta pertenecen a Spotify. Sin embargo, destaca que la aplicación también establece conexión con Crashlytics. Crashlytics es un producto propiedad de Google que registra fallos en tiempo real. En la imagen también se aprecia que la aplicación se conecta a la página web songkick.com. Esta página proporciona a Spotify información acerca de giras y conciertos de artistas. Como se puede comprobar, la información de las conexiones que una aplicación realiza son de gran utilidad en el análisis de aplicaciones.

Por otro lado, en la Figura 4.12 se pueden apreciar algunos de los nombres de las clases que la aplicación implementa. Como se puede ver, todas las clases comienzan por `SPT` que es la abreviatura usada por Spotify internamente. El propio nombre de la clase proporciona mucha información acerca de la funcionalidad que realiza. Por ejemplo, la clase `SPTAccesoryConnection` está involucrada en las conexiones que realiza la aplicación, mientras que la clase `SPTAccesoryPremiunInfoTableViewCell` muestra información en forma de tabla acerca de la cuenta premium del usuario.

Figura 4.11: Direcciones a la que se conecta la aplicación Spotify analizada

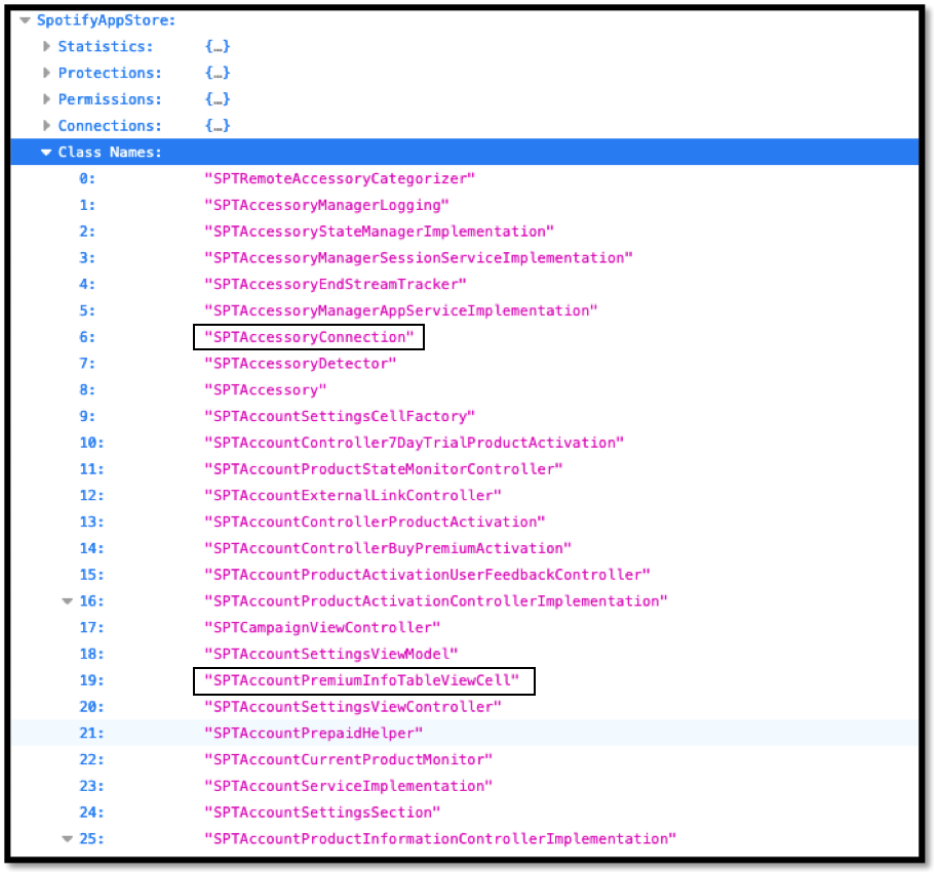


Figura 4.12: Nombres de las clases que la aplicación Spotify analizada tiene en su código fuente

La Figura 4.13 muestra la información del archivo IPA descrita anteriormente. La versión de la aplicación Spotify analizada es la 8.4.96 y la versión mínima de iOS en la que se puede ejecutar es la 10. La directiva `Allow Arbitrary Loads` se encuentra a `false` y por lo tanto los dominios a los que la aplicación se conecta sin un cifrado seguro se encuentran detallados en `Network Exception Domains`. Por otro lado, en la directiva `Application Queries Schemes` se pueden ver las aplicaciones o APIs con las que Spotify interactúa en el dispositivo. En este caso se puede observar Google Maps, Twitter y Facebook.

Figura 4.13: Información del archivo IPA de la aplicación Spotify analizada

4.2. Sistema de comparación

El sistema de comparación se nutre del sistema de análisis descrito anteriormente. Cuando un analista obtiene el fichero IPA descifrado de una aplicación del App Store y sus diferentes copias obtenidas a través de mercados alternativos, puede usar el comparador para obtener las características de ambas aplicaciones y mostrar las diferencias entre ellas.

El objetivo de este comparador es mostrar al analista de forma rápida y automática las diferencias entre las aplicaciones. Obtener estas diferencias pueden ayudar a un analista a determinar si una muestra obtenida de mercados alternativos ha sido modificada.

Para utilizar el comparador, se necesita introducir un mínimo de tres parámetros. En primer lugar, la carpeta donde se realizarán las operaciones necesarias para obtener los binarios y los ficheros `Info.plist` de las aplicaciones comparadas. En segundo lugar, el fichero IPA de la aplicación original que servirá como referencia en la comparación. Por último, como mínimo hay que proporcionar al comparador un tercer fichero IPA que será comparado con la aplicación original. El comparador admite múltiples archivos IPA para ser comparados con la aplicación original en una misma ejecución y por lo tanto el número de parámetros puede ser superior a tres. En la Figura 4.14 se muestra la entrada utilizada para comparar la aplicación de Spotify descargada del App Store (referencia) y su homónima descargada del mercado alternativo AppCake (aplicación a comparar).

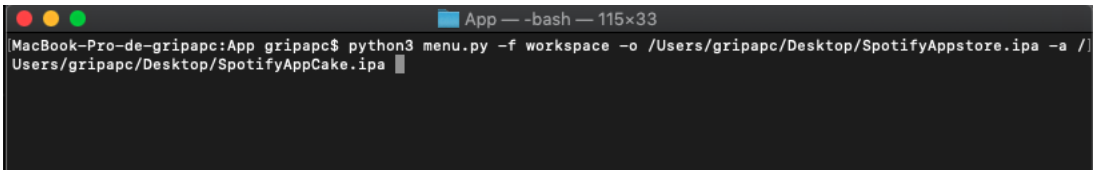


Figura 4.14: Entrada de la comparación de la Aplicación Spotify descargada de AppCake respecto a su homónima del App Store

En el título de la comparativa se puede observar los nombres de las aplicaciones analizadas. Obsérvese que a estos nombres se les ha añadido el sufijo "_X" en el caso de la aplicación original y el sufijo "_Y" en el caso de la aplicación alternativa. Hay que tener en cuenta que siempre se realiza la comparativa de Y respecto a X.

El comparador devuelve en varios apartados las diferencias que encuentra entre las aplicaciones. Los apartados en los que las características sean iguales se mantendrán vacíos. A modo de ejemplo ilustrativo, se ha realizado una comparación entre la aplicación de Spotify descargada del App Store y su homónima descargada de AppCake. Como se puede ver en la Figura 4.15, ambas tienen las mismas protecciones y por lo tanto, el apartado se encuentra vacío.

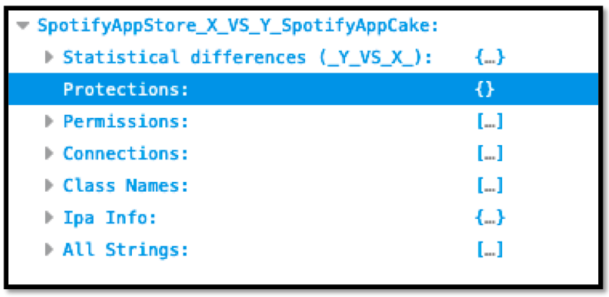


Figura 4.15: Salida de la comparación entre la app Spotify descargada del App Store y de AppCake

En lo que a "Permissions", "Connections", "Class Names" y "All Strings" se refiere, toda la información que se muestra en el reporte corresponde con las características adicionales que la aplicación alternativa tiene respecto de la original. Por ejemplo, en la Figura 4.16 se puede ver el apartado de permisos. En este caso, solo hay un único permiso que es `NSMotionUsageDescription`, que significa que la aplicación descargada de AppCake solicita al usuario permiso para utilizar el acelerómetro, mientras que la app obtenida del App Store no lo hace.

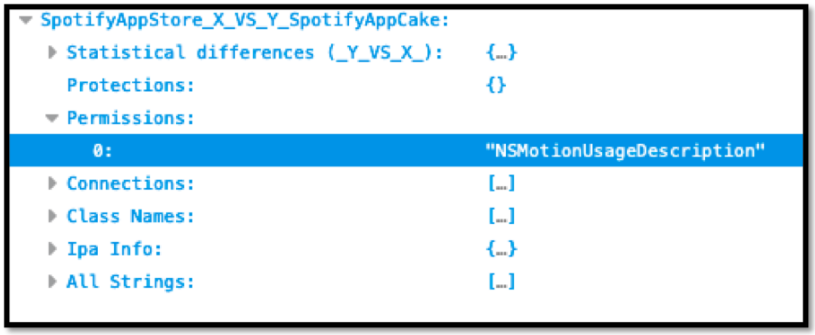


Figura 4.16: Apartado de permisos

En "Ipa Info" los apartados correspondientes a `Network Exception Domains` y `Application Queries Schemes` funcionan de forma similar a los descritos anteriormente. En el resto de los apartados se especifican los datos tanto de la aplicación original como de la alternativa. Se ha tomado esta decisión ya que se considera importante que el analista conozca, por ejemplo, qué versión de la aplicación es la original y qué versión la alternativa. En el caso de que el campo en cuestión sea el mismo en ambas aplicaciones, solo se mostrará una vez. En el caso de no coincidir, se especificará a cuál de las dos aplicaciones pertenece usando los sufijos explicados anteriormente. En la Figura 4.17 se puede observar el apartado "Ipa Info" de la comparativa entre Spotify descargado de App Store y la versión descargada de AppCake
.

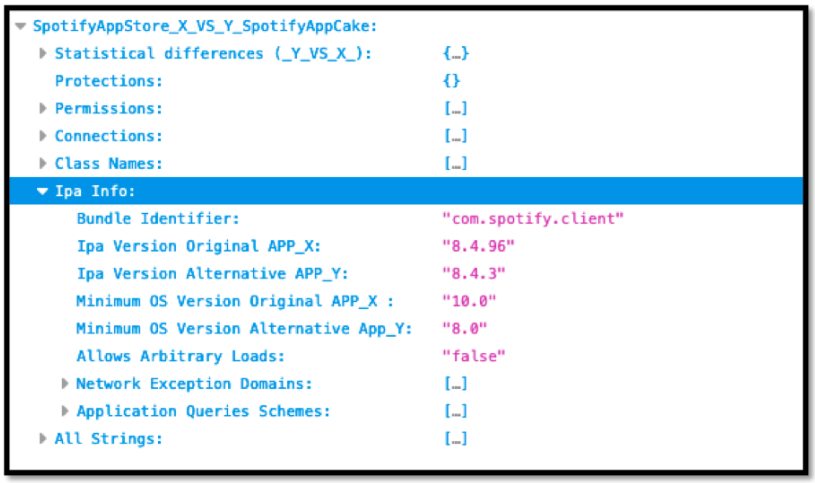


Figura 4.17: Información de "Ipa Info" en la comparativa entre Spotify procedente de App Store y de AppCake

Para finalizar, en el campo "Statistical differences (_Y_VS_X_)" se muestra la diferencia numérica que hay entre la aplicación original y la app alternativa. Por

ejemplo, en el caso de que la aplicación original tenga más permisos, conexiones, clases o cadenas que la aplicación alternativa se verá reflejado con números negativos. En la Figura 4.18 se puede ver que la aplicación de Spotify descargada de App Store tiene 1 permiso más, realiza 3 conexiones más, tiene 2976 clases menos y contiene 17515 cadenas más que la app procedente de AppCake.

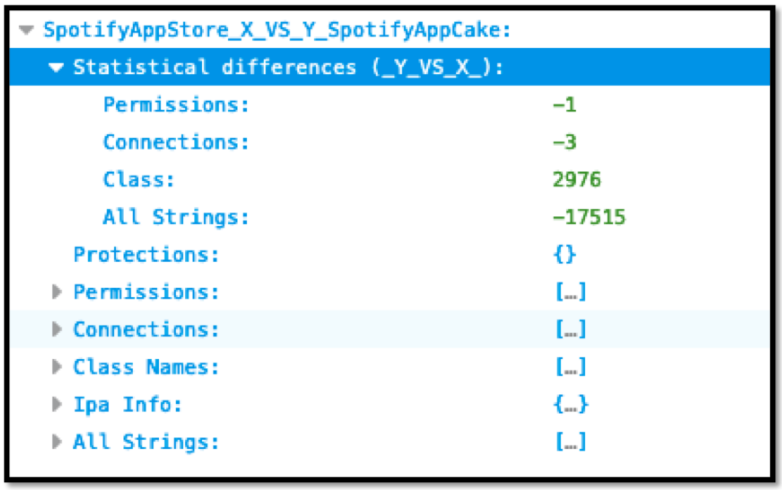


Figura 4.18: Diferencia de conexiones entre la aplicación Spotify descargada de App Store y de AppCake

Comparativa gráfica

Con el objetivo de que un analista pueda ver qué diferencias en términos porcentuales tienen las aplicaciones comparadas, el sistema devuelve una gráfica mostrando en qué porcentaje ha variado la aplicación alternativa respecto de la original en cada una de las características analizadas. Hay que tener en cuenta que la aplicación original siempre tendrá sus valores a 0. En la Figura 4.19 se puede apreciar la gráfica comparativa entre la aplicación Spotify descargada del AppStore y su homónima de AppCake, según los resultados comentados anteriormente.

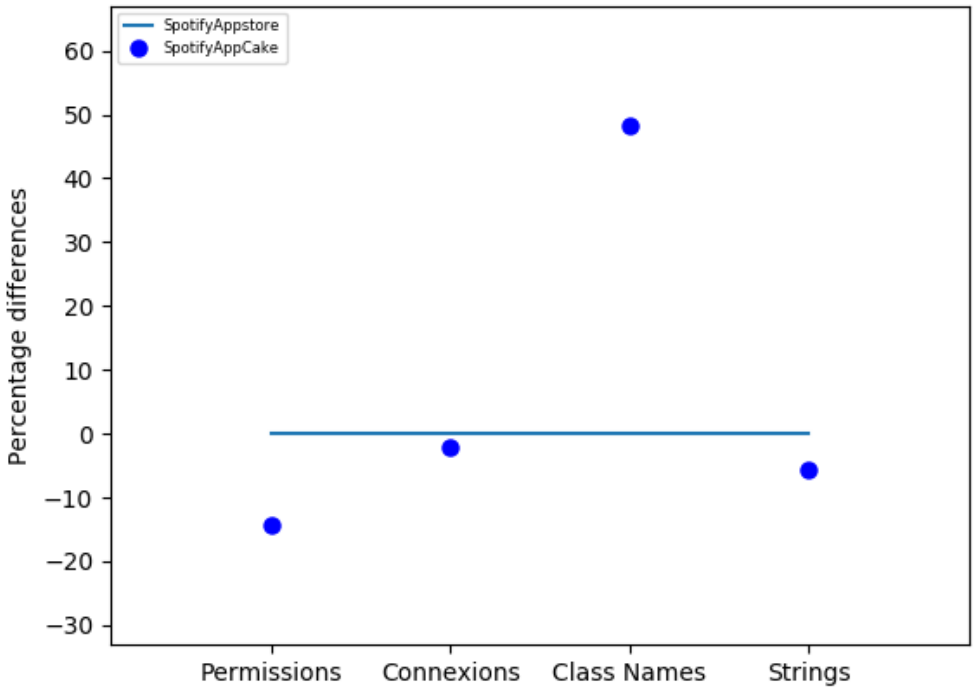


Figura 4.19: Gráfica comparativa entre la aplicación Spotify descargada de App Store y de AppCake

Tabla 4.1: Lista de permisos que iOS utiliza

Clave	**Uso**
`NSPhotoLibraryAddUsageDescription`	La aplicación añade fotografías a la galería de imágenes del usuario
`NSPhotoLibraryUsageDescription`	La aplicación accede a la galería de imágenes del usuario
`NSCameraUsageDescription`	La aplicación utiliza la cámara del dispositivo
`NSLocationAlwaysUsageDescription`	La aplicación utiliza el servicio de localización del dispositivo en todo momento
`NSLocationWhenInUseUsageDescription`	La aplicación utiliza el servicio de localización del dispositivo solo cuando se encuentra en ejecución
`NSContactsUsageDescription`	La aplicación utiliza la agenda de contactos del usuario
`NSCalendarsUsageDescription`	La aplicación utiliza o modifica la información del calendario
`NSRemindersUsageDescription`	La aplicación crea recordatorios en la app "Recordatorios"
`NSHealthShareUsageDescription`	La aplicación utiliza información de la app "Salud"
`NSHealthUpdateUsageDescription`	La aplicación suministra información sanitaria a la app "Salud"
`NFCReaderUsageDescription`	La aplicación utiliza el lector NFC del dispositivo
`NSBluetoothPeripheralUsageDescription`	La aplicación interactúa con dispositivos Bluetooth
`NSMicrophoneUsageDescription`	La aplicación utiliza el micrófono del dispositivo
`NSSiriUsageDescription`	La aplicación proporciona información a "SiriKit"
`NSSpeechRecognitionUsageDescription`	La aplicación utiliza reconocimiento por voz
`NSMotionUsageDescription`	La aplicación utiliza el hardware de seguimiento de movimiento
`NSAppleMusicUsageDescription`	La aplicación se integra con "Apple Music"
`NSFaceIDUsageDescription`	La aplicación utiliza "FaceID"

Capítulo 5

Experimentos y validación

En este capítulo se detallan los experimentos realizados y los resultados obtenidos.

Una vez desarrollado el analizador estático y el comparador, la idea inicial del estudio consistía en seleccionar aplicaciones de diferentes mercados alternativos y compararlas con la aplicación correspondiente obtenida del App Store. Los criterios de selección estaban totalmente determinados por la disponibilidad de aplicaciones en diferentes mercados, ya que es relativamente complejo encontrar copias de la misma aplicación en los mercados alternativos de aplicaciones iOS. Esto es debido a que la mayoría de estos mercados no cuentan con un gran número de aplicaciones disponibles.

Cuando se procedió a la descarga de las aplicaciones se rechazó el estudio previamente diseñado debido a que la gran mayoría de las aplicaciones de los diferentes mercados alternativos se encuentran alojadas en AppEven, y el resto de mercados alternativos se conectan a AppEven para descargar las aplicaciones. Por lo tanto, las aplicaciones que se trataría de comparar serían siempre las mismas. Además, en el espacio temporal en el que se realizó el estudio, AppEven no se encontraba disponible.

Debido a los problemas mencionados anteriormente, se decidió modificar el estudio, de forma que se han obtenido un total de 14 aplicaciones de un único mercado, AppCake. AppCake dispone de sus propias aplicaciones y no las descarga de AppEven, contiene un gran número de aplicaciones y es uno de los mercados alternativos más populares en la actualidad [38, 39]. El objetivo del estudio es analizar qué aplicaciones descargadas de AppCake han sido modificadas y qué es lo que se ha sido

modificado. Para realizar el estudio se ha utilizado el analizador estático y el comparador previamente implementados en este trabajo.

De las 14 aplicaciones seleccionadas, 7 son aplicaciones descargadas previo pago en el App Store mientras que de App Cake fueron obtenidas de forma gratuita. Las 7 aplicaciones restantes han sido obtenidas de forma gratuita tanto en el App Store como en App Cake. En este estudio no se ha tenido en cuenta la temática de las aplicaciones. Cabe destacar la dificultad de encontrar aplicaciones en mercados alternativos que ya se encuentran disponibles gratuitamente en el App Store.

En la Tabla 5.1 se pueden ver las aplicaciones gratuitas del AppStore estudiadas. Se muestra el mercado del que han sido descargadas, si incorporaban cifrado alternativo (explicado más adelante) y por último la versión de las mismas. Con el mismo formato, en la Tabla 5.2 se pueden ver las aplicaciones que fueron descargadas previo pago del App Store.

Tabla 5.1: Aplicaciones gratuitas utilizadas en el estudio

Nombre de la aplicación	**Mercado**	**Cifrado alternativo**	**Versión**
Discord	App Store	No	3.1.0
Discord	AppCake	Sí	2.4.0
HelixDump	App Store	No	2.5.1
HelixDump	AppCake	No	1.1.2
Pinterest	App Store	No	7.28.1
Pinterest	AppCake	No	6.59
Spotify	App Store	No	8.5.19
Spotify	AppCake	Sí	8.5.19
Tinder	App Store	No	10.18.0
Tinder	AppCake	Sí	10.17.0
Twitch	App Store	No	7.13
Twitch	AppCake	Sí	7.1
Vlc	App Store	No	3.1.9
Vlc	AppCake	No	3.1.9

En la Tabla 5.2 destaca que la versión de las aplicaciones de pago es la misma tanto en el mercado alternativo como en el original. Este hecho supone que las aplicaciones de pago se actualizan de forma regular en AppCake. En el caso de las aplicaciones gratuitas, esto no ocurre. Exceptuando un caso, todas las demás aplicaciones descargadas de AppCake se encuentran obsoletas respecto a la versión

Tabla 5.2: Aplicaciones de pago utilizadas en el estudio

Nombre de la aplicación	Mercado	Cifrado alternativo	Versión
2 Players Quiz Pro	App Store	No	1.9.6
2 Players Quiz Pro	AppCake	Sí	1.9.6
Ebook Search	App Store	No	3.11
Ebook Search	AppCake	Sí	3.11
Good Reader	App Store	No	5.1.5
Good Reader	AppCake	Sí	5.1.5
Infinity Flight	App Store	No	19.03.2
Infinity Flight	AppCake	No	19.03.2
Plague	App Store	No	1.16.8
Plague	AppCake	Sí	1.16.8
Pou	App Store	No	1.4.77
Pou	AppCake	No	1.4.77
Rebel	App Store	No	1.3.3
Rebel	AppCake	Sí	1.3.3

del App Store. Tan solo en el caso de Spotify la versión del App Store coincide con la de AppCake.

Cabe destacar que de las 14 aplicaciones descargadas de AppCake 9 incorporaban cifrado. Este dato es muy relevante en la investigación, ya que quiere decir que estas apps cifradas no han podido ser analizadas con el sistema desarrollado. Comparando las cadenas resultantes de la misma aplicación cifrada procedente de App Store y de AppCake se puede concluir que el cifrado de estas aplicaciones no es el mismo que Apple utiliza en su mercado oficial. Mientras que en el cifrado de Apple no se puede apreciar ningún texto legible, en el caso de la aplicación cifrada de AppCake sí se observa texto legible (mostrados en la Figura 5.1 y la Figura 5.2, respectivamente). Esto hace sospechar que estas aplicaciones puedan contener algún tipo de carga maliciosa y el cifrado alternativo haya sido utilizado para ocultar dicha carga maliciosa. Para verificar estas sospechas sería necesario realizar un análisis dinámico de la aplicación.

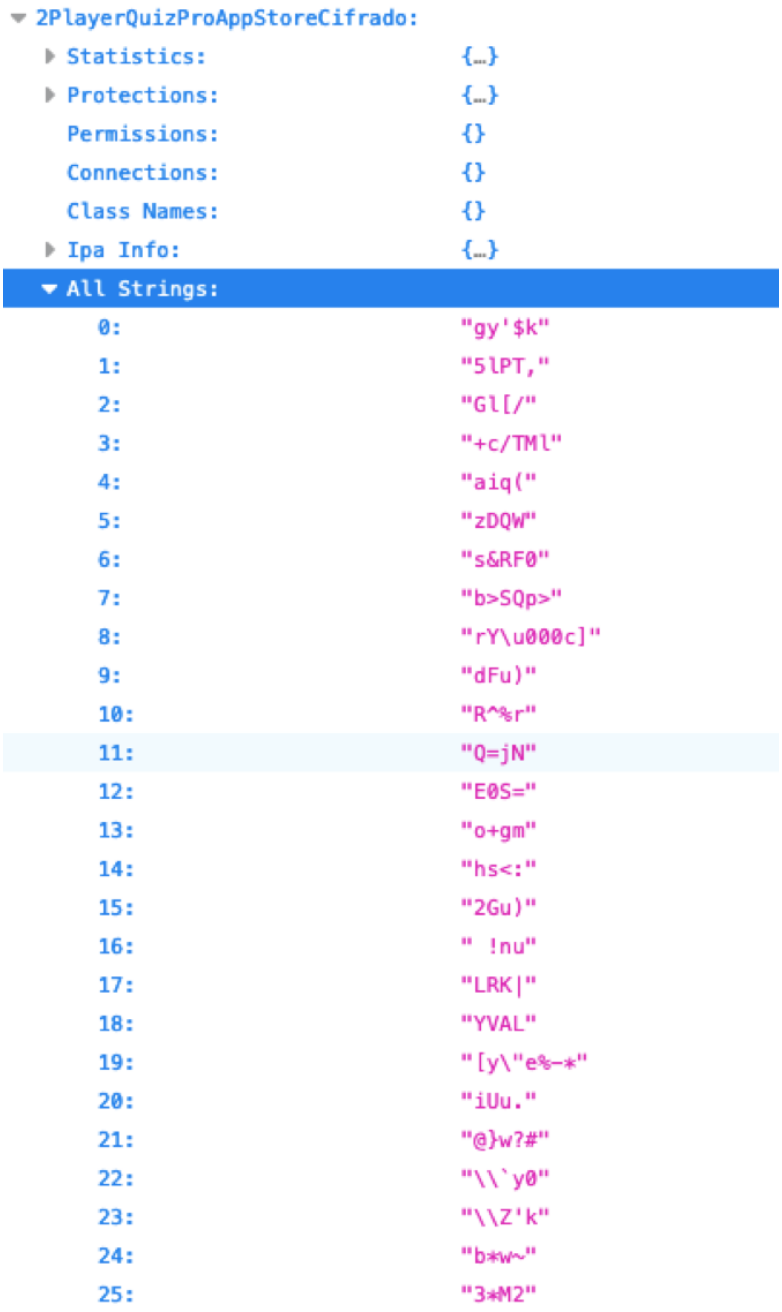


Figura 5.1: Cadenas cifradas provenientes del App Store

Figura 5.2: Cadenas cifradas provenientes AppCake

Respecto a las 5 aplicaciones sin cifrado alternativo, en la Tabla 5.3 se pueden apreciar las diferencias entre ambas en términos numéricos. Como se puede observar, Infinity Flight, Vlc y Helix Dump tienen exactamente las mismas cifras tanto en su versión obtenida de la App Store como la obtenida de App Cake y por lo tanto se puede afirmar que no han sufrido modificaciones. En cuanto a la app de Pou, esta cuenta con los mismos permisos y el mismo número de conexiones, pero cuenta con una conexión diferente a la de la AppStore. Analizando en detalle las cadenas, se puede afirmar que la versión descargada de AppCake ha sido modificada para añadir publicidad. En la Figura 5.4 se puede ver una cadena que hace referencia a un script de la plataforma de publicidad de Google. Por último y después de analizar las diferencias de la aplicación de Pinterest, se ha concluido que estas diferencias se

deben a la disparidad de versiones entre la app perteneciente al AppStore y la app relativa a AppCake.

Tabla 5.3: Características numéricas de las aplicaciones estudiadas

Nombre	Permisos	Conexiones	Clases	Strings
Pinterest (App Store)	8	88	3827	110133
Pinterest (AppCake)	8	113	7554	110660
Vlc (App Store)	3	97	3210	118588
Vlc (AppCake)	3	97	3210	118588
Infinity Flight (App Store)	2	2	187	19683
Infinity Flight (AppCake)	2	2	187	19683
Helix Dump (App Store)	5	138	3951	127312
Helix Dump (AppCake)	3	138	3951	127312
Pou (App Store)	7	27	2722	21395
Pou (AppCake)	7	27	2722	22759

Figura 5.3: Cadena de conexión diferente encontrada en la app Pou procedente de AppCake

Figura 5.4: Referencia a un JavaScript de GoogleAds

Capítulo 6

Trabajo relacionado

En este capítulo se va a llevar a cabo una revisión de los trabajos relacionados con este proyecto.

Existen trabajos que utilizan técnicas de análisis tanto estático como dinámico de aplicaciones iOS. El análisis estático es el análisis que se realiza al código de una aplicación que no se encuentra en ejecución. Este tipo de análisis permite a un analista conocer todos los posibles caminos que puede tomar la aplicación durante su ejecución. El análisis dinámico de código, por el contrario, es el análisis que se realiza en tiempo de ejecución y por lo tanto solo analiza el camino que el código toma en esa ejecución concreta.

En el año 2011 M. Egele et al. presentan PiOS, un sistema de detección de fugas de información para aplicaciones iOS [40]. PiOS realiza un análisis estático para detectar flujos de datos iOS. Los autores analizaron cerca de 1400 aplicaciones, detectando que 264 obtenían datos del usuario.

En el año 2012 M. Szydlowski et al. presentan técnicas de análisis dinámico de aplicaciones iOS [41]. En las pruebas realizadas se utilizaron dos enfoques diferentes de análisis dinámico obteniendo dos resultados diferentes. Por un lado, se detectó un 64 % de las llamadas a funciones que la aplicación realizaba cuando el usuario interactúa con la interfaz. Por otro lado, si no se controlaba la interfaz de usuario, tan solo un 16 % de las llamadas a funciones que realizaba la aplicación eran detectadas.

En el año 2016 L. García y R. J. Rodríguez realizaron un estudio de las características de 36 aplicaciones maliciosas dirigidas al sistema operativo iOS [42]. Las muestras estudiadas fueron descubiertas entre los años 2009 y 2015, y se clasificaron en función a diferentes criterios. Los resultados obtenidos ponen de manifiesto el

peligro que conlleva tener un dispositivo con jailbreak, ya que todas las muestras analizadas afectaban a dispositivos con jailbreak y tan solo el 30,5 % afectaban a dispositivos sin jailbreak. Además, este estudio refleja como más de dos tercios del *malware* se distribuye fuera del mercado oficial de aplicaciones, ya que tan solo el 13,9 % de las muestras de *malware* analizadas fueron descargadas del App Store.

En el año 2019 Z. Deng et al. proponen iRis [43], una herramienta que realiza un análisis estático y dinámico a aplicaciones iOS con el objetivo de detectar qué llamadas a APIs (*Application Programming Interface*) privadas realiza la aplicación que está siendo analizada. iRiS se probó en más de 2000 aplicaciones iOS descargadas del App Store, y en 146 se descubrieron llamadas a APIs privadas que habían pasado inadvertidas en el veto del mercado oficial de aplicaciones iOS. iRiS utiliza análisis estático únicamente para detectar llamadas a APIs privadas, mientras que el analizador estático desarrollado en este proyecto obtiene más características como conexiones, protecciones y permisos, entre otras.

De los trabajos realizados hasta la fecha se han podido obtener varias conclusiones que sirvieron como punto de partida para este trabajo. El análisis de aplicaciones iOS necesita de análisis estático, ya que si solo se utilizan técnicas de análisis dinámico el porcentaje de código analizado es inferior al esperado. Por otro lado, la publicación [42] pone de manifiesto que la gran mayoría de las muestras de *malware* analizadas se encuentran alojadas en mercados alternativos y por lo tanto existe la necesidad de estudiar estos mercados y las aplicaciones que contienen.

Capítulo 7

Conclusiones y trabajos futuros

Este apartado presenta las conclusiones obtenidas a lo largo del proyecto, las aportaciones realizadas y los trabajos futuros que se pueden llevar a cabo con el objetivo de mejorar y aportar más valor al proyecto.

Toda la historia de iOS como sistema operativo va ligada a las numerosas restricciones que Apple ha impuesto tanto a usuarios como a desarrolladores e investigadores. Con en el objetivo de aportar personalización y funcionalidad a los usuarios nació el jailbreak y con él, los mercados alternativos de aplicaciones iOS. El surgimiento de mercados alternativos que no implementan las restricciones y las medidas de seguridad del App Store ha sido una gran oportunidad para los cibercriminales para introducir código malicioso en sistemas iOS. En este trabajo se han llevado a cabo varias aportaciones en lo que al estudio de mercados alternativos de aplicaciones iOS y al análisis estático de apps se refiere.

En la primera fase del proyecto se ha llevado a cabo un estudio de los diferentes mercados alternativos de aplicaciones existentes, analizando 23 mercados alternativos diferentes que ofrecían aplicaciones iOS. Estos mercados han sido clasificados en función de si son gratuitos, si ofrecen apps originalmente de pago de forma gratuita, si requieren jailbreak y por último si el usuario necesita registrarse para poder utilizar el mercado. De los 23 mercados analizados, tan solo uno de ellos no ofrece réplicas de aplicaciones disponibles en el App Store. Además, 13 de los 23 mercados no necesitan jailbreak lo que significa que están dedicados en exclusiva a ofrecer aplicaciones de forma gratuita que en el mercado oficial son de pago.

La segunda fase del proyecto ha consistido en la investigación y desarrollo de un sistema de análisis estático capaz de extraer características importantes desde

el punto de vista de la ciberseguridad de aplicaciones iOS. Algunas de estas características son las conexiones que realiza la aplicación o el número de clases que implementa entre otras. Esta herramienta se considera un paso más en el análisis de aplicaciones ya que facilita y automatiza en gran medida la tarea de un analista de seguridad de aplicaciones iOS. Un punto importante es la compatibilidad de la herramienta, ya que se ha desarrollado un sistema capaz de funcionar en un dispositivo iOS con jailbreak y en un ordenador con macOS instalado.

Con el objetivo de facilitar el análisis de aplicaciones, se ha implementado un comparador de características que utiliza el sistema de análisis desarrollado previamente. Este comparador se desarrolló con el objetivo de mostrar las diferencias entre las aplicaciones analizadas, permitiendo a un analista poder ver en qué difiere una aplicación descargada de un mercado alternativo respecto de la descargada del mercado oficial. Este comparador tiene una doble funcionalidad: además de analizar las diferencias entre aplicaciones, también permite analizar los cambios que ha tenido una misma aplicación en sus diferentes versiones.

El sistema desarrollado se ha evaluado con 14 aplicaciones descargadas del mercado alternativo AppCake, algunas gratuitas y otras de pago en el App Store. Los resultados muestran que tan solo 4 no habían sido modificadas. De las 10 aplicaciones restantes, 9 contenían una capa de cifrado diferente a la utilizada por Apple y no se pudieron analizar. Probablemente esta capa de cifrado se ha incorporado con el objetivo de ocultar modificaciones. La aplicación restante es el conocido juego Pou. Gracias al comparador rápidamente se determinó que esta aplicación incorporaba un sistema de cobro por publicidad no existente en la aplicación original.

Como trabajo futuro, se considera importante continuar mejorando la herramienta desarrollada en este proyecto. Para ello, a continuación se exponen algunas posibles mejoras:

- En lo que a análisis estático se refiere, se pretende añadir además del nombre de las clases, los métodos que implementan dichas clases.
- Se pretende introducir la posibilidad de realizar análisis dinámico de aplicaciones iOS, pudiendo comparar los resultados obtenidos entre varias apps de diferentes mercados.
- Con el objetivo de tener una visión más global de los diferentes mercados, es importante continuar analizando aplicaciones de los mismos y ampliar el número de mercados alternativos analizados.

- Se considera también importante tratar de analizar y descifrar la capa de seguridad añadida por AppCake en alguna de las aplicaciones que almacena.
- Se pretender ampliar aún más la compatibilidad de las herramientas desarrolladas, dándole soporte para sistemas operativos basados en Linux.

Bibliografía

[1] Empresas más grandes del mundo 2019. (https://economipedia.com/ranking/empresas-mas-grandes-del-mundo-2019.html)

[2] ¿Cuánto ha vendido cada generación de iPhone? (https://www.revistagq.com/noticias/tecnologia/articulos/cuantos-iphone-vendido-modelo/23617)

[3] Stahl, F., Ströher, J.: Security testing guidelines for mobile apps. AppSec Research EU, The OWASP Foundation [PDF File]. Retrieved October **18** (2013) 2015

[4] Secure Enclave: Qué es y cómo funciona esta capa de seguridad de Apple. (https://www.redeszone.net/2018/01/29/como-funciona-secure-enclave/)

[5] Thiel, D.: IOS Application Security: The Definitive Guide for Hackers and Developers. No Starch Press (2016)

[6] Miller, C., Blazakis, D., DaiZovi, D., Esser, S., Iozzo, V., Weinmann, R.P.: iOS Hacker's Handbook. 1st edn. Wiley Publishing (2012)

[7] Llegan las apps pirata a los iPhone por un "fallo" en los certificados de Apple. (https://www.adslzone.net/2019/02/14/apps-pirata-iphone-fallo-certificados/)

[8] Revisión de apps en la App Store de Apple. https://www.applesfera.com/app-store-1/cnbc-detalla-como-funciona-proceso-revision-apps-app-store-apple (29-07-2019)

[9] Xcode 7 permite instalar aplicaciones en iOS sin pasar por App Store. (https://www.fayerwayer.com/2015/06/xcode-7-permite-instalar-aplicaciones-en-ios-sin-pasar-por-app-store/)

[10] Understanding the Code Signature. (https://developer.apple.com/library/archive/documentation/Security/Conceptual/CodeSigningGuide/AboutCS/AboutCS.html)

[11] Mercado alternativo Tuto App. (https://www.tutuapp.vip)

[12] Mercado alternativo vShare. (https://vsharedownload.org/)

[13] Mercado alternativo Zestia Extender. (https://zestia.lmdinteractive.com/)

[14] Mercado alternativo Apps4iPhone. (https://apps4iphone.net/)

[15] Mercado alternativo AppValley. (https://appvalleyapp.com/)

[16] Mercado alternativo Tweakbox. (https://tweak-box.com/)

[17] Mercado alternativo Appcake. (https://iphonecake.com/)

[18] Mercado alternativo 51ipa. (https://www.51ipa.com/)

[19] Mercado alternativo 25 app. (http://pro.25pp.com/ppios)

[20] (Mercado alternativo Tongbu, howpublished = http://www.tongbu.com/, u...)

[21] Mercado alternativo App 704. (https://www.app704.com/)

[22] Mercado alternativo Mojo installer. (https://www.mojoinstaller.co/)

[23] Mercado alternativo Ipa Box. (https://ipabox.store/)

[24] Mercado alternativo App China. (http://www.appchina.com/)

[25] Mercado alternativo AppEven. (https://app-even.com/)

[26] Mercado alternativo HipStore. Enlace de descarga a través de Cydia. (https://cydia-app.com/hipstore/)

[27] Mercado alternativo TopStore. (https://topstorevipapp.com/)

[28] Mercado alternativo iNoJB. (https://inojb.net/)

[29] Mercado alternativo Emus4U. (https://emus4udownload.org/)

[30] Mercado alternativo IpaStore. (https://ipastore.me/)

[31] Mercado alternativo Lee890. (http://m.le890.com/)

[32] Mercado alternativo Asterix installer. (https://www.asterixinstaller.com/)

[33] Mercado alternativo Taigone Tikiri. (`https://taigone.com/`)

[34] JSON. (`https://www.json.org/`)

[35] Wagle, P., Cowan, C., et al.: Stackguard: Simple stack smash protection for gcc. In: Proceedings of the GCC Developers Summit. (2003) 243–255

[36] Radare 2. (`https://radare.gitbooks.io/radare2book/first_steps/overview.html`)

[37] NSAllowsArbitraryLoads. (`https://developer.apple.com/documentation/bundleresources/information{_}property{_}list/nsapptransportsecurity/nsallowsarbitraryloads`)

[38] Las 12 mejores alternativas a TutuApp para Android y iPhone. (`https://www.malavida.com/es/listas/las-mejores-alternativas-a-tutuapp-para-android-y-iphone-006543{#}gref`)

[39] Algunas tiendas alternativas a la App Store oficial de Apple. (`https://www.adslzone.net/2015/11/21/algunas-tiendas-alternativas-a-la-app-store-oficial-de-apple/`)

[40] Egele, M., Kruegel, C., Kirda, E., Vigna, G.: PiOS: Detecting Privacy Leaks in iOS Applications. In: Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011. (2011)

[41] Szydlowski, M., Egele, M., Kruegel, C., Vigna, G.: Challenges for Dynamic Analysis of iOS Applications. In Hutchison, D., Kanade, T., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M.Y., Weikum, G., Camenisch, J., Kesdogan, D., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Rangan, C.P., Steffen, B., eds.: International Workshop on Open Problems in Network Security (iNetSec). Volume LNCS-7039 of Open Problems in Network Security., Lucerne, Switzerland, Springer (2011) 65–77 Part 2: Malware Detection.

[42] García, L., Rodríguez, R.J.: A Peek under the Hood of iOS Malware. In: 2016 11th International Conference on Availability, Reliability and Security (ARES), IEEE (2016) 590–598

[43] Deng, Z., Saltaformaggio, B., Zhang, X., Xu, D.: iRiS: Vetting Private API Abuse in iOS Applications. In: Proceedings of the 22Nd ACM SIGSAC Con-

ference on Computer and Communications Security. CCS '15, New York, NY, USA, ACM (2015) 44–56

Anexo A

Realización de tareas

El proyecto ha tenido una duración total de 765 horas, que equivale aproximadamente a 3 horas y 25 minutos diarios desde el mes de enero hasta septiembre. Las diferentes fases de las que consta el proyecto, desglosadas en la Tabla A.1 y cuya representación gráfica se muestra en la Figura A.1, son las siguientes:

Fase 1: Estudio del estado de la cuestión, recopilación de información y adquisición del conocimiento necesario para la elaboración del proyecto.

Fase 2: Investigación de los diferentes jailbreaks y realización de los mismos en los dispositivos destinados al proyecto. El objetivo de esta fase era tener preparado todo el material necesario para la realización del proyecto.

Fase 3: Recopilación de los diferentes mercados alternativos de aplicaciones iOS. Una vez seleccionados los mercados, se elaboraron los criterios de clasificación que se aplicarían a cada uno de los mercados alternativos encontrados.

Fase 4: Investigación de las diferentes características relevantes desde el punto de vista de la ciberseguridad de las aplicaciones desarrolladas para iOS.

Fase 5: Desarrollo de un sistema de análisis estático escrito en Python y orientado a objetos.

Fase 6: Desarrollo del comparador de aplicaciones descargadas de diferentes mercados. El comparador utiliza el analizador desarrollado en la fase anterior y también se encuentra escrito en Python.

Fase 7: Con el objetivo de una mayor compatibilidad del proyecto, se adaptan tanto el analizador como el comparador para que puedan ejecutarse en macOS sin la necesidad de un dispositivo iOS.

Fase 8: Evaluación del sistema. En esta fase se han seleccionado las mismas aplicaciones provenientes de los diferentes mercados alternativos previamente estudiados y se han analizado utilizando las implementaciones realizadas en las fases anteriores.

Fase 9: Elaboración de la memoria de este trabajo de fin de máster.

Tabla A.1: Estimación tareas

Fases	Días	Fecha inicio	Fecha fin
Fase 1	27	2/01/19	29/01/19
Fase 2	5	30/01/19	04/02/19
Fase 3	25	05/02/19	02/03/19
Fase 4	50	03/03/19	22/04/19
Fase 5	55	23/04/19	17/06/19
Fase 6	27	18/06/19	15/07/19
Fase 7	6	16/07/19	22/07/19
Fase 8	10	23/07/19	02/08/19
Fase 9	31	03/08/19	03/09/19

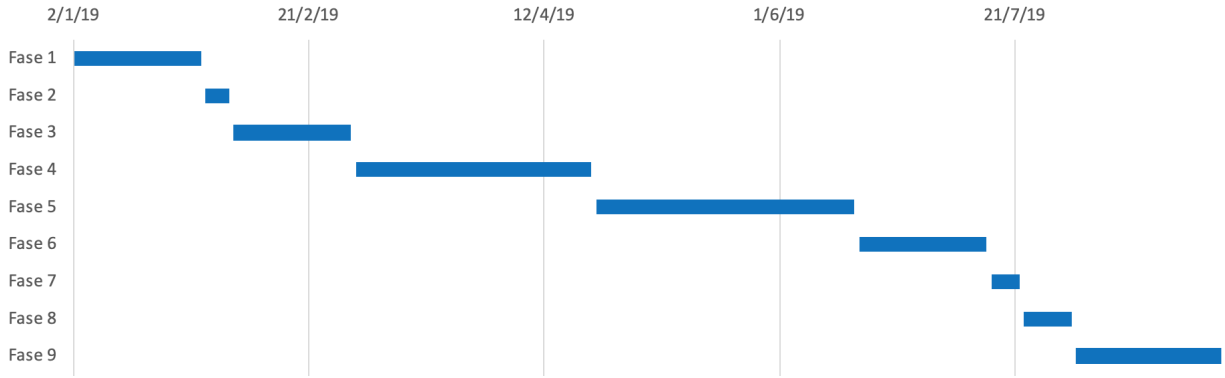


Figura A.1: Diagrama de Gantt estimación duración de las tareas