Análisis y explotación de vulnerabilidades en iOS

Pablo Montes Sierra, Instituto Politécnico Nacional, Octubre de 2018

Abstract - The scope of this Project is based on the analysis and exploitation of public vulnerabilities that affect iOS at kernel level. For the purposes of this project, the version used is iOS 11.

I. Introducción

La información almacenada en los dispositivos móviles es un activo informático que resulta ser muy atractivo para un atacante que tiene el potencial de causar daño a través del robo, destrucción, divulgación, modificación de datos o denegación de servicio, entre otras. En América Latina, durante el 2017, los países donde se registró mayor presencia de malware en iOS fueron: **México** (29%), **Ecuador** (17%) y **Brasil** (14%)[1]. En Agosto de 2016, *Citizenlab* en conjunto con *Lookout Security*, identificaron un ataque altamente sofisticado que permitía explotar vulnerabilidades críticas que afectaba directamente al ***sandbox***[1] y ***kernel***[2] del Sistema Operativo iOS[2], el cual permite obtener privilegios de administrador (*jailbreak*[3]) de forma remota.

Este informe de seguridad documenta, por primera vez, como un malware puede comprometer la seguridad del dispositivo por medio de un mal manejo de memoria en el motor del navegador Safari (*WebKit*), para realizar después un escape de ***sandbox*** que le permite tener acceso al Sistema Operativo con bajos privilegios, esta vulnerabilidad se encuentra registrada bajo el ***CVE-2016-4657.*** El malware busca llegar al ***kernel*** del sistema aprovechándose de una segunda vulnerabilidad (divulgación de memoria, ***CVE-2016-4655***) que le permite conocer dónde se encuentra el ***kernel*** en memoria.

Finalmente, el malware explota una tercera vulnerabilidad por medio de la técnica ***User After Free***[4] que permite sobrescribir la memoria de ***kernel*** para obtener privilegios de administrador, esta vulnerabilidad está documentada con el ***CVE-2016-4656***.

Un atacante con permisos de administrador puede acceder a la información del dispositivo que se lista a continuación:

- **Datos de aplicaciones de mensajería:** WhatsApp, SMS, Messenger.
- **Datos de usuario:** fotos, video, audio, contactos, entre otras.
- **Datos de localización:** Celdas móviles, dirección MAC, GPS.
- **Registro de palabras por teclado**.

Posterior a la explotación, se realiza la extracción de información del dispositivo en modo administrador, también se pueden

[1] Sandbox. Se refiere a la contención de procesos, de una manera que permite la restricción de sus llamadas al sistema. La restricción puede modificarse para permitir o no permitir llamadas al sistema en función de archivos particulares u objetos[3].

[2] Kernel. Es un programa que constituye el núcleo central de un Sistema Operativo[4], Apple lo nombra XNU.

[3] Jailbreak. Es un proceso que permite a los usuarios de los dispositivos obtener acceso raíz a la línea de comandos del Sistema Operativo iOS, para eliminar las limitaciones de uso y acceso impuestas por Apple. Una vez liberados, los usuarios de iPhone pueden descargar extensiones y temas que no están disponibles a través de App Store[5].

[4] User After Free. Es el resultado de acceder al contenido de la dirección de memoria reservada después de que ésta ha sido liberada[6].

desactivar los certificados *SSL*[5] (*Secure Sockets Layer*) y aprovechar un ataque hombre en medio para visualizar y analizar el tráfico *HTTPS* que generan las aplicaciones.

II. Arquitectura del Sistema Operativo iOS

La seguridad en los sistemas operativos es fundamental para el correcto funcionamiento de los elementos ***sandbox***, ***kernel***, entre otros que lo conforman, debido a que se debe de salvaguardar la Confidencialidad, Integridad y Disponibilidad en el tratamiento de la información del usuario.

Apple proporciona en su documentación oficial un diagrama de arquitectura de seguridad, para que el investigador comprenda como implementa las funciones de seguridad en su tecnología, ver Figura 1.

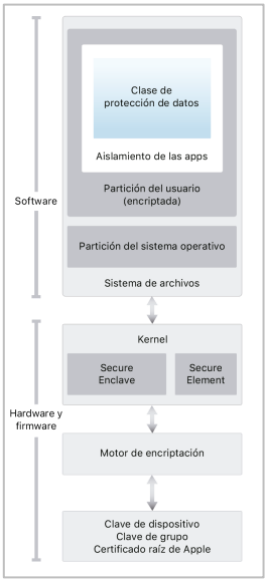

Figura 1. Diagrama de arquitectura de seguridad de iOS (Fuente: Apple)

III. Modelo de seguridad en iOS

En la parte superior de la arquitectura de seguridad está el Sistema Operativo, conformado por el sistema de archivos, la Partición del Sistema Operativo y la Partición del usuario. Dentro de la Partición de usuario se encuentra el área segura denominada ***sandbox*****,** en la capa inferior se encuentra el ***kernel,*** que tiene control completo sobre todo lo que ocurre en el sistema y se encuentra en modo lectura.

Apple implementa más elementos de seguridad que no pública, pero se encuentran presentes en las últimas versiones del Sistema Operativo, por ejemplo, el ***ASLR***[6] (*Address Space Layout Randomization*), ***KASLR***[7] (*Kernel Address Space Layout Randomization*), ***KPP***[8] (*Kernel Patch Protection*), y el ***KTRR***[9].

IV. Vulnerabilidades importantes en la historia de iOS

En esta sección se realiza un análisis de las vulnerabilidades públicas que afectan al Sistema Operativo iOS, de las cuales, las más críticas y relevantes han sido encontradas por el investigador en seguridad Ian Beer. Una vez que Apple es alertada de los fallos encontrados por este u otros investigadores, la empresa corrige el fallo y después libera actualizaciones con la corrección del mismo. Para este caso de estudio, se basó en el reporte de seguridad que Beer hace público con las vulnerabilidades encontradas, las cuales se analizan a continuación.

---

[5] SSL. Usa cifrado para proporcionar privacidad de mensaje, integridad de mensaje y autenticación de cliente y servidor[7].

[6] ASLR. Se trata de una técnica dirigida a prevenir que un atacante que sabe previamente que una función de un programa es vulnerable y puede ser explotada, pueda acceder a ella impidiéndole conocer la dirección de memoria que ocupa dicha función dentro del proceso en ejecución[8].

[7] KASLR. Aleatorizar la ubicación del código del propio kernel en memoria cuando arranca el sistema[8].

[8] KPP. Comprueba la integridad del kernel en intervalos de tiempo aleatorio.

[9] KTRR. Marca la memoria como de solo lectura a nivel de Hardware[3]. Este mecanismo de seguridad se encuentra presente a partir del iPhone 7.

4.1 Async wake ios

Es un error en ***IOSurface***[10]***,*** pertenece al controlador de ***IOKit***[11]***,*** que permite aprovecharse de una técnica denominada ***User After Free*** con la cual se construye una serie de puertos falsos, ésta está registrada en el ***CVE-2017-13861****[9]*, adicionalmente, para mejorar la capacidad de explotación, Beer utiliza otra vulnerabilidad documentada en el ***CVE-2017-13865****[10]* que le permite revelar los punteros de ***kernel***.

4.2 Triple fetch

Es un error de lógica en el framework ***libxpc***[12]***,*** registrado bajo el ***CVE-2017-7047****[11],* que permite inyectar código en el momento en que se encuentra procesando el Sistema Operativo el mensaje original, esto se debe a que el canal de comunicación sigue abierto después de haber terminado de enviar el mensaje original.

Beer realiza una prueba de concepto que permite tener acceso de forma remota al Sistema Operativo, sin los límites que genera el ***sandbox***, es decir, que puede visualizar los identificadores de los procesos (PID) y tener un acceso controlado a ellos. Ésta es la primera explotación que se necesita para tener un *jailbreak*. El alcance de esta prueba es un escape de ***sandbox***, se necesita tener acceso al dispositivo físicamente al menos una vez.

4.3 Extra recipe

Mach se concibió originalmente como un microkernel de comunicaciones capaz de ejecutarse como un ***kernel*** independiente. ***Mach*** proporciona ejecución altamente paralela, por ejemplo, hilos programados de forma preventiva y otras funciones más de la misma índole. *XNU* tiene un pequeño número de llamadas adicionales al sistema que hacen uso de ***Mach*** también conocidas como ***trampas Mach****[12].*

El fallo de seguridad se aprovecha cuando los ***mensajes Mach*** leen y escriben fuera de los límites de las asignaciones ***kalloc***[13], después se envía un ***mensaje Mach*** por medio de la llamada ***mach_voucher***. Con esta función se envía una gran cantidad de datos controlados por el emisor hasta que el hilo generado se cuelga[13], esta vulnerabilidad se encuentra registrada con el ***CVE-2017-2370****[14]*.

V. Planeación

Para comprometer la seguridad de un dispositivo móvil, es necesario infectarlo con una aplicación maliciosa, se puede alcanzar el objetivo de dos formas: de manera local o remota.

5.1 Instalación local

De manera local se necesita tener físicamente el dispositivo del usuario. Existen dos formas de instalar una aplicación maliciosa de forma local: ***Xcode***[14] y ***Cydia Impactor***[15]*.*

[10] IOSurface. Comparte datos de buffer acelerados por hardware (framebuffers y texturas) en múltiples procesos. Administra la memoria de imagen de manera eficiente[15].

[11] I/O Kit. Es una colección de frameworks del sistema, librerías, herramientas y otros recursos para la creación de controladores de dispositivos en OS X[16].

[12] libxpc.dylib. Es una API de XPC, XPC es el estándar de facto para las comunicaciones de procesos en iOS. s

[13] kalloc. El kernel llama a la función kalloc() cuando necesita memoria

[14] Xcode. Es el entorno de desarrollo integrado (IDE) oficial de Apple para programar aplicaciones en iOS.

[15] Cydia Impactor. Es una herramienta de interfaz gráfica de usuario para trabajar con dispositivos móviles, instala archivos IPA en iOS[17].

5.1.1 Xcode

Para la instalación mediante ***Xcode*** es necesario tener el código fuente, ***Xcode*** otorga el acceso al archivo ***IPA***[16], esto es posible si se cuenta con la licencia de paga de desarrollador de Apple, de lo contrario se instala la aplicación sin obtener el archivo ***IPA***. Después de instalar la aplicación se necesita confiar en el certificado de desarrollador para hacer uso de la aplicación, como se muestra en la Figura 2.

Figura 2. Instalación de app mediante Xcode

5.1.2 Cydia Impactor

Para realizar la instalación de la aplicación por la opción de ***Cydia Impactor*** se crea un script *bash*, como se muestra en la Figura 3, el script *bash* tiene los comandos para compilar el código fuente y generar el archivo ***IPA***, para después instalarlo mediante la herramienta ***Cydia Impactor,*** como se describe en la Figura 4, es necesario que se proporcione la confianza al certificado del desarrollador, para que la aplicación sea funcional.

```
#!/bin/bash
echo "Compilando"
$(which xcodebuild) clean build CODE_SIGNING_REQUIRED=NO CODE_SIGN_IDENTITY="" \
-sdk `xcrun --sdk iphoneos --show-sdk-path` -arch arm64
mv build/Release-iphoneos/Prueba.app Prueba.app
mkdir Payload
mv Prueba.app Payload/Prueba.app
echo "Comprimiendo en .ipa"
zip -r9 Prueba.ipa Payload/Prueba.app
rm -rf build Payload
echo "Listo!"
```

Figura 3. Script bash

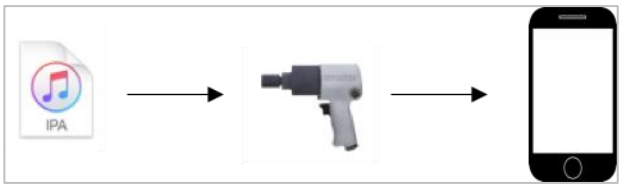

Figura 4. Instalación de app mediante Cydia Impactor

[16] IPA. Las aplicaciones de iOS se distribuyen en archivos IPA (Paquete de aplicaciones de iOS) un paquete comprimido que contiene el código necesario de la aplicación compilada y los recursos necesarios para ejecutar la aplicación[18].

5.2 Instalación remota

Para utilizar la aplicación en forma remota, es necesario que el usuario la instale y otorgue el nivel de confianza al certificado del desarrollador, como se describe en la Figura 5, para tal efecto, se pueden utilizar técnicas de Ingeniería Social.

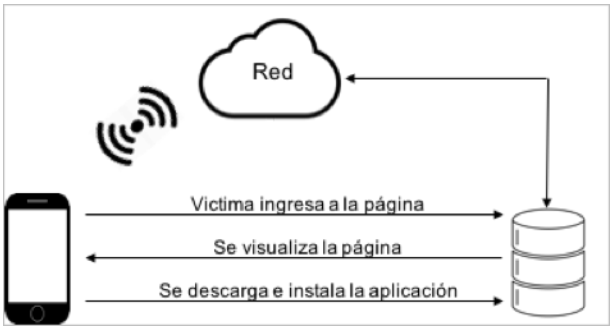

Figura 5. Instalación de app vía remota

VI. Explotación

De los CVE presentados en la sección IV, el de mayor impacto y que afecta directamente a la versión 11 de Sistema Operativo iOS es ***Async wake ios***[19], están documentadas con los ***CVE-2017-13865*** y ***CVE-2017-13861***. Beer hace una prueba de concepto del cual se retoma para la implementación de la aplicación que se presenta en esta sección.

La explotación de la vulnerabilidad en el ***kernel*** del sistema realizada por Beer fue explotada mediante la técnica de ataque de ***User After Free***. Para demostrar esta vulnerabilidad, se realizó una prueba de concepto, por medio de la instalación de una aplicación de forma local que permite consumir recursos del ***kernel*** mediante la creación de un puerto de tareas que saturan la memoria usando asignaciones ***kalloc*** que posteriormente el sistema se encargará de liberar y reasignar el búfer ***kalloc*** reemplazándolo con un puerto falso de

tareas del ***kernel***, permitiendo así obtener acceso al ***kernel*** para montarlo en modo lectura y escritura. Dentro del ***kernel*** se tienen los elementos necesarios para realizar un bypass al ***KPP*** y ***KTRR***, posteriormente se crean parches para habilitar ***TFP0***[17] (task_for_pid_0), ***setuid*** (0) (acceso raíz), ***AMFI***[18] (Apple Mobile File Integrity). Con los pasos anteriores se obtiene un *jailbreak* funcional. Para las pruebas de *jailbreak* se usó un iPhone 5s con iOS 11.0.2 como se muestra en la Figura 6.

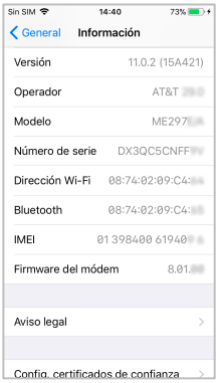

Figura 6. Información del dispositivo móvil

Para poder hacer uso de la aplicación en el dispositivo, se optó por instalarla siguiendo la metodología usada en ***Xcode***. En la siguiente Figura se observa la aplicación de ***Prueba*** instalada, además de aplicaciones nativas del Sistema Operativo iOS y WhatsApp.

Figura 7. Aplicación Prueba instalada

Para ejecutar nuestra aplicación ***Prueba*** es necesario confiar en el certificado del desarrollador, como se describe en la Figura 8. En el dispositivo se deben de seguir los siguientes pasos: ***Configuración*** -> ***General*** -> ***Administración de dispositivos*** -> ***Certificado del Desarrollador*** -> ***Confiar***.

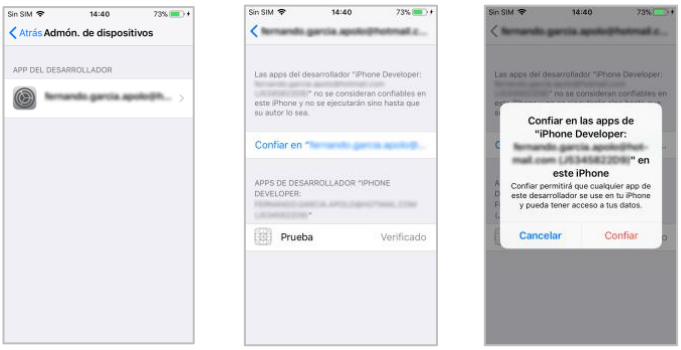

Figura 8. Pasos para confiar en el certificado del desarrollador

Una vez que se confió en el certificado, la aplicación puede abrirse, como se muestra en la Figura 9.

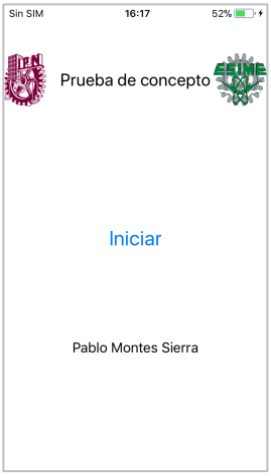

Figura 9. Pantalla de inicio de la aplicación Prueba

[17] TFP0. Al habilitar TFP0 da acceso al identificador de proceso 0, cuando Apple lo tiene bloqueado, permitiendo que cualquier ejecutable en modo root llame a task_for_pid 0 (TFP0).

[18] AMFI. Debe ser parcheado para permitir que funcionen binarios no firmados por Apple.

Al presionar el botón “Iniciar”, se obtienen datos del dispositivo tales como el número de compilación, tipo de dispositivo y versión del ***kernel*** desde el depurador de ***Xcode***, ver Figura 10.

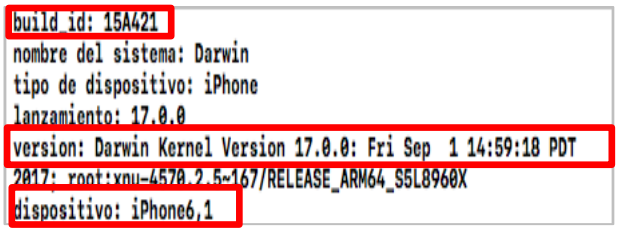

Figura 10. Depurador No.1

Posteriormente, se comienza a hacer asignaciones de ***kalloc.4096*** apuntando al puerto objetivo, se crean varios ***puertos match*** para garantizar que la paginación sea dirigida al puerto objetivo, ver Figura 11.

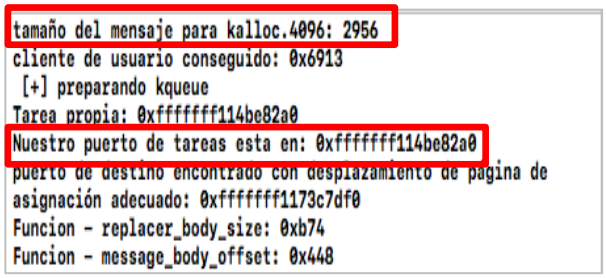

Figura 11. Depurador No.2

En seguida, se realiza una reasignación que contiene ***mensajes IPC*** (Comunicación entre Procesos) con un puerto de tareas falso***,*** esto es aprovechado para hacer una lectura arbitraria de ***IPC*** y obtener la dirección de memoria de ***vm_map*** del kernel, ver Figura 12.

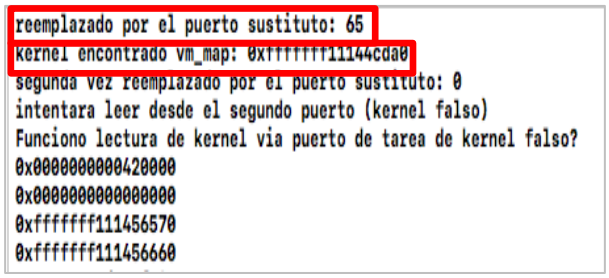

Figura 12. Depurador No.3

A continuación, se crea un archivo de pruebas para validar que se pueda escribir en el ***kernel***. El puerto de tareas del ***kernel*** y ***TFP0*** son reasignados, finalmente el ***kernel*** se remonta en modo lectura/escritura (R/W) y se valida, ver Figura 13.

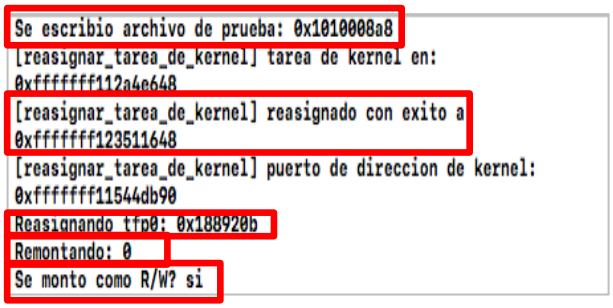

Figura 13. Depurador No.4

Una vez realizado los pasos descritos, se copian binarios, previamente compilados para la arquitectura ARM64, son comandos para el Sistema Operativo iOS, ver Figura 14.

Figura 14. Depurador No.5

Se vuelve a leer las direcciones de memoria donde se inyectaron los binarios y posteriormente se valida que todo funcione correctamente, ver Figura 15.

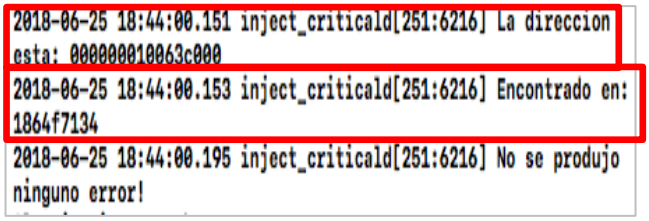

Figura 15. Depurador No.6

Un iPhone con el Sistema Operativo de fábrica, no tiene habilitada la conexión *SSH* (*Secure Shell*), en la Figura 16 se muestra una captura de pantalla de una computadora

que intenta conectarse con el iPhone y éste rechaza la conexión.

Figura 16. Conexión SSH antes de la explotación

En la Figura 17 se demuestra cómo se conecta el iPhone mediante *SSH*, la conexión es exitosa después de la explotación.

Figura 17. Conexión SSH después de la explotación

VII. Post explotación

En esta sección se muestran los resultados obtenidos después de la explotación, los cuales se basan en la extracción de información almacenada en el dispositivo procedente de aplicaciones nativas del Sistema Operativo y WhatsApp previamente instalada, además del análisis de trafico de las aplicaciones.

7.1 Extracción de información sensible del dispositivo

La información que se logra comprometer, una vez obtenidos los permisos de administrador del iPhone, son los contactos, mensajes de texto (SMS), localización por antenas telefónicas, base de datos de WhatsApp y registro de palabras por teclado.

7.1.1 Contactos

Cada que vez que se consulta, se hace un nuevo registro o se visualizan contactos desde la aplicación, Apple guarda dicha información en una base de datos la cual se llama: ***AddressBook.sqlitedb***, se encuentra en la siguiente ruta: */private/var/mobile/Library/AddressBook/*.

Se extrae y para poder visualizar su contenido se usa el manejador de base de datos SQLite 3, la cual contiene 30 tablas en las cuales la tabla *ABPerson* como se muestra en la Figura 18, cuenta con los nombres de los contactos donde *ROWID* es el identificador del contacto. En la tabla *ABMultiValue* se encuentra guardado el número del contacto, asociado a *record_id* que es el identificador del mismo, en las Figuras 18 - 19.

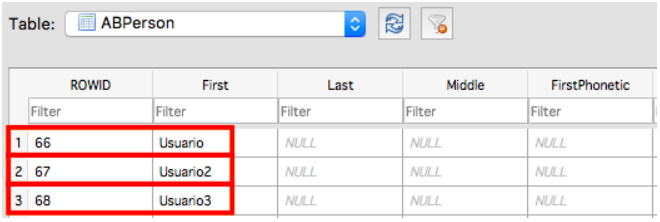

Figura 18. Base de datos de contactos – ABPerson

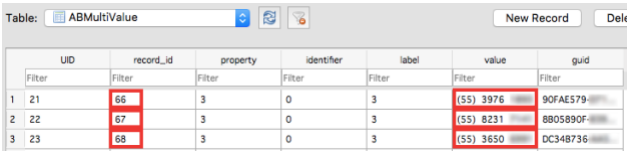

Figura 19. Base de datos de contactos – ABMultiValue

7.1.2 Mensajes de texto

La base de datos donde se guarda los mensajes de texto, se denomina ***sms.db***, está en la ruta: */private/var/mobile/Library/SMS/*. Al igual que en el registro de contactos, se usa el manejador de base de datos SQLite 3, esta base de datos contiene 10 tablas, como se describen en la Figura 20, la subtabla chat_identifier muestra el identificador del chat, así como el correo o número del emisor, el correo es un campo que Apple usa para la mensajería de iMessage vinculado al Apple ID. En la tabla message se muestra el mensaje en texto plano, que puede apreciarse en la Figura 21.

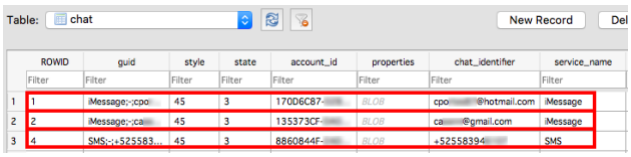

Figura 20. Base de datos de mensajes – chat

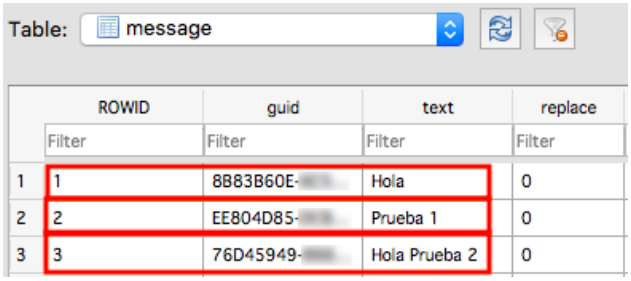

Figura 21. Base de datos de mensajes – message

7.1.3 Localización por antenas telefónicas

Cada vez que se conecta a una nueva antena telefónica, el equipo iOS realiza un registro, con el cual guarda la información de la conexión que se realiza con la base de datos **cache_encryptedB.db**, se encuentra en siguiente ruta: */private/var/root/Library/Caches/locationd/*.
Se visualiza en la tabla CellLocation los datos: MCC[19] (Mobile Country Code), MNC[20] (Mobile Network Code), LAC[21] (Location Area Code), CI[22] (Cell ID), UARFCN[23] (UTRA Absolute Radio Frequency Channel Number), latitud, longitud, entre otros datos que corresponde a la información de la SIM y la antena que brindó el servicio al dispositivo móvil.
En la Figura 22 se observan tres registros de localización en formato GPS, obtenidas a partir de los identificadores de celda. Se puede observar que la ubicación con el identificador 1 y 2 de la base de datos de la Figura 22 es representado con el punto verde en la Figura 23, mientras la ubicación con el identificador 3 de la base de datos de la Figura 22 es representado con el punto azul en la Figura 23.

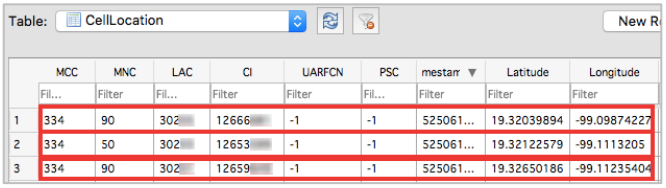

Figura 22. Base de datos de localización

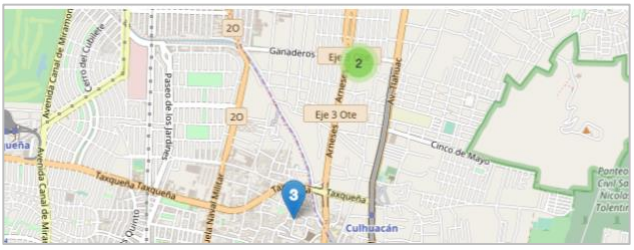

Figura 23. Mapa de localización

7.1.4 Base de datos de Whatsapp

Whatsapp genera bases de datos para el uso de la propia aplicación sin cifrar, una de las bases de datos se llama **ChatStorage.sqlite**, está en la ruta: */private/var/mobile/Containers/Shared/App Group/Hash-de-Whatsapp/*. Se usa el manejador de base de datos SQLite 3 para visualizar dicha base de datos la cual contiene 18 tablas: la tabla ZWACHATSESSION en ZPARTHERNAME muestra el nombre del contacto y el número del contacto en ZCONTACTJID en la siguiente Figura se puede visualizar.

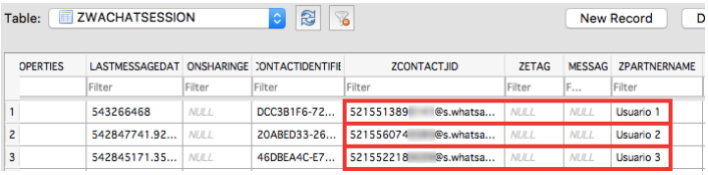

Figura 24. Base de datos de WhatsApp – chat sesión

En la tabla ZWAMESSAGE se muestra los mensajes enviados y recibidos, los cuales están presentes en ZTEXT, como se muestra en la Figura 25.

[19] MCC. Código de identificación del país[20].
[20] MNC. Código de identificación de la red del país[20].
[21] LAC. Código de área de ubicación[20].
[22] CI. Identificador de la celda móvil[20].
[23] UARFCN. Número de Canal de Radiofrecuencia Absoluto (downlink/uplink).

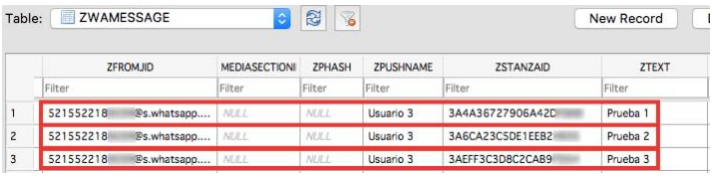

Figura 25. Base de datos de WhatsApp – mensajes

7.1.5 Registro de palabras por teclado

Cada vez que un usuario escribe manualmente una palabra en el iPhone, el dispositivo genera un archivo de diccionario dinámico que almacena palabras únicas para ese usuario. Por ejemplo, todo lo que se ingrese en un mensaje de texto, correo electrónico, nota u otra forma de ingreso de texto se almacenará en un archivo, el archivo es llamado ***es_MX-dynamic-text.dat***, las letras: ***es,*** indica el código de idioma (español), las letras: ***MX,*** indica el código de país (México), como se muestra en la Figura 26, el archivo se encuentra en la siguiente ruta: */private/var/mobile/Library/Keyboard/*.

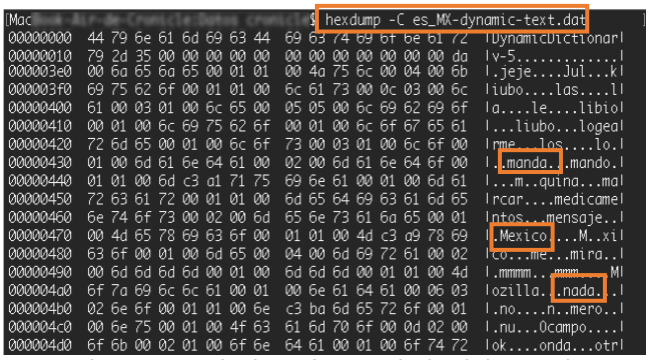

Figura 26. Diccionario – teclado del usuario

7.2 Análisis de trafico de las aplicaciones

Con el dispositivo en modo administrador, se puede desactivar los certificados *SSL* para visualizar en texto plano el tráfico que generan las aplicaciones por medio de ataque hombre en medio, ver Figura 27, con ello se analiza la información que extraen las aplicaciones de los usuarios, para esto se hace uso de la herramienta *Burp Suite*.

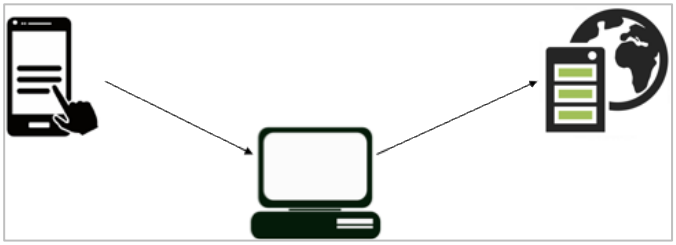

Figura 27. Ataque hombre en medio

7.2.1 Configuración de proxy en Burp Suite

Se inicia *Burp Suite*, se selecciona ***Proxy -> Options -> Add,*** se añade el puerto y la dirección IP del proxy, como se muestra en la Figura 28.

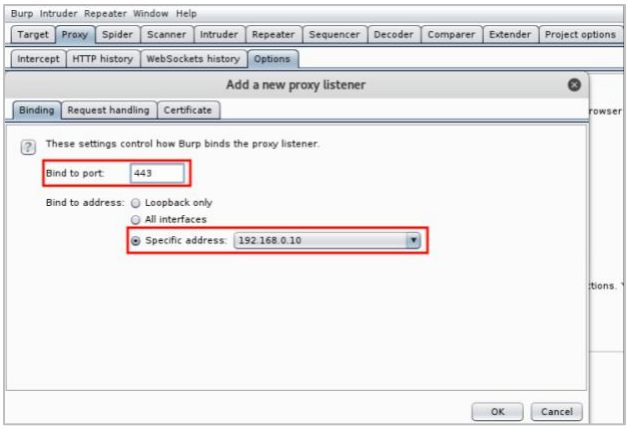

Figura 28. Configuración de proxy en Burp Suite

7.2.2 Configuración de parámetros de proxy en el dispositivo móvil

En el dispositivo móvil ir a ***Configuración -> Wi-Fi -> Nombre de la Red -> Configurar proxy -> Manual -> Servidor: IP definida en Burp Suite y Puerto: Puerto definido en Burp Suite -> Guardar,*** como se describe en la Figura 29.

Se debe instalar el certificado de *Burp Suite* en el dispositivo móvil, ir a ***navegador -> url: http://*burp -> CA Certificate -> Alerta: Permitir -> Instalar perfil: Instalar,*** como se muestra en la Figura 29***.***

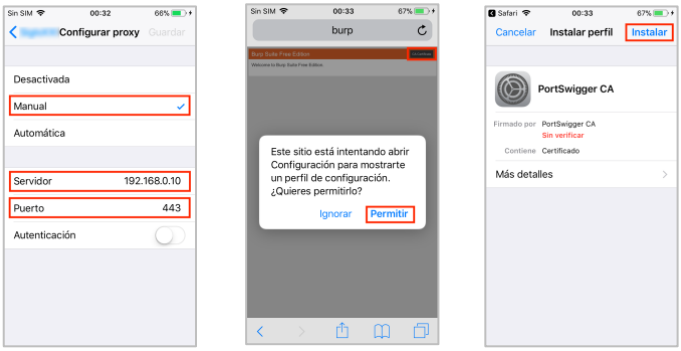

Figura 29. Configuración de parámetros de proxy en el dispositivo móvil

7.2.3 Ubicación

Se realizaron pruebas con la aplicación de ***Facebook***, en las cuales se desactivo la localización, durante el análisis se halla que la aplicación extrae el nombre del **punto de acceso** al que se está conectado, **dirección MAC**, **latitud**, **longitud** y **altitud**, como se muestra en la Figura 30.

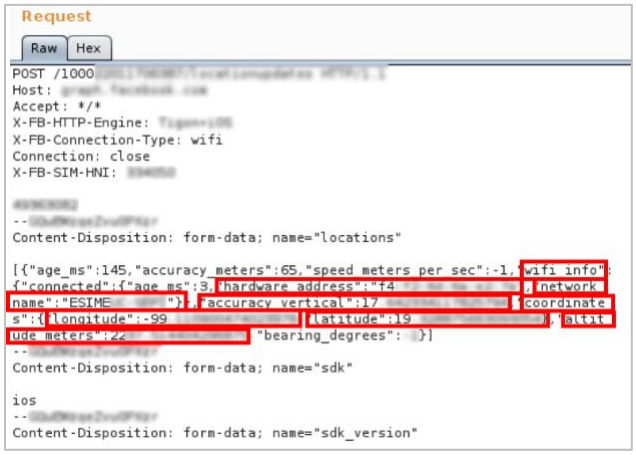

Figura 30. Análisis de tráfico de la aplicación de Facebook

7.2.4 Operadora

Se realizaron pruebas con la aplicación de ***Instagram***, en las cuales se desactivo la localización, aquí se encuentra que la aplicación extrae el **nombre de operadora**, **región** y habilita el **tracking**, como se muestra en la Figura 31.

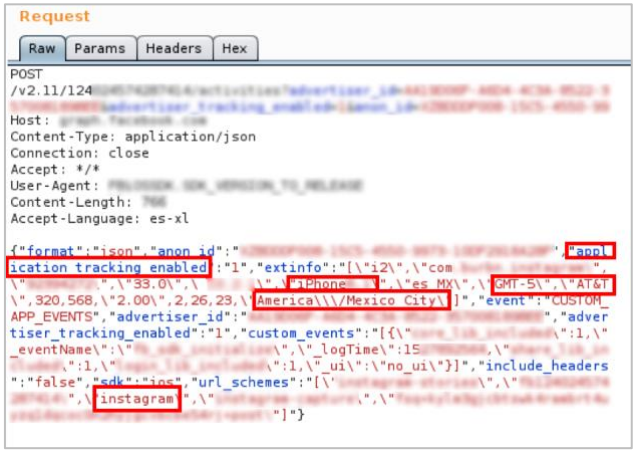

Figura 31. Análisis de tráfico de la aplicación de Instagram

VIII. Conclusión

Se logró comprometer la seguridad del Sistema Operativo iOS 11 a través de la explotación de divulgación de memoria y ***IOSurface***. Con esto se permite a un posible atacante extraer de forma no autorizada la información del dispositivo.

La principal aportación que se realizó durante la exposición del presente artículo, fue el demostrar el nivel de vulnerabilidad que poseen todas las aplicaciones del usuario después de adquirir privilegios de administrador utilizando conexión *SSH* y deshabilitando los certificados *SSL*. Toda información del usuario y del Sistema Operativo es fácilmente comprometida si un atacante logra infectar de forma local o remota un dispositivo con Sistema Operativo iOS, como se demostró en este trabajo.

Referencias

[1] Denise Giusto Bilić, Balance 2017: análisis de riesgos y amenazas móviles, URL: https://www.welivesecurity.com/la-es/2017/12/27/balance-2017-riesgos-amenazas-moviles/, fecha de consulta: 05/08/18.

[2] Citizenlab, The million dollar dissident, URL:

https://citizenlab.ca/2016/08/million-dollar-dissident-iphone-zero-day-nso-group-uae/, fecha de consulta: 05/08/18.

[3] Jonathan Levin, OS Internals: Volume III security & insecurity, 2018.

[4] Apple, Seguridad de iOS - iOS 11 - Enero de 2018, URL: https://www.apple.com/es/business/docs/iOS_Security_Guide.pdf, fecha de consulta: 05/08/18.

[5] Huang Heqing, An Analysis on iOS Jailbreak, URL: https://papers.put.as/papers/ios/2011/ios_jailbreak_analysis.pdf, fecha de consulta: 05/08/18.

[6] Security Intelligence, Use-after-frees: That pointer may be pointing to something bad, URL: https://securityintelligence.com/use-after-frees-that-pointer-may-be-pointing-to-something-bad/, Fecha de consulta: 07/08/18.

[7] Stephen Northcutt, SSL/TLS, URL: https://www.sans.edu/cyber-research/security-laboratory/article/ssl-tts, Fecha de consulta: 07/08/18.

[8] Daniel López Azaña, Diferencias entre ASLR, KASLR y KARL, URL: http://www.daniloaz.com/es/diferencias-entre-aslr-kaslr-y-karl/, Fecha de consulta: 07/08/18.

[9] NIST, CVE-2017-13861 Detail, URL: https://nvd.nist.gov/vuln/detail/CVE-2017-13861, Fecha de consulta: 08/08/18.

[10] NIST, CVE-2017-13865 Detail, URL: https://nvd.nist.gov/vuln/detail/CVE-2017-13865, Fecha de consulta: 08/08/18.

[11] NIST, CVE-2017-7047 Detail, URL: https://nvd.nist.gov/vuln/detail/CVE-2017-7047, Fecha de consulta: 08/08/18.

[12] Apple, Mach Overview, URL: https://developer.apple.com/library/archive/documentation/Darwin/Conceptual/KernelProgramming/Mach/Mach.html, fecha de consulta: 08/08/18.

[13] Ian Beer, Exception-oriented exploitation on iOS, URL: https://googleprojectzero.blogspot.com/2017/04/exception-oriented-exploitation-on-ios.html, Fecha de consulta: 10/08/18.

[14] NIST, CVE-2017-2370 Detail, URL: https://nvd.nist.gov/vuln/detail/CVE-2017-2370, Fecha de consulta: 12/08/18.

[15] Apple, IOSurface, URL: https://developer.apple.com/documentation/iosurface, Fecha de consulta: 14/08/18.

[16] Apple, IOKit Fundamentals, URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.693.3915&rep=rep1&type=pdf, Fecha de consulta: 13/08/18.

[17] Jay Freeman, Cydia Impactor, URL: http://www.cydiaimpactor.com/, Fecha de consulta: 15/08/18.

[18] Dominic Chell, Tyrone Erasmus, Shaun Colley, Ollie Whitehouse, The Mobile Application Hacker's Handbook, 2015.

[19] Ian Beer, iOS/MacOS kernel double free due to IOSurfaceRootUserClient not respecting MIG ownership rules, URL: https://bugs.chromium.org/p/project-zero/issues/detail?id=1417, Fecha de consulta: 15/08/18.

[20] ETSI, ETSI TS 123 003 v4.3.0 (2001-12), URL: https://www.etsi.org/deliver/etsi_ts/123000_123099/123003/04.03.00_60/ts_123003v040300p.pdf, Fecha de consulta: 17/08/18.