DESARROLLO DE UNA HERRAMIENTA PARA LA AUTOMATIZACIÓN DE LA EXPLOTACIÓN DE VULNERABILIDADES: ANDROID EXPLOIT SUGGESTER

Alejandro Fernández Barroso
Universidad Rey Juan Carlos
Master en Ciberseguridad y Privacidad

Resumen

En la fase de explotación de vulnerabilidades de una auditoría de seguridad, es de vital importancia conocer la mayor información posible del objetivo. Para ello el auditor se basará en los datos obtenidos en las fases anteriores de la auditoría: la recolección de información y el escaneo de puertos.

Uno de los datos más importantes de un sistema es la versión que corre en él, ya que con este dato se podrá conocer si el sistema está desactualizado o tiene vulnerabilidades.

Para facilitar la tarea del auditor y automatizar la explotación de vulnerabilidades, existen herramientas que dada la versión de un sistema operativo, proporcionan una lista de las distintas vulnerabilidades y sus correspondientes exploits, en el caso que existieran. Este tipo de herramientas se denominan Exploit Suggesters.

Este proyecto se basa en el desarrollo de una herramienta que automatice la explotación de vulnerabilidades para el sistema operativo Android, en este caso el nombre de la aplicación desarrollada es Android Exploit Suggester.

1. Objetivos

1.1. Descripción del problema

Con el fin de automatizar la fase de explotación de vulnerabilidades, se va a desarrollar una herramienta que permita obtener todas las vulnerabilidades existentes para la versión de Android sobre la que se esté ejecutando el programa. Haciendo uso de la herramienta se podrá conocer toda la información referente a cada una de las vulnerabilidades que pudieran existir en el sistema.

1.2. Objetivos

Una vez enunciado el problema se plantean los siguientes objetivos:

1. Desarrollo de una aplicación portable que se pueda ejecutar en cualquier versión de Android sin necesidad de instalar ninguna dependencia.
2. La aplicación debe tener tres modos de operación:
   a) Modo automático: la aplicación detecta la versión de Android sobre la que se está ejecutando.
   b) Modo manual: el auditor debe proporcionar por parámetro la versión de Android, de la cual se quieren conocer las vulnerabilidades, al ejecutar la herramienta.
   c) Modo online: la aplicación se conectará a Internet para consultar las vulnerabilidades. De esta manera, se obtendrán todas las vulnerabilidades existentes. Si no se hace uso de este modo de operación, las vulnerabilidades se consultarán en el repositorio local de la herramienta.

3. La herramienta debe proporcionar las siguientes características de cada vulnerabilidad:
   a) CVE.
   b) Puntuación.
   c) Versiones afectadas.
   d) Referencias.

2. Introducción

Una auditoría de seguridad consiste en un análisis de los diferentes activos de una organización con el fin de identificar los diferentes riesgos y vulnerabilidades que tienen dichos activos.

2.1. Tipos de auditoría

Existen diferentes tipos de auditorías de seguridad:

- **Auditoría interna.** En este tipo de auditoría se tiene acceso a todos los activos internos de la organización, por lo que el auditor se conecta directamente a la red interna para llevar a cabo la auditoría.
- **Auditoría externa.** El análisis se centra en los activos que la organización presenta hacia internet, es decir, los servicios públicos de la organización.
- **Auditoría web.** Las aplicaciones web suelen ser los activos más importantes para la organización, ya que la mayor parte de su negocio depende de ellos. En este tipo de auditoría el análisis se realiza sobre estas aplicaciones.
- **Auditoría wireless.** Esta auditoría se basa en el análisis de las redes inalámbricas de la organización. Suele ser considerada como una auditoría interna, pero por la diferenciación del entorno se suele separar.

2.2. Fases de la auditoría de seguridad

Dentro de la auditoría de seguridad existen diferentes fases, las cuales se componen de un conjunto de pruebas a realizar por el auditor, dichas fases son las siguientes:

2.2.1. Recolección de información

Es una de las fases más importantes de la auditoría ya que los resultados proporcionan mucha información para fases posteriores de la auditoría. Esta fase centra sus esfuerzos en la búsqueda de información del objetivo. Cabe destacar, que dentro de esta fase no se debe tener interacción con el objetivo, es decir, toda la información debe obtenerse a partir de fuentes públicas. Esto permite construir un perfil del objetivo sin interactuar con él.

Esta fase se suele denominar también footprinting. Algunas de las técnicas que se usan en esta fase son:

- Herramientas de red: whois y traceroute.
- Información del sitio web de la organización.
- Búsquedas en distintos buscadores de internet.
- Extracción de metadatos.
- OSINT [1].
- Ingeniería social.

2.2.2. Escaneo

El objetivo de esta fase es la obtención de los distintos puertos y servicios, que tienen abiertos los activos de la organización. Las pruebas realizadas en esta fase sí tienen interacción con los activos, por lo que se debe tener cuidado al llevarlas a cabo, ya que pueden dejar rastro e incluso alterar el comportamiento de los activos.

Los resultados obtenidos permiten identificar el estado de los puertos e identificar servicios y sistemas operativos.

Esta fase de suele denominar fingerprinting, y algunas de las herramientas que se suelen usar en esta fase son:

- Nmap [2].
- Hping [3].

2.2.3. Análisis de vulnerabilidades

El análisis de vulnerabilidades tiene como objetivo identificar si alguno de los objetivos es débil o susceptible de ser afectados o atacados de alguna manera.

Existen diferentes herramientas que permiten automatizar el análisis de vulnerabilidades, algunas de ellas son:

- Nessus [4].
- OpenVas [5].

2.2.4. Explotación

Esta fase se basa en las vulnerabilidades obtenidas en la fase anterior y se centra en la explotación de las mismas. La explotación consiste en un ataque directo hacia el objetivo.

Para llevar a cabo la explotación será necesario un exploit, que es un código que permite al auditor explotar una vulnerabilidad para comprometer un objetivo. Dependiendo del payload que se use en combinación con el exploit, se podrán obtener diferentes resultados de la explotación:

- Denegación de servicio.
- Ejecución de código remoto.
- Escalada de privilegios.
- Obtención de Shell remota.

2.3. Post-Explotación

La fase de post-explotación consiste en una serie de tareas que permiten al auditor vulnerar en más profundidad el objetivo. Alguna de las técnicas que se suelen emplear en esta fase son:

- Escalado de privilegios.
- Mantener el acceso.
- Creación de usuarios.
- Obtención de contraseñas del objetivo.
- Pivoting a otras redes.

Dentro de esta fase se suele realizar una última tarea que consiste en la eliminación de las huellas que se dejen al realizar la auditoría, de manera que no quede rastro, ni ninguna evidencia de que se ha llevado a cabo una explotación.

2.3.1. Generación del informe

La última fase de la auditoría de seguridad consiste en la elaboración del informe, en el cual se detallan las distintas pruebas realizadas y los resultados obtenidos tras la realización de la auditoría. Esta fase debe ser tratada muy cuidadosamente, ya que en el informe se refleja todo el trabajo realizado y además es el único documento que se le presenta al cliente final.

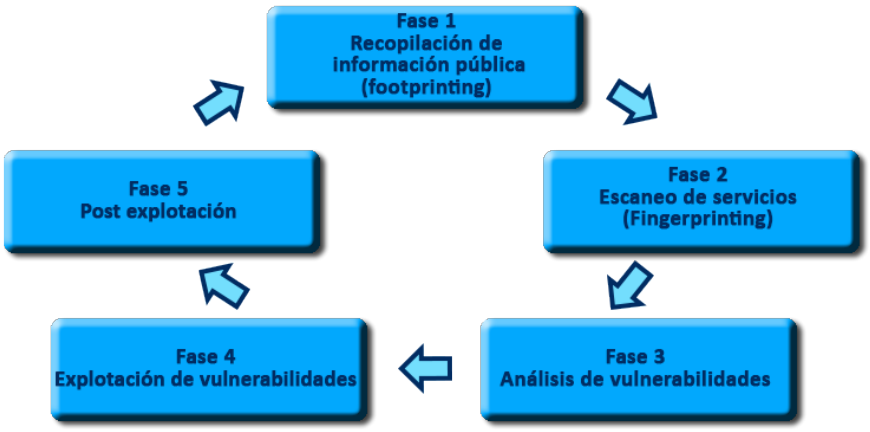

Figura 1: Fases de la auditoría de seguridad[6]

3. Estado del arte

Dentro de la fase de explotación de vulnerabilidades, uno de los datos que más ayuda proporciona al auditor es la versión del sistema operativo que está ejecutando el objetivo. Esto permite al auditor identificar las diferentes vulnerabilidades que existen para el objetivo y localizar los exploits disponibles para dichas vulnerabilidades. Existen diferentes herramientas que proporcionan toda la información sobre las vulnerabilidades que existen para una versión de un sistema operativo. La mayoría suelen ser servicios web que mantienen una base de datos actualizada con todas las vulnerabilidades para cada sistema, de manera que el usuario puede realizar distintas consultas y aplicar distintos filtros como la versión, la fecha de descubrimiento de la vulnerabilidad, la criticidad, etcétera. Algunos de los sitios más conocidos para la consulta de vulnerabilidades son:

- National Vulnerability Database [7].
- SecurityFocus [8].
- CVEDetails [9].
- Rapid7 [10].

Es posible que el auditor a la hora de realizar una auditoría no disponga de conexión a Internet, por lo que no tiene acceso a estas bases de datos. Para solventar este problema existen programas portables que mantienen una base de datos actualizada con toda la información referente a las vulnerabilidades existentes para todas las versiones de un determinando sistema operativo, de manera que el auditor puede obtener toda la información de las vulnerabilidades sin tener acceso a Internet. Estas herramientas se denominan Exploit Suggester, existen versiones de estas herramientas tanto para Windows como para Linux:

- Windows-Exploit-Suggester [11].
- linux-exploit-suggester [12].
- linux-Exploit-Suggester-2 [13].

Debido al crecimiento del uso de los sistemas móviles, cada vez existe más demanda de auditorías a este tipo de dispositivos. En cuanto a sistemas operativos, el que ha tenido mayor crecimiento los últimos años es el sistema Android.

Este trabajo consiste en el desarrollo de una herramienta que permita automatizar la fase de explotación de vulnerabilidades, anteriormente explicada, para el sistema operativo Android. Concretamente la herramienta consistirá en una base de datos portable que permite obtener toda la información referente a las vulnerabilidades del sistema. El nombre de la herramienta será **Android Exploit Suggester**.

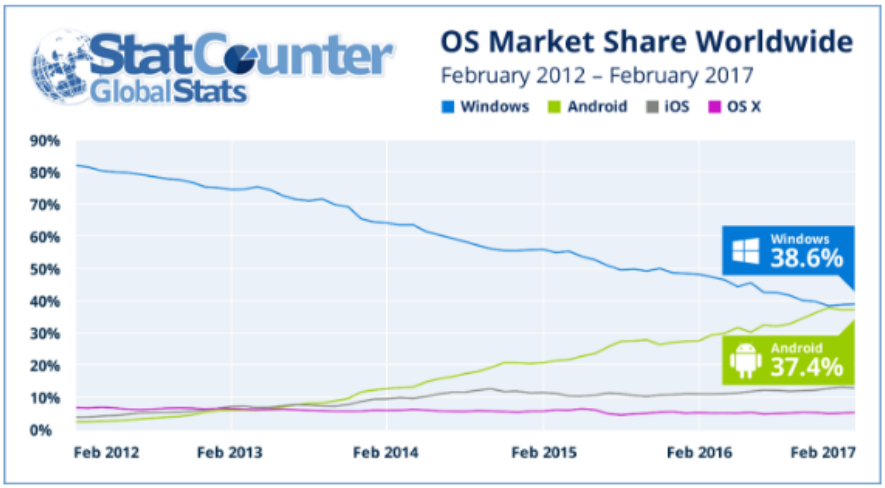

Figura 2: Evolución del uso de los sistemas operativos[14]

4. Descripción del trabajo desarrollado

4.1. Metodología

Para el desarrollo de este proyecto se ha llevado a cabo la metodología conocida con el nombre de "desarrollo en espiral" [16]. Este modelo nos permite desarrollar el software de manera incremental, aumentando progresivamente la dificultad de los problemas planteados y presentando al final de cada etapa una versión funcional del software. El desarrollo en espiral se compone de una serie de ciclos. Cada ciclo se compone de cuatro actividades, las cuales se detallan a continuación:

1. **Determinar objetivos.** En esta fase se fijan los objetivos que se deben cumplir al final de cada iteración, teniendo en cuenta el objetivo final del software.

2. **Análisis del riesgo.** Se estudian las causas de las amenazas y eventos no deseados junto con los daños que estos pueden producir. También se buscan soluciones para evitar dichos daños.

3. **Desarrollar y probar.** Se desarrolla el software y una vez que se finaliza se realizan una serie de pruebas para cerciorar que se cumplen los objetivos definidos en la iteración.

4. **Planificación.** En la última actividad se comienza a planificar la siguiente fase, teniendo en cuenta los objetivos obtenidos y los problemas surgidos en la fase actual para evitarlos en la siguiente.

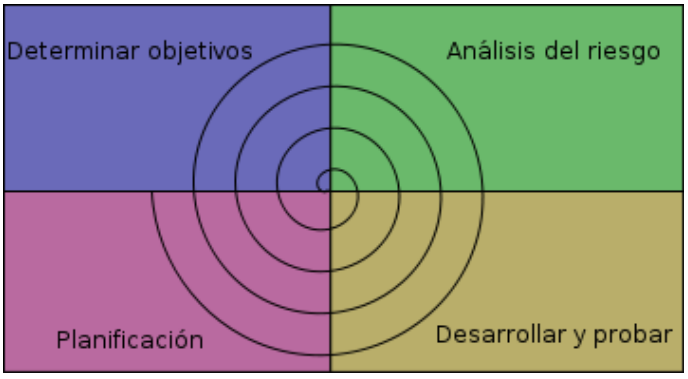

Figura 3: Desarrollo en espiral[17]

4.2. Iteraciones

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

- Desarrollo del modo de operación manual. En esta fase se implementa el funcionamiento de la herramienta proporcionando la versión Android por parámetro.
- Desarrollo del modo de operación automático. Se implementa el funcionamiento de la aplicación para la obtención de la versión de Android automáticamente.
- Tratamiento de las vulnerabilidades. Haciendo uso de diferentes fuentes se construye una estructura de datos con todas las vulnerabilidades existentes, de manera que pueda ser tratado por la herramienta.
- Desarrollo del modo de operación online. Se implementa la obtención de todas las vulnerabilidades existentes a través de Internet y se realiza un tratamiento de las mismas.
- Pruebas. En la última fase del proyecto se realizan diversas pruebas, en diferentes versiones de Android y en los diferentes modos de operación, para comprobar el correcto funcionamiento de la herramienta.

4.3. Infraestructura y herramientas

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

4.3.1. Android

Android es un sistema operativo para dispositivos móviles cuyo núcleo está basado en Linux.

Figura 4: Arquitectura del sistema Android[15]

Para el desarrollo del proyecto se usarán distintas versiones de este sistema operativo de manera que se compruebe el correcto funcionamiento de la aplicación.

4.3.2. GO

GO [18] es un lenguaje de programación concurrente de código abierto desarrollado por Google e inspirado en la sintaxis del lenguaje de programación C.

Las principales características de este lenguaje son:

- Lenguaje compilado, concurrente, imperativo, estructurado y orientado a objetos.
- Sintaxis similar al lenguaje C.
- Tipado estático y eficiencia similar al lenguaje C.
- Proporciona facilidades de lenguajes dinámicos como Python.
- Incluye un recolector de basura.

4.3.3. CVEDetails

CVEDetails es un servicio web que permite realizar búsquedas de vulnerabilidades entre diversos fabricantes, productos y versiones.

La información de las vulnerabilidades se toma del fichero .xml que proprocina NVD (National Vulnerability Database).

Además permite descargar esta información en formato .csv, lo que facilita su posterior tratamiento, ya que se puede filtrar por versiones, tipos de vulnerbilidades, puntuación, etc.

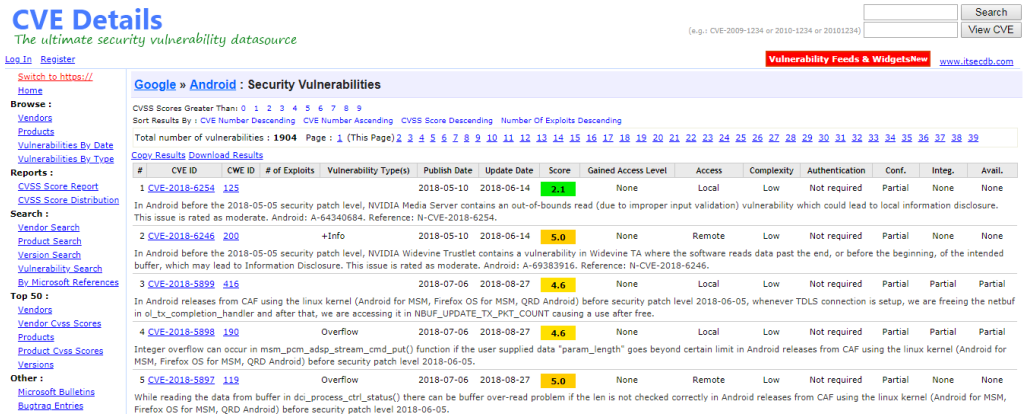

Figura 5: Lista de vulnerabilidades en Android proporcionada por CVE Details

4.3.4. Cve-search

Cve-search es un API HTTP desarrollado por el Centro de Respuestas frente a incidencias de Luxemburgo (CIRCL [19]), que permite realizar búsquedas de vulnerabilidades en software y hardware.

Este API nos permite descargar una lista de vulnerabilidades en formato .json para su posterior tratamiento.

Las fuentes de información de este API son:

- NVD.
- Common Platform Enumeration (CPE) [20].
- Common Weakness Enumeration (CWE) [21].
- toolswatch/vFeed [22].

5. Desarrollo de la aplicación

Para llevar a cabo el desarrollo de la aplicación es primordial identificar correctamente sobre que versión de Android se quieren conocer las vulnerabilidades. Para ello, tal y como se ha visto en los objetivos, se han desarrollado dos modos de operación.

El modo manual permite al auditor indicar por parámetro cual es la versión de la que se quieren conocer las vulnerabilidades. Para la implementación de este modo únicamente se obtendrá el valor pasado por parámetro, se comprobará que tiene un formato válido y se buscarán las vulnerabilidades coincidentes con dicha versión.

Para el modo automático, la aplicación debe obtener la versión del sistema automáticamente. En Android, con el comando getprop se pueden obtener las diferentes propiedades del sistema. La propiedad "ro.build.version.release" indica la versión del mismo. Por tanto, para conocer la versión, la aplicación ejecutará el comando:

getprop ro.build.version.release.

De esta manera, se tendrá versión que está corriendo en el sistema. Finalmente, al igual que en el modo anterior, se buscarán las vulnerabilidades existentes para dicha versión.

Para obtener las vulnerabilidades que afectan a la versión objetivo, la aplicación consulta una estructura de datos de vulnerabilidades. Estas vulnerabilidades se agrupan en un array de tipo Vuln. A su vez la estructura Vuln se compone de:

- CVE: variable de tipo String que se encarga de almacenar el CVE de la vulnerabilidad.
- versions: array de Strings que almacena las versiones que están afectadas por la vulnerabilidad.
- score: esta variable recoge la puntuación de la vulnerabilidad en Float32.
- description: variable de tipo String que contiene la descripción de la vulnerabilidad.
- ref: array de Strings que contiene las diferentes referencias de la vulnerabilidad.

```
type Vuln struct {
    CVE         string
    versions    []string
    score       float32
    description string
    ref         []string
}
```

Figura 6: Estructura de tipo Vuln

Para generar la información necesaria para completar la estructura de datos, se han empleado las bases de datos de los servicios CVEDetails y cve-search, los cuales se detallan en la sección 3.3 Infraestructura y herramientas.

Para localizar que vulnerabilidades afectan a la versión del sistema operativo objetivo, el programa iterará entre todo el array de vulnerabilidades y dentro de cada vulnerabilidad iterará entre las versiones afectadas comprobando si alguna coincide con la versión objetivo. Si existe coincidencia, se imprimirán todos los datos referentes a esa vulnerabilidad. Si no existe, se pasará a la siguiente vulnerabilidad hasta completar todas las vulnerabilidades.

5.1. Modo online

Uno de los requisitos definidos en los objetivos era la implementación de un modo de operación que permitiese conectarse a Internet para obtener todas las vulnerabilidades existentes. Usando este modo, el auditor obtendrá un listado actualizado de todas las vulnerabilidades que afecten a la versión objetivo.

Para el desarrollo del modo online se ha hecho uso del API cve-search. Este API permite obtener un json con todas las vulnerabilidades del sistema operativo Android ejecutando el siguiente comando:

wget -q http://cve.circl.lu/api/search/google/android -o android.json.

Con el parámetro "-q" no se imprime el proceso de descarga del programa wget y con el parámetro "-o android.json" se imprime la salida del comando en el fichero android.json.

Una vez finalizada la descarga, se procederá a parsear el fichero android.json. De esta manera se obtendrán todas las vulnerabilidades en una estructura de datos similar a la que está integrada en el código de la aplicación.

Obtenidas todas las vulnerabilidades, de manera similar que con las vulnerabilidades que están integradas en el código, se itera entre todas las vulnerabilidades y dentro de cada una de las versiones por las que está afectada la vulnerabilidad, imprimiendo en la salida las vulnerabilidades que afecten a la versión objetivo del sistema.

6. Funcionamiento de la aplicación

A continuación se va ha mostrar el funcionamiento de la herramienta en diferentes contextos, diferentes modos de operación y distintas versiones de Android.

En el primer ejemplo de ejecución se va a suponer que un auditor está realizando una auditoría y en fases anteriores ha descubierto que la máquina objetivo es una versión 5.0 de Android.

Localizado en la fase de explotación de vulnerabilidades, el auditor usará la herramienta en modo manual para descubrir las posibles vulnerabilidades existentes en el sistema, para así poder explotarlas.

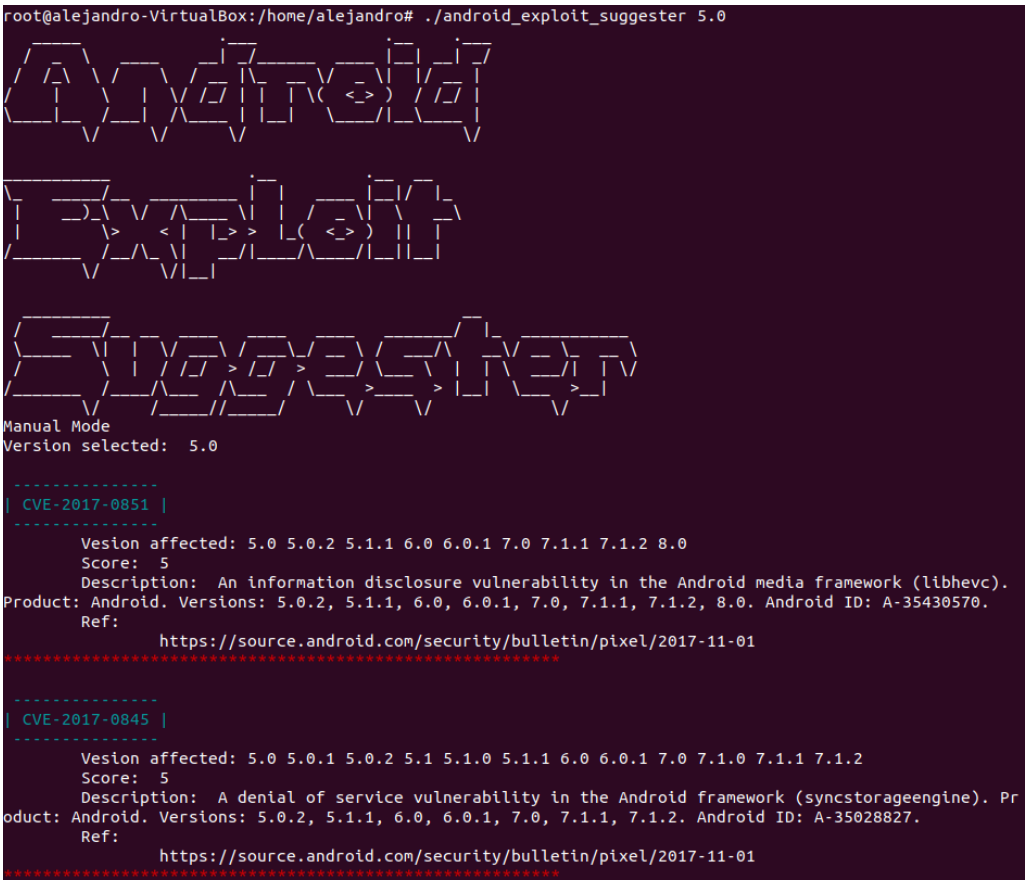

Figura 7: Ejecución de Android Exploit Suggester en modo manual

Es posible que el auditor quiera asegurarse que las vulnerabilidades que va a obtener estén actualizadas. En la siguiente ejecución se hace uso del modo manual en combinación con el modo online.

Figura 8: Ejecución de Android Exploit Suggester en combinación del modo manual y el modo online

Otro contexto en el que aplica el uso de la aplicación, es un caso en el cual un auditor dispone de acceso a un dispositivo Android del cual no conoce la versión y quiere conocer las vulnerabilidades que tiene el dispositivo para parchear el sistema y así protegerse frente ataques. Además el dispositivo tiene acceso a Internet, por lo que el auditor ejecutará la herramienta en modo online y en modo automático.

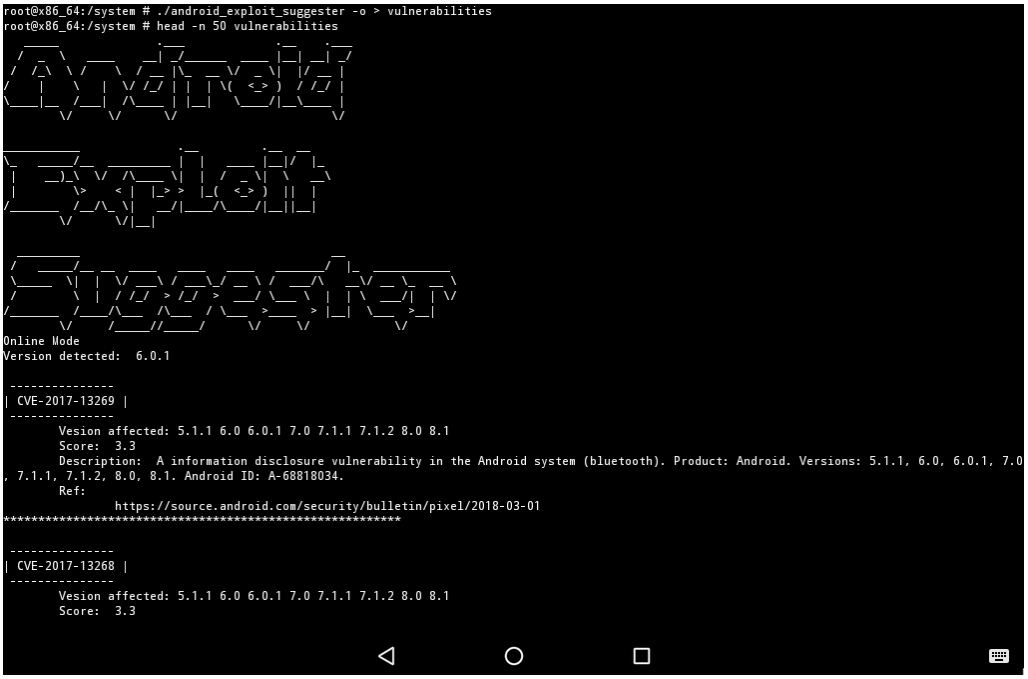

Figura 9: Ejecución de Android Exploit Suggester en combinación del modo automático y el modo online

Para finalizar, el último ejemplo de ejecución se basa en un auditor que tiene acceso al sistema, no conoce la versión y no tiene acceso a Internet. El auditor ejecutará la aplicación en su modo automático para conocer las vulnerabilidades existentes para la versión detectada.

Figura 10: Ejecución de Android Exploit Suggester en modo automático

7. Conclusiones

En esta memoria se ha documentado el proceso de desarrollo de una aplicación para la automatización de vulnerbilidades. En este apartado se va a analizar la consecución de los objetivos

propuestos y por último se redactarán unas ideas sobre cómo se podría mejorar el proyecto desarrollado.

7.1. Consecución de objetivos

En este apartado se va a analizar si se cumplen los objetivos definidos al inicio.

En primer lugar, la herramienta desarrollada es totalmente portable, esto quiere decir que no es necesaria la instalación de ninguna dependencia o componente para su correcto funcionamiento. Esto era uno de los objetivos que se puede considerar como realizado.

Otro de los objetivos marcados era la disponibilidad de tres modos de operación en la herramienta. Tal y como se ha podido comprobar en el apartado Funcionamiento de la aplicación, es posible ejecutar la herramienta en el modo manual, en el modo automático y en el modo online. Además en el apartado Desarrollo de la aplicación se puede comprobar cómo se han desarrollado dichos modos. Por tanto se puede dar este objetivo como completado.

La información que presenta la herramienta de cada vulnerabilidad es la definida en el último de los objetivos del proyecto como se ha podido comprobar en el apartado Funcionamiento de la aplicación. Este objetivo también ha sido cumplido.

7.2. Trabajos futuros

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

- **Integración continúa de nuevas vulnerabilidades en el código.** La aplicación debe ir actualizando las nuevas vulnerabilidades en el código, de manera que la aplicación esté siempre actualizada, sin necesidad de usar el modo online.

8. URL del vídeo de presentación del proyecto

La URL del vídeo en el cual se presenta el proyecto es: https://vimeo.com/289135101

9. URL de descarga

La URL del repositorio en el cual se encuentra el código de la aplicación es: https://github.com/MCYP-UniversidadReyJuanCarlos/17-18_alfeba

Referencias

[1] OSINT http://osintframework.com/ Agosto-2018

[2] Nmap https://nmap.org/ Agosto-2018

[3] Hping http://www.hping.org/ Agosto-2018

[4] Nessus https://www.tenable.com/products/nessus/nessus-professional Agosto-2018

[5] OpenVas http://www.openvas.org/ Agosto-2018

[6] Fases de la auditoría de seguridad https://www.i2multimedia.es/seguridad-informatica-madrid/ Septiembre-2018

[7] NIST NATIONAL VULNERABILITY DATABASE https://nvd.nist.gov/ Agosto-2018

[8] SecurityFocus https://www.securityfocus.com/ Agosto-2018

[9] CVEDetails http://www.cvedetails.com/ Agosto-2018

[10] Rapid7 https://www.rapid7.com/ Agosto-2018

[11] Windows-Exploit-Suggester https://github.com/GDSSecurity/Windows-Exploit-Suggester Agosto-2018

[12] Linux-exploit-suggester https://github.com/mzet-/linux-exploit-suggester Agosto-2018

[13] Linux-exploit-suggester-2 https://github.com/jondonas/linux-exploit-suggester-2 Agosto-2018

[14] Android overtakes Windows for first time http://gs.statcounter.com/press/android-overtakes-windows-for-first-time Septiembre-2018

[15] Arquitectura Android https://chsos20141910073.wordpress.com/tag/estructura-android/ Septiembre-2018

[16] A Spiral Model of Software Development and Enhancement, *Boehm, B, 1986, ACM*

[17] Modelo espiral https://es.wikipedia.org/wiki/Archivo:ModeloEspiral.svg Septiembre-2018

[18] Go https://golang.org/ Julio-2018

[19] CIRCL https://www.circl.lu/ Agosto-2018

[20] Common Platform Enumeration http://cpe.mitre.org/ Julio-2018

[21] Common Weakness Enumeration http://cwe.mitre.org/ Agosto-2018

[22] vFeed https://github.com/toolswatch/vFeed/ Agosto-2018