UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

MÁSTER EN CIBERSEGURIDAD Y SEGURIDAD DE LA INFORMACIÓN

TRABAJO FIN DE MÁSTER

Automatización de auditorías de código en Android

Mónica Pastor Abánades

Autora: Mónica Pastor Abánades

Director: Cristian Barrientos

Noviembre, 2020

RESUMEN

El objetivo de este Trabajo de Fin de Master es la creación de una herramienta para la realización de auditorías estáticas de código en aplicaciones Android de manera automatizada, proveyendo al analista de la mayor información disponible de las aplicaciones auditadas, incluyendo información y que le permita un posterior triaje de las evidencias o findings encontrados.

Además de las vulnerabilidades que se puedan encontrar en las aplicaciones, hemos puesto el foco en la puesta en valor de las buenas prácticas en seguridad que se realicen durante la implementación, identificando en qué puntos se codifica de manera segura, para no solo quedarnos con la parte negativa sino hacer un refuerzo positivo donde se esté realizando correctamente.

Por último, debido a que en el sistema operativo Android nos encontramos con frecuencia numerosas muestras de malware, se han incluido funcionalidades en la herramienta para hacer una homologación completa de las aplicaciones, enfocándose tanto en la seguridad como en la búsqueda de código malicioso en estas.

La implementación consistirá en una web que permitirá la subida de apks y, tras la realización de una serie de pasos: desde el decompilado de la aplicación, extracción de información, procesamiento mediante algoritmos y creación del modelo en base de datos, hasta la posterior visualización de este modelo completo que permita el análisis de las evidencias y la verificación de estas.

A lo largo de este trabajo se verá en qué se ha tenido en cuenta para la implementación, qué riesgos se intentan evitar con esta herramienta y una explicación detallada sobre todas las funcionalidades que se han implementado. Para finalizar, se indicará cómo utilizar la herramienta para la realización de auditorías estáticas de manera muy fácil y a la vez muy potente.

Además, en otro de los puntos, indicaremos las distintas integraciones que ofrece nuestra aplicación, como son: la inclusión de la base de datos de Malware DB, el uso de la API de Virus Total y, por último, el envío automático de las evidencias encontradas a la herramienta de gestión de defectos Defect Dojo.

Por último, para finalizar el trabajo, enunciaremos las conclusiones que hemos obtenido durante la realización del desarrollo y destacaremos las posibles mejoras de la aplicación de cara a futuro.

ÍNDICE

CAPÍTULO 1. INTRODUCCIÓN 12
1.1 Motivación 12
1.2 Estado del Arte 13
1.3 Objetivos 13
1.4 Estructura de la memoria 13
CAPÍTULO 2. FUNDAMENTOS SOBRE ANDROID 15
2.1 Aplicaciones 15
2.2 Componentes 16
2.2.1 Activities 16
2.2.2 Services 16
2.2.3 Content Providers 16
2.2.4 Broadcast Receivers 16
2.2.5 Intents 17
2.3 Seguridad 17
2.3.1 Sandboxing 17
2.3.2 Permissions 17
2.4 Auditorías 18
CAPÍTULO 3. PRINCIPALES RIESGOS DE SEGURIDAD EN ANDROID 21
3.1 Autenticación insuficiente 21
3.2 Autorización inadecuada 22
3.2.1 Access control 22
3.2.2 Application permissions 23
3.3 Comunicación insegura 25
3.3.1 Acceso a URLs sin SSL/TLS 25
3.3.2 Certificados 25
3.4 Guardado inseguro de información 26

3.4.1 Weak cryptography 26
3.4.2 SQL databases 28
3.4.3 Storage 28
3.5 Sensitive Information leak 29
3.5.1 Sensitive information in log 29
3.5.2 Hardcoded URLs o IPs 30
3.5.3 Hardcoded keys 30
3.6 Reverse Engineering 31
3.6.1 Root Detection 31
3.6.2 Debugger Detection 32
CAPÍTULO 4. HERRAMIENTA DE AUDITORÍAS ESTÁTICAS 33
4.1 Reglas o Patterns 33
4.2 Creación de la auditoría 35
4.3 Información recopilada durante el escaneo 38
4.3.1 Información de la aplicación 39
4.3.2 Permissions 39
4.3.3 Security Info 41
4.3.4 Activities 41
4.3.5 Components 41
4.3.6 Certificates 42
4.3.7 Strings 42
4.3.8 Files 44
4.3.9 Información sobre VirusTotal 45
4.4 Findings 48
4.4.1 Creación de findings de manera manual 49
4.5 Triaje de los resultados 50
4.5.1 Editar criticidad 50
4.5.2 Editar estado 50

4.5.3 Ver findings 51
4.5.4 Ver findings 51
4.5.5 Enviar a Defect Dojo 53
4.6 Mejores prácticas en seguridad 55
4.7 Informe de resultados 56
CAPÍTULO 5. DESARROLLO DE LA HERRAMIENTA 59
5.1 Herramientas utilizadas 59
5.2 Herramientas para la auditoría 60
5.2.1 Acceso al APK 60
5.2.2 Reversing de la APK 61
5.2.3 Análisis estático 61
5.2.4 Triaje de resultados 62
5.2.5 Informe de resultados 62
CAPÍTULO 6. DESPLIEGUE DE LA HERRAMIENTA 63
6.1 Configuración de la herramienta 63
6.2 Integraciones con otras aplicaciones 65
6.2.1 Defect Dojo 65
6.2.2 Virus Total 65
6.2.3 Malware DB 66
6.3 Componentes de la aplicación 66
6.3.1 Base de datos PostgreSQL 67
6.3.2 Servidor NGINX 68
6.3.3 Herramienta de auditorías 69
6.3.4 Volúmenes 71
6.4 Despliegue de la aplicación 72
CAPÍTULO 7. CONCLUSIONES Y PROPUESTAS 75
7.1 CONCLUSIONES 75

7.2 Trabajo futuro y posibles ampliaciones 76

CAPÍTULO 8. BIBLIOGRAFÍA 77

CAPÍTULO 9. CONTENIDO DEL ENTREGABLE 79

ÍNDICE DE FIGURAS

Figura 1 - Findings de Hardcoded API Keys....22
Figura 2 - Findings de autorización....22
Figura 3 - Ejemplo de application permissions....23
Figura 4 - Finding de Exported Component....24
Figura 5 - Finding de component backup....24
Figura 6 - Finding de debuggable component....24
Figura 7 - Acceso a URLs sin TLS....25
Figura 8 - Imporper certificate validation....25
Figura 9 - Ejemplos de SSL Pinning....26
Figura 10 - Findings de Insecure Random Number....26
Figura 11 - Ejemplos de Secure Random Number....27
Figura 12 - Findings de Weak Cryptography....27
Figura 13 - Findings de Weak Hash algorithm....28
Figura 14 - Ejemplo de extracción de información de una BD....28
Figura 15 - Findings de operaciones con ficheros....29
Figura 16 - Ficheros relevantes en escaneo....29
Figura 17 - Ejemplo de log sensitive information....30
Figura 18 - Ejemplo de Hardcoded URLs o IPs....30
Figura 19 - Ejemplo de hardcoded keys....31
Figura 20 - Ejemplo de Root detection....31
Figura 21 - Ejemplo de debugger detection....32
Figura 22 - Menú para acceder a las reglas/patterns....33
Figura 23 - Lista de reglas/patterns....34
Figura 24 - Editar el estado de las patterns....34
Figura 25 - Cambio de estado en las reglas....35
Figura 26 - Botón de creación de una aplicación....35

Figura 27 - Menú de Creación....35
Figura 28 - Creación de la aplicación a auditar....36
Figura 29 - Creación de un escaneo de la aplicación a auditar....36
Figura 30 - Dashboard principal de la herramienta....37
Figura 31 - Dashboard de una aplicación....37
Figura 32 - Botón de refresh del scan....38
Figura 33 - Menú lateral con información del escaneo....38
Figura 34 - Información de la aplicación....39
Figura 35 - Ejemplo de permisos de una apk....40
Figura 36 - Menú base de datos de Permissions....40
Figura 37 - Other Permissions....40
Figura 38 - Security Info de la aplicación....41
Figura 39 - Listado de activities....41
Figura 40 - Listado de components....42
Figura 41 - Certificados....42
Figura 42 - Strings de una aplicación....43
Figura 43 - Malware DB....43
Figura 44 - Files de una aplicación....44
Figura 45 - Visualización del Manifest.xml....45
Figura 46 - Información VT sobre Radar Covid....46
Figura 47 - Información Antivirus Radar Covid....46
Figura 48 - Información VT malware....47
Figura 49 - Información Antivirus malware....47
Figura 50 - VT opciones....48
Figura 51 - Findings por categoría de una aplicación....48
Figura 52 - Findings de una aplicación....49
Figura 53 - Creación de findings manuales....49
Figura 54 - Visualización de un finding manual....49

Figura 55 - Edición de criticidad de los findings....50
Figura 56 - Edición de estado de los findings....51
Figura 57 - Visualización de los findings para el triaje....51
Figura 58 - Visualizar un finding....52
Figura 59 - Visualizar fichero de un finding....52
Figura 60 - Enviar findings a Defect Dojo....53
Figura 61 - Visualizar id del finding de Defect Dojo....53
Figura 62 - Visualizar finding creado en Defect Dojo....54
Figura 63 - Visualizar finding creado en la herramienta Defect Dojo....54
Figura 64 - Security Best Practices Patterns....55
Figura 65 – Best Security Practices de una aplicación....56
Figura 66 - Informe de resultados de la auditoría....57
Figura 67 - Arquitectura de la herramienta....60
Figura 68 - Esquema de la arquitectura de la aplicación....67
Figura 69 - Dashboard de inicio....73
Figura 70 - Menú de login....73
Figura 71 - Datos de registro de la aplicación....74
Figura 72 - Menú del usuario....74

CAPÍTULO 1. INTRODUCCIÓN

Durante el desarrollo del primer capítulo, hablaremos de la motivación de este trabajo, además, haremos una introducción al estado del arte en el sector de las auditorías estáticas para móviles y, teniendo en cuenta lo anterior, se hablará de los objetivos y una estructura básica de la memoria para ponernos en contexto durante la lectura de este trabajo.

1.1 Motivación

En primer lugar, se ha realizado un estudio y revisión de las herramientas *open source* disponibles de análisis de código en Android y, se ha llegado a la conclusón de que no permiten un desarrollo de una auditoría de manera completa. Aunque en muchos de los casos la información que recaban es bastante completa, se han encontrado defectos en cuanto a la categorización de estos datos, el modelo que disponen y, sobre todo, la imposible verificación de las evidencias que se encuentran, lo que hace tener un informe que no se corresponde con la realidad, puesto que muchas de las evidencias se tratan de falsos positivos, es decir, no suponen un problema de seguridad.

Por otro lado, no se han encontrado aplicaciones que verifiquen que se están siguiendo las mejores prácticas en seguridad, por lo que se incluido esa puesta en valor de la codificación que se realice de manera segura y, poniendo en evidencia donde no se realiza correctamente, mostrando esas vulnerabilidades o findings.

En último lugar, una de las observaciones principales en cuanto a las aplicaciones Android es que, debido a su popularidad y su facilidad para la modificación, la inclusión de malware en aplicaciones populares y su difusión con código malicioso es muy sencilla. Por lo tanto, se ha creido importante extender la auditoría de seguridad a una búsqueda de código malicioso para llevar a cabo una homologación de las aplicaciones, es decir, que sean verificadas para que no contengan ni vulnerabilidades ni malware para poder ser instaladas de manera segura.

1.2 Estado del Arte

Como hemos comentado en el anterior punto, actualmente existen aplicaciones que realizan auditorías de aplicaciones Android de manera bastante precisa y herramientas que permiten la búsqueda de malware en estas, pero ni herramientas open source ni en el mercado se ha encontrado una herramienta que permita ambas cosas, por lo que supone un desarrollo muy útil que pretende cubrir ese hueco en la comunidad de herramientas de seguridad.

1.3 Objetivos

Una vez comentado lo anterior, queda en evidencia el defecto encontrado en el resto de las herramientas y, el porqué de la creación de una herramienta que permita tanto una búsqueda de defectos, como un triaje de estos. Además, que nos permita la creación de un informe que se corresponda con la realidad de la aplicación auditada y nos permita verificar la seguridad de esta. Por último, como hemos comentado, además de en los defectos de seguridad, se ha puesto el foco en la búsqueda de posibles inclusiones de malware y la integración con diversas herramientas para agilizar el proceso y realizar una auditoría de manera completa.

1.4 Estructura de la memoria

La memoria está dividida en nueve capítulos, incluyendo este primero, en el que se hace una introducción al trabajo. En los sucesivos capítulos, se verá, en primer lugar, una sección sobre los fundamentos de Android, que incluye una introducción al sistema operativo, sus componentes, seguridad y las auditorías en esta plataforma. En segundo lugar, listaremos los posibles riesgos en estas aplicaciones junto a ejemplos sacados de la herramienta desarrollada. A partir de este punto, se verán las funcionalidades, cómo ha sido el desarrollo y el despliegue de esta. Por último, se verá la bibliografía y el contenido de la entrega.

CAPÍTULO 2. FUNDAMENTOS SOBRE ANDROID

En primer lugar, para comprender algunos de los elementos que se verán en la herramienta de auditorías, introduciremos los principales componentes del sistema operativo de Android, que fue creado por Google y es ampliamente utilizado, de hecho, tiene un 74,43 % de cuota de mercado de dispositivos móviles en el mundo y, en España llegando a ser de un 80,29 %. Este uso extendido del sistema operativo hace que, en numerosas ocasiones sea víctima de ataques por parte de usuarios maliciosos, por lo que es importante realizar auditorías a las aplicaciones para verificar la seguridad de estas.

2.1 Aplicaciones

A lo largo de este epígrafe, veremos los dos tipos de aplicaciones que podemos encontrar en Android: de sistema o de usuario. Las aplicaciones de sistema vienen preinstaladas en la propia imagen del sistema operativo (SO) y, están dentro de la partición */system* del disposivo. Por otro lado, las aplicaciones de usuario se denominan así porque son las que instala el propio usuario del dispositivo y, son las que se centrará nuestra investigación. Se enceuntran dentro del dispositivo en la partición */data* y pertenecen a una zona de seguridad aislada a la que otras aplicaciones no pueden acceder. Además, siguiendo el principio de mínimo privilegio, sólo pueden tener los permisos que se han explicitado para esta aplicación dentro del fichero *AndroidManifest.xml*, que es un fichero binario que contiene la información de esta.

Por último, estas aplicaciones una vez desarrolladas, se crea un paquete con toda la información de estas dentro de un fichero *.apk*, que permitirá su instalación en los dispositivos y además será el formato que soportará nuestra aplicación para la realización de las auditorías.

2.2 Componentes

Dentro de estas aplicaciones, se incluyen numerosos componentes que permiten la realización de diferentes acciones basándose en la interacción de los usuarios, además, de realizar eventos y lanzar notificaciones al usuario entre otras muchas cosas.

A continuación, se enumerarán los componentes básicos de las aplicaciones en Android que aparecerán en nuestra aplicación:

2.2.1 Activities

Una activity es una pantalla de la aplicación, que proporciona al usuario una interfaz para interactuar con esta. Por lo general, cada aplicación tiene múltiples activities, que permiten ejecutarse en cierto orden. Además, también pueden ejecutarse de manera independiente o incluso, en caso de que se permita, podrían ser accedidas por otras aplicaciones.

2.2.2 Services

Un service se trata de un componente que se ejecuta en segundo plano para realizar tareas y así evitar bloquear la interfaz de usuario que se encuentra en primer plano. Estas tareas pueden ser desde descargar algún archivo, reproducir música, capturar datos de la red etcétera.

2.2.3 Content Providers

Un content provider es un componente que gestiona los datos de una aplicación para que tengan persistencia. Esta se puede conseguir haciendo uso del sistema de ficheros del dispositivo, mediante bases de datos SQLite etcétera. Además, proporciona una API estándar que permite realizar transacciones mediante un esquema de URI que permite compartir datos con otras aplicaciones.

2.2.4 Broadcast Receivers

Es un componente que permite al dispositivo enviar eventos a la aplicación fuera del flujo del usuario, incluso aunque la aplicación no se esté ejecutando. Como, por ejemplo, una notificación sobre un evento, el envío de un recordatorio, el aviso de que una actualización se ha completado etcétera.

2.2.5 Intents

Un Intent se trata de un mensaje para requerir que un componente de la aplicación realice cierta acción. De los cuatro componentes comentados anterior, tres de ellos (activities, services y broadcast receivers), se activan mediante los mensajes asíncronos llamados Intent. Para los dos primeros, se define la acción a realizar y, puede espcificar los datos que requiere. Como, por ejemplo, enviar una solicitud para que se abra la cámara. Por otro lado, en cuanto a los broadcasts receivers, solo se especifica el broadcast a emitir, por ejemplo, de que el nivel de batería es bajo.

2.3 Seguridad

El sistema operativo Android está basado en Linux y posee numerosos sistemas de seguridad basados en capas, que no vamos a comentar a bajo nivel puesto que queda fuera del ámbito de este trabajo, lo que sí vamos a comentar son dos de las medidas de seguridad que existen, que tienen que ver con el sandboxing y los permisos de las aplicaciones:

2.3.1 Sandboxing

En primer lugar, en cuanto al sandboxing, podemos comentar que Android tiene un permisionado de ficheros de manera que las aplicaciones por defecto solo tienen acceso a los datos de las propias aplicaciones generan y, así permanecen aisladas unas de otras y no pueden interactuar entre ellas.

Por el contrario, para poder compartir datos entre ellas, deberán solicitarlo de manera explícita en el fichero *AndroidManifest.xml* de la aplicación, por lo que podremos verificar estos permisos en caso de que se produzcan.

Por otro lado, es posible evitar estos mecanismos de seguridad mediante el rooteo del dispositivo, lo que consiste en dar acceso root al dispositivo, lo que permitirá que este usuario pueda acceder a cualquier dato dentro del dispositivo. En nuestra aplicación, buscaremos que las aplicaciones verifiquen que no se realicen estas malas prácticas, para evitar posibles filtraciones de información sensible.

2.3.2 Permissions

Por otro lado, cuando las aplicaciones necesitan el acceso a ciertas funcionalidades, lo hacen a través de estos permisos en Android, por ejemplo, si necesita acceso a los contactos, requerirá este permiso dentro del archivo *AndroidManifest.xml.*

A continuación, veremos los permisos de las aplicaciones en cuanto a su clasificación:

Normal

Se tratan de permisos que no presentan riesgo para la privacidad o funcionamiento del dispositivo.

Dangerous

Por el contrario, se trata de permisos potencialmente peligrosos para la privacidad del usuario o que pudieran afectar al funcionamiento normal del dispositivo.

Signature

El sistema otorga estos permisos en el momento de la instalación cuando la aplicación que intenta usar ese permiso tiene la firma del mismo certificado que la que define el permiso, es decir, para que no sean usados por aplicaciones de terceros.

En nuestra aplicación, contaremos con un diccionario custom de permisos en Android clasificadas además de con estos criterios, por criticidad, según sean menos (*Low*) o más peligrosas (*High*).

2.4 Auditorías

Una vez visto lo anterior, vamos a identificar la metodología de auditorías que se va a emplear, con las siguientes fases.

En primer lugar, conocer los principales riesgos que tiene nuestra aplicación, para poder entender las diferentes amenazas y ser conscientes de las posibles debilidades y malware que nos podemos encontrar en el código de nuestras aplicaciones.

En segundo lugar, definir el alcance de esta auditoría: si va a ser estática, es decir, del código de la aplicación o si será dinámica, es decir, si va a requerir la ejecución de esta. En nuestro caso solo incluiremos la auditoría estática, debido a que se realizará un análisis de seguridad del código fuente de esta, sin ejecución.

Por otro lado, una vez definido el alcance, se procede a la iniciar la auditoría, que incluye una preparación inicial del entorno, para poder tener acceso al apk de la aplicación que se auditará.

Al tener disponible el apk, es decir, el ejecutable compilado, se procede a la extracción del código fuente de la aplicación para su posterior análisis en una fase de reversing.

Posteriormente, al tener el código de la aplicación decompilado, se realiza el análisis estático de esta para la obtención de toda la información posible, así como búsqueda de vulnerabilidades o malware en este. Una vez concluida la auditoría, se deberán verificar los findings encontrados de manera manual, en lo que se conoce como triaje, desechando los que sean falsos positivos.

Por último, se termina la auditoría con una fase de elaboración de resultados que incluye un informe final con las conclusiones e información extraída.

En los siguientes puntos, desarrollaremos este proceso y cómo hemos llevado a cabo nuestra herramienta automática de auditorías.

CAPÍTULO 3. PRINCIPALES RIESGOS DE SEGURIDAD EN ANDROID

Previo a la realización de una auditoría en Android, se deberán conocer los principales riesgos que podemos encontrarnos en estas aplicaciones. Para ello, se presenta el siguiente listado de los más importantes y, para verlas de manera práctica, y en cada una de ellas lo ilustraremos con parte de la información que podremos sacar de manera automática haciendo uso de la herramienta desarrollada, aunque será en posteriores capítulos donde desarrollaremos con un mayor detalle las distintas funcionalidades de las que se compone.

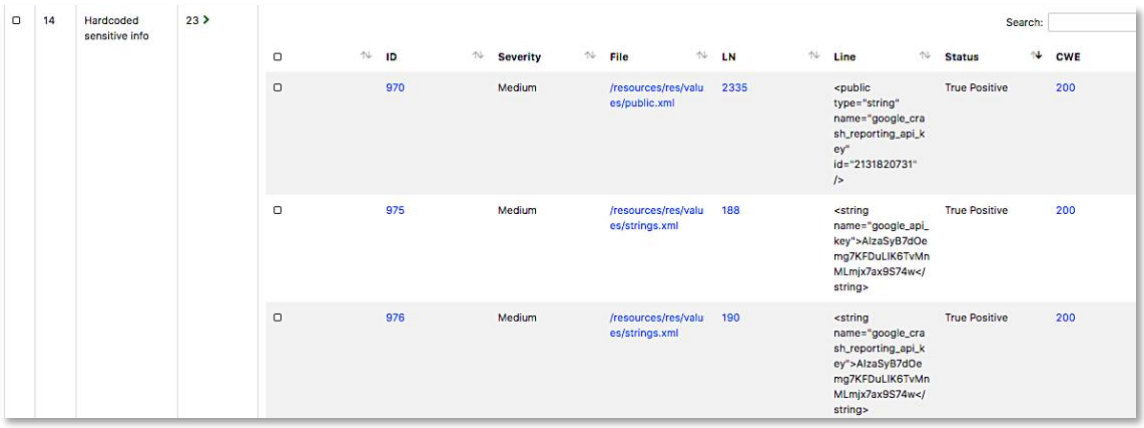
3.1 Autenticación insuficiente

Este riesgo se trata fundamentalmente de la no identificación del usuario, es decir del no aseguramiento de que el usuario es quien dice ser durante todo el uso que hace este de la aplicación.

Esto puede consistir desde la no identificación en absoluto del usuario, como el no mantenimiento de su identidad y/o debilidades en el manejo de sesiones.

Dentro del código, buscaremos posibles usuarios, contraseñas, API keys o access tokens que nos permitan autenticarnos dentro de la aplicación.

A continuación, veremos dentro de la aplicación un ejemplo de esas vulnerabilidades:



		Hardcoded sensitive info	23

Search:

ID	Severity	File	LN	Line	Status	CWE
970	Medium	/resources/res/values/public.xml	2335	<public type="string" name="google_crash_reporting_api_key" id="2131820731" />	True Positive	200
975	Medium	/resources/res/values/strings.xml	188	<string name="google_api_key">AIzaSyB7dOemg7KFDuLIK6TvMnMLmjx7ax9S74w</string>	True Positive	200
976	Medium	/resources/res/values/strings.xml	190	<string name="google_crash_reporting_api_key">AIzaSyB7dOemg7KFDuLIK6TvMnMLmjx7ax9S74w</string>	True Positive	200

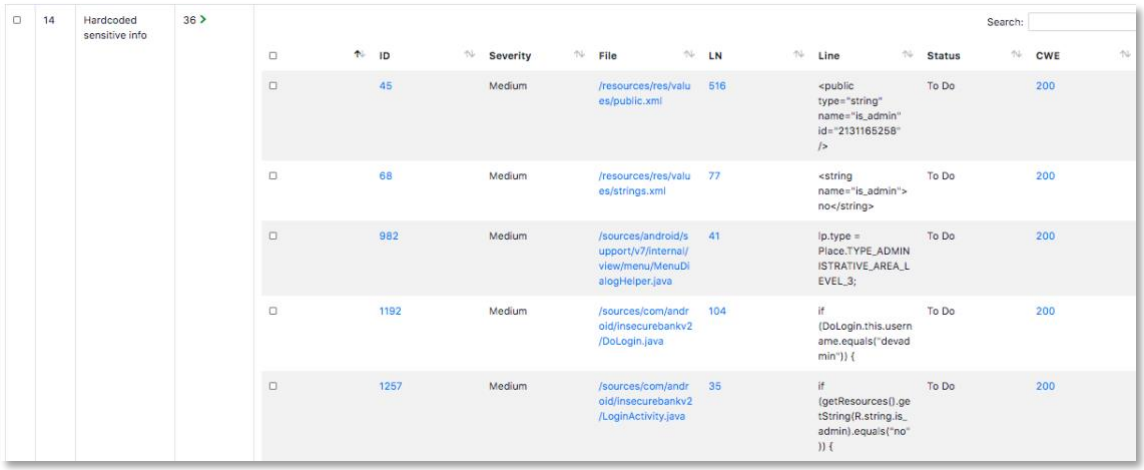
Figura 1 - Findings de Hardcoded API Keys

3.2 Autorización inadecuada

Este riesgo consiste en que no se asegure correctamente que el usuario tiene o no permisos para realizar cierta acción dentro de la aplicación, lo que podría resultar en que este realice acciones que no le corresponden.

3.2.1 Access control

Como hemos comentado, el control de acceso consiste en que no se validen correctamente los permisos que tiene el usuario dentro de la aplicación, haciendo uso de validaciones inseguras respecto a los roles del usuario. Por ejemplo, lo que podemos ver a continuación, en el finding con ID 68, en esta aplicación se verifican los permisos del usuario haciendo uso de una variable *is_admin* dentro de un fichero *strings.xml*:



		Hardcoded sensitive info	36

Search:

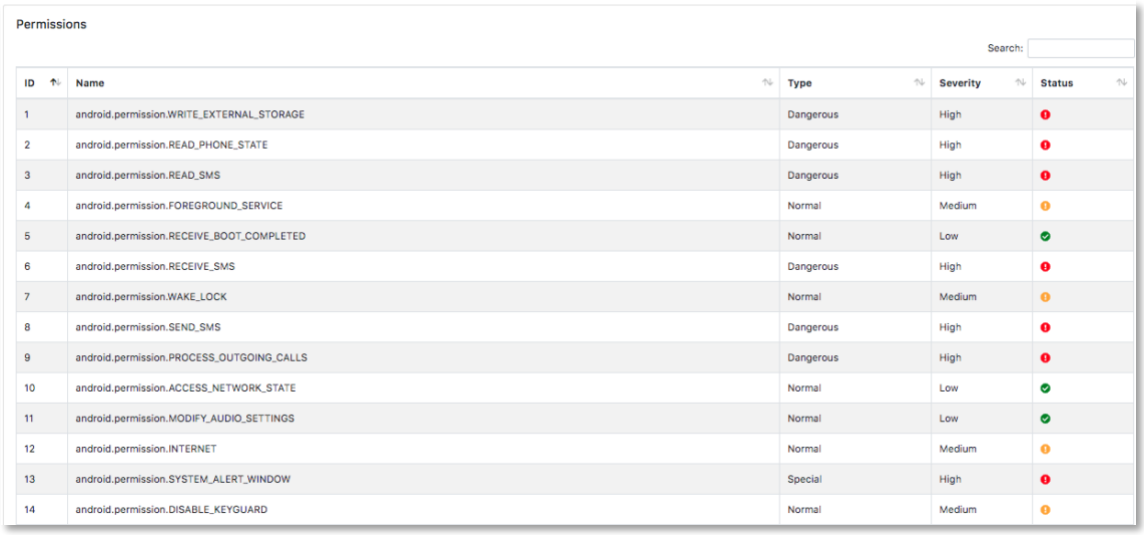
ID	Severity	File	LN	Line	Status	CWE
45	Medium	/resources/res/values/public.xml	516	<public type="string" name="is_admin" id="2131165258" />	To Do	200
68	Medium	/resources/res/values/strings.xml	77	<string name="is_admin">no</string>	To Do	200
982	Medium	/sources/android/support/v7/internal/view/menu/MenuDialogHelper.java	41	lp.type = Place.TYPE_ADMINISTRATIVE_AREA_LEVEL_3;	To Do	200
1192	Medium	/sources/com/android/insecurebankv2/DoLogin.java	104	if (DoLogin.this.username.equals("devadmin")) {	To Do	200
1257	Medium	/sources/com/android/insecurebankv2/LoginActivity.java	35	if (getResources().getString(R.string.is_admin).equals("no")) {	To Do	200

Figura 2 - Findings de autorización

3.2.2 Application permissions

Por otro lado, los permisos son esenciales en Android, estos permiten conocer qué va a poder realizar la aplicación, por lo que, para nuestra auditoría, listaremos los permisos que tiene la aplicación y los categorizaremos según criticidad (desde permisos normales, como puede ser el acceso a internet, como permisos más peligrosos como el acceso a contactos o envío de SMS), para verificar que la aplicación no tiene más permisos de los necesarios.

Por ejemplo, como podemos ver a continuación, la aplicación que aparentemente es una literna, tiene permisos de envío y recepción de SMS, internet etcétera, lo cual es bastante sospechoso:



Permissions

Search:

ID	Name	Type	Severity	Status
1	android.permission.WRITE_EXTERNAL_STORAGE	Dangerous	High	
2	android.permission.READ_PHONE_STATE	Dangerous	High	
3	android.permission.READ_SMS	Dangerous	High	
4	android.permission.FOREGROUND_SERVICE	Normal	Medium	
5	android.permission.RECEIVE_BOOT_COMPLETED	Normal	Low	
6	android.permission.RECEIVE_SMS	Dangerous	High	
7	android.permission.WAKE_LOCK	Normal	Medium	
8	android.permission.SEND_SMS	Dangerous	High	
9	android.permission.PROCESS_OUTGOING_CALLS	Dangerous	High	
10	android.permission.ACCESS_NETWORK_STATE	Normal	Low	
11	android.permission.MODIFY_AUDIO_SETTINGS	Normal	Low	
12	android.permission.INTERNET	Normal	Medium	
13	android.permission.SYSTEM_ALERT_WINDOW	Special	High	
14	android.permission.DISABLE_KEYGUARD	Normal	Medium	

Figura 3 - Ejemplo de application permissions

Por último, destacaremos algunas de las malas prácticas que se realizan en el *AndroidManifest.xml*, como pueden ser las siguientes:

Component export

Si realizamos un export de un componente (*android:exported="true"*), cualquier aplicación puede tener acceso a este, lo que podría llevar a comportamiento no intencionado o inyecciones.

Un ejemplo de esta vulnerabilidad la podemos observar en el siguiente finding:

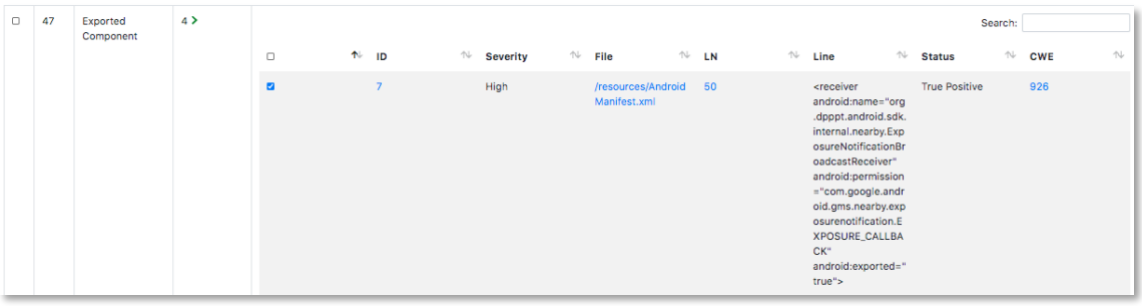


Figura 4 - Finding de Exported Component

Backup de los componentes

En caso de que en un componente esté habilitada la opción *android:allowBackup="true"*, cualquier usuario con permiso de USB debugging podrá acceder a datos de la aplicación, lo cual podría exponer información sensible de esta.

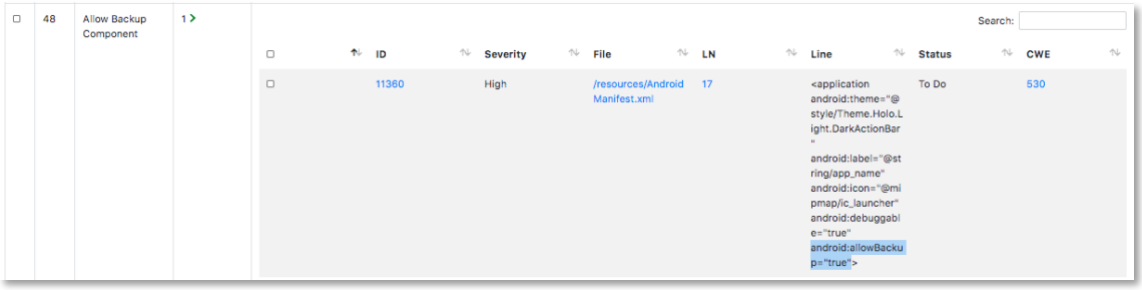


Figura 5 - Finding de component backup

Debuggable components

Si está habilitada la opción *android:debuggable="true",* permitiría a usuarios sin permisos root acceder a información de la aplicación y ejecutar código arbitrario haciendo uso de los permisos de esta.

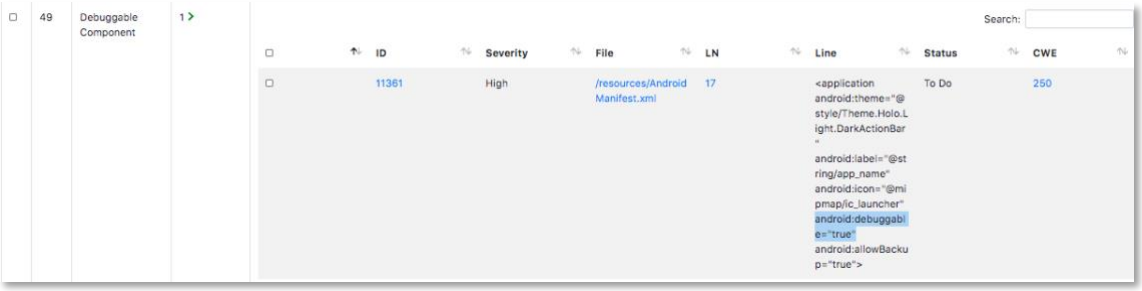


Figura 6 - Finding de debuggable component

3.3 Comunicación insegura

Este riesgo tiene que ver con la integridad, confidencialidad y autenticación de las comunicaciones, clave dentro de una comunicación que solo queremos que el emisor y receptor (usuario-aplicación y backend) conozcan el contenido de los mensajes intercambiados y, así evitar ataques de Man-in-the-Middle. Para ello, siempre la comunicación debería ser mediante TLS y hacer uso de verificación de certificados o SSL Pinning.

3.3.1 Acceso a URLs sin SSL/TLS

Uno de los posibles riesgos es el acceso a URLs, IPs etcéra sin hacer uso de TLS, por lo que se buscarán conexiones de este tipo a lo largo de todo el código.

Un ejemplo de vulnerabilidades de este tipo serían las siguientes:

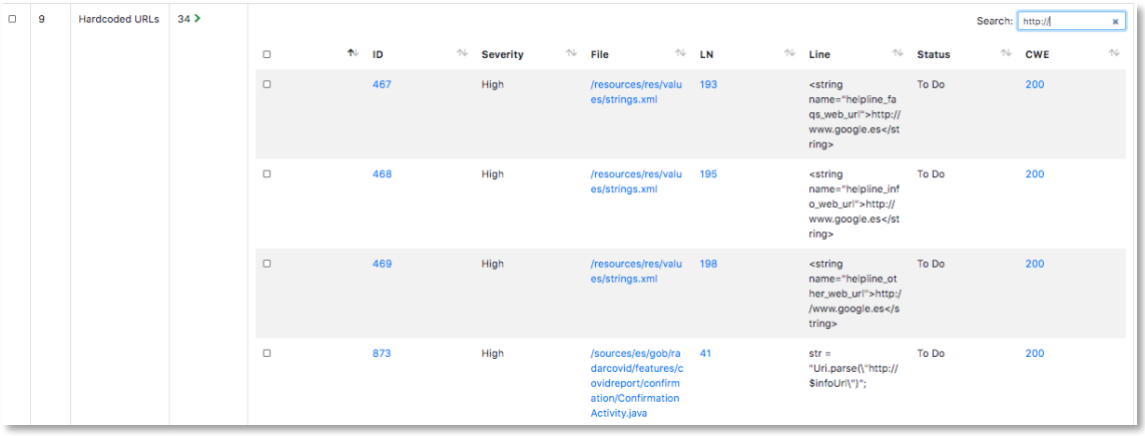


Figura 7 - Acceso a URLs sin TLS

3.3.2 Certificados

Para verificar las conexiones, la aplicación deberá hacer uso de certificados. Eso sí, hay que tener en cuenta que algunos métodos de verificación contienen debilidades, como puede ser el siguiente:

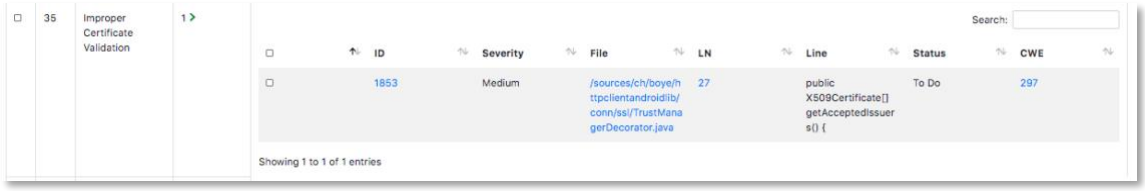


Figura 8 - Imporper certificate validation

Por otro lado, para una mejora de la seguridad, en todas las aplicaciones se debería implementar una verificación de toda la cadena de confianza de certificados, haciendo uso de lo que se llama SSL Pinning. Dentro de la aplicación, buscaremos mejores prácticas en este aspecto buscando los métodos que realizan esta verificación:

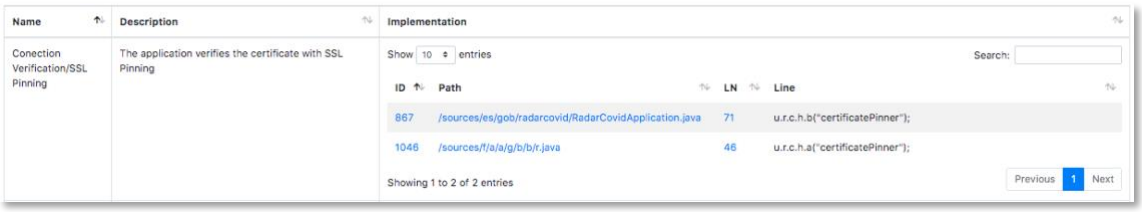


Figura 9 - Ejemplos de SSL Pinning

3.4 Guardado inseguro de información

El guardado de información en las aplicaciones se puede realizar de diversas maneras, por lo tanto, deberemos centrarnos en todas ellas para así buscar posibles datos sensibles que no se estén procesando correctamente. De este modo, se identificarán los siguientes elementos dentro de la aplicación:

3.4.1 Weak cryptography

La aplicación buscará los métodos criptográficos utilizados y verá cuáles utilizan algoritmos no robustos o funciones hash que tienen colisiones. Un ejemplo de vulnerabilidades de este tipo son las siguientes:

Insecure Random Number

Se trata de usar una clase que genera números aleatorios que no es criptograficamente segura, por lo que pondría en peligro la aleatoriedad de estos.

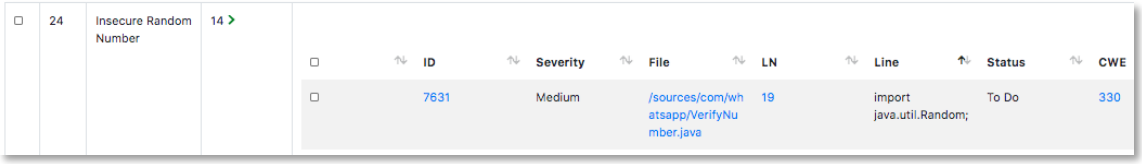
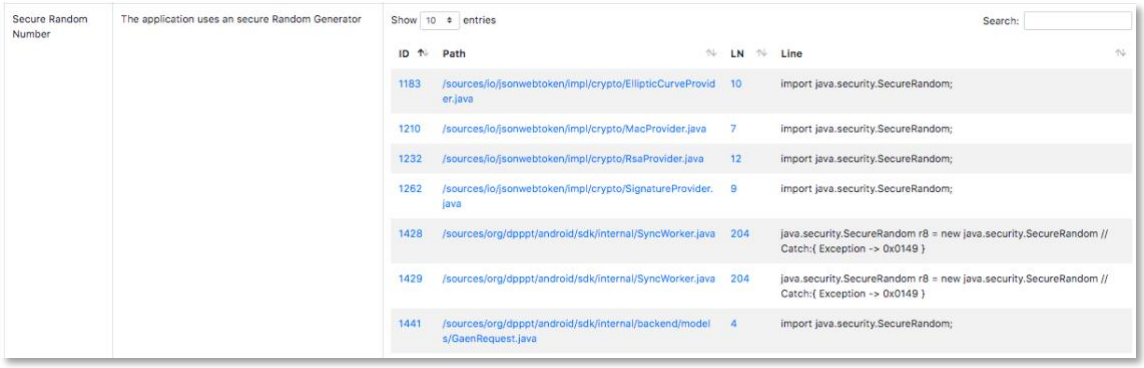


Figura 10 - Findings de Insecure Random Number

Por el contrario, en caso de que se utilicen números aleatorios seguros, se valorarán estas mejores prácticas:



Secure Random Number	The application uses an secure Random Generator

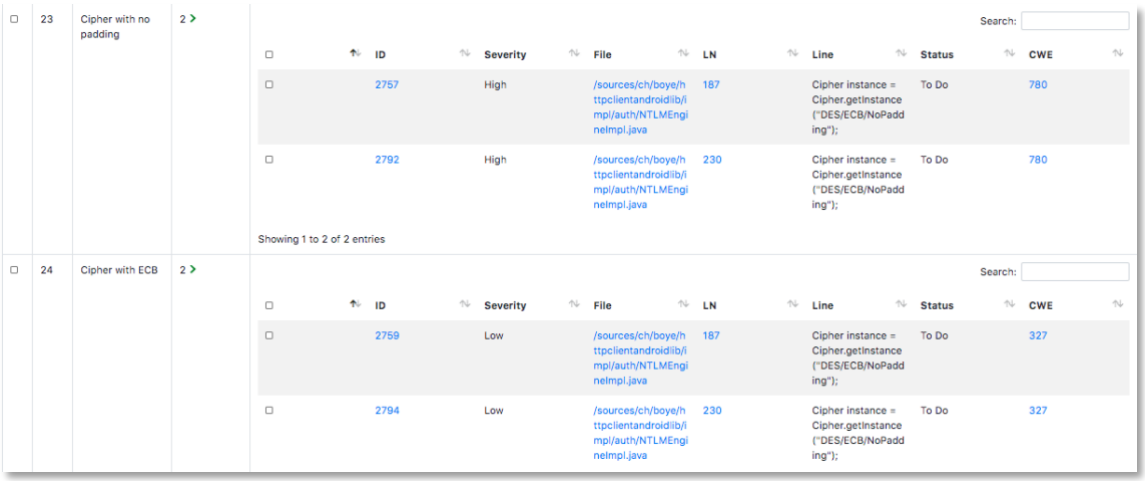
Show 10 entries Search:

ID	Path	LN	Line
1183	/sources/io/jsonwebtoken/impl/crypto/EllipticCurveProvider.java	10	import java.security.SecureRandom;
1210	/sources/io/jsonwebtoken/impl/crypto/MacProvider.java	7	import java.security.SecureRandom;
1232	/sources/io/jsonwebtoken/impl/crypto/RsaProvider.java	12	import java.security.SecureRandom;
1262	/sources/io/jsonwebtoken/impl/crypto/SignatureProvider.java	9	import java.security.SecureRandom;
1428	/sources/org/dpppt/android/sdk/internal/SyncWorker.java	204	java.security.SecureRandom r8 = new java.security.SecureRandom // Catch:{ Exception -> 0x0149 }
1429	/sources/org/dpppt/android/sdk/internal/SyncWorker.java	204	java.security.SecureRandom r8 = new java.security.SecureRandom // Catch:{ Exception -> 0x0149 }
1441	/sources/org/dpppt/android/sdk/internal/backend/models/GaenRequest.java	4	import java.security.SecureRandom;

Figura 11 - Ejemplos de Secure Random Number

Weak Cryptography

Esta vulnerabilidad consiste en usar un protocolo de cifrado que no sea seguro y, por lo tanto, un atacante, podría romper el cifrado y acceder a información sensible. Algunos de los algoritmos no seguros son DES o cualquier algoritmo de cifrado propio. Además, también busca posibles usos de tipos de bloque o la no inclusión de padding etcétera.



23	Cipher with no padding	2 >	

Search:

ID	Severity	File	LN	Line	Status	CWE
2757	High	/sources/ch/boye/httpclientandroidlib/impl/auth/NTLMEngineImpl.java	187	Cipher instance = Cipher.getInstance ("DES/ECB/NoPadding");	To Do	780
2792	High	/sources/ch/boye/httpclientandroidlib/impl/auth/NTLMEngineImpl.java	230	Cipher instance = Cipher.getInstance ("DES/ECB/NoPadding");	To Do	780

Showing 1 to 2 of 2 entries

24	Cipher with ECB	2 >	

Search:

ID	Severity	File	LN	Line	Status	CWE
2759	Low	/sources/ch/boye/httpclientandroidlib/impl/auth/NTLMEngineImpl.java	187	Cipher instance = Cipher.getInstance ("DES/ECB/NoPadding");	To Do	327
2794	Low	/sources/ch/boye/httpclientandroidlib/impl/auth/NTLMEngineImpl.java	230	Cipher instance = Cipher.getInstance ("DES/ECB/NoPadding");	To Do	327

Figura 12 - Findings de Weak Cryptography

Weak Hash algorithm

La aplicación utiliza un algoritmo de hashing que no es seguro, por lo que puede tener colisiones y, por lo tanto, no ser viable para garantizar la integridad de la información, como pueden ser MD5 y SHA-1 entre otros. A continuación, podemos ver un ejemplo de estas vulnerabilidades que encuentra nuestra aplicación:

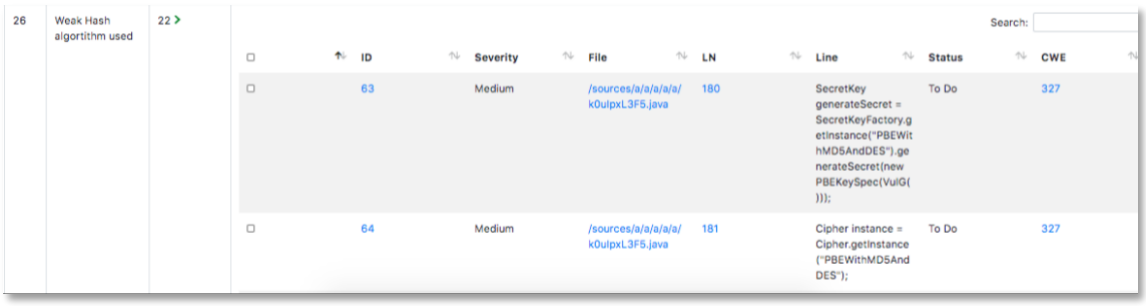


Figura 13 - Findings de Weak Hash algorithm

3.4.2 SQL databases

Una vez recorridos los ficheros de la apk, se buscarán los ficheros de base de datos y se tratará de sacar la información que contengan.

Un ejemplo es el siguiente, en el que se ha encontrado un fichero de base de datos con información de sesiones:

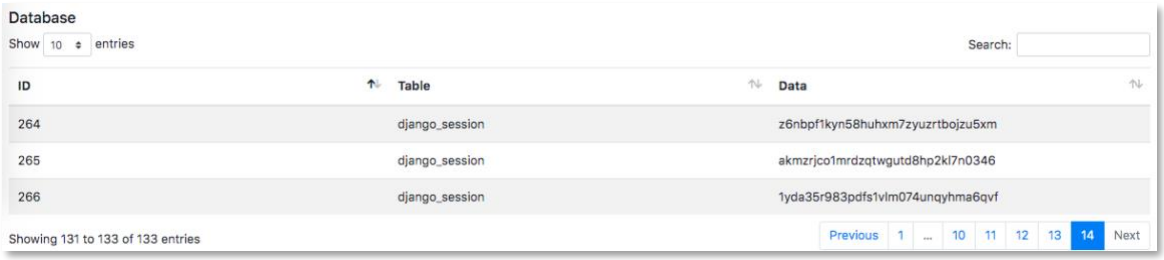


Figura 14 - Ejemplo de extracción de información de una BD

3.4.3 Storage

Este riesgo está relacionado con las operaciones que la aplicación realiza sobre ficheros, y pueden ser de dos tipos:

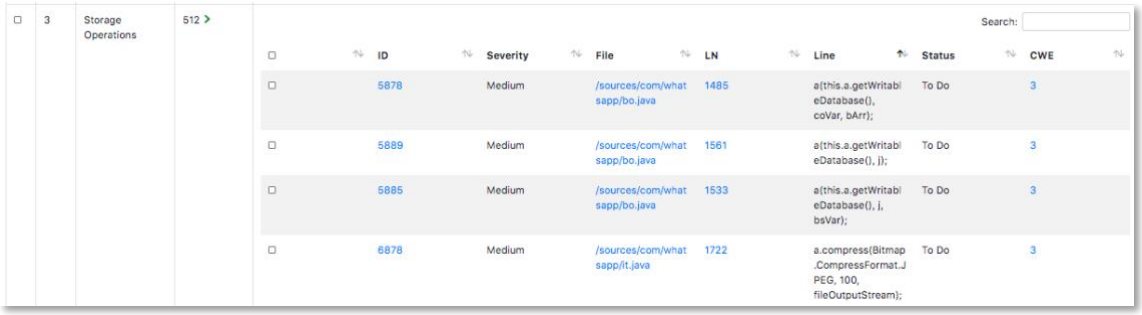
Internal

La aplicación utiliza el almacenamiento que dispone cada una de las aplicaciones de manera privada dentro del dispositivo, aunque igualmente, deberíamos verificar que no se guarda información sensible que pueda ser accedida por un usuario malintencionado.

External (sd-card)

La aplicación utiliza el almacenamiento externo, es decir, público, por lo que podrá ser accedido por cualquier aplicación o actor, por lo tanto, deberemos verificar que no se realiza ninguna exposición de información sensible de este modo.

Un ejemplo de estas vulnerabilidades podría ser un uso de operaciones de fichero que pudieran ser peligrosas:



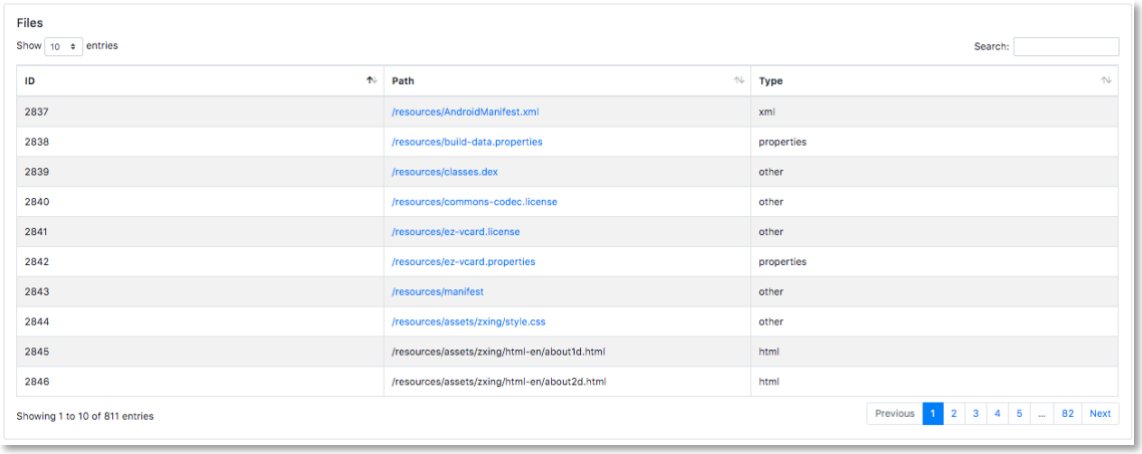
☐	3	Storage Operations	512 >						

Search:

	ID	Severity	File	LN	Line	Status	CWE
☐	5878	Medium	/sources/com/whatsapp/bo.java	1485	a(this.a.getWritableDatabase(), coVar, bArr);	To Do	3
☐	5889	Medium	/sources/com/whatsapp/bo.java	1561	a(this.a.getWritableDatabase(), j);	To Do	3
☐	5885	Medium	/sources/com/whatsapp/bo.java	1533	a(this.a.getWritableDatabase(), j, bsVar);	To Do	3
☐	6878	Medium	/sources/com/whatsapp/it.java	1722	a.compress(Bitmap.CompressFormat.JPEG, 100, fileOutputStream);	To Do	3

Figura 15 - Findings de operaciones con ficheros

Además, realizamos una búsqueda y guardamos los ficheros que podrían ser relevantes o pudieran contener infomación sensible para mostrarlos en el escaneo.



Files

Show 10 entries Search:

ID	Path	Type
2837	/resources/AndroidManifest.xml	xml
2838	/resources/build-data.properties	properties
2839	/resources/classes.dex	other
2840	/resources/commons-codec.license	other
2841	/resources/ez-vcard.license	other
2842	/resources/ez-vcard.properties	properties
2843	/resources/manifest	other
2844	/resources/assets/zxing/style.css	other
2845	/resources/assets/zxing/html-en/about1d.html	html
2846	/resources/assets/zxing/html-en/about2d.html	html

Showing 1 to 10 of 811 entries Previous 1 2 3 4 5 ... 82 Next

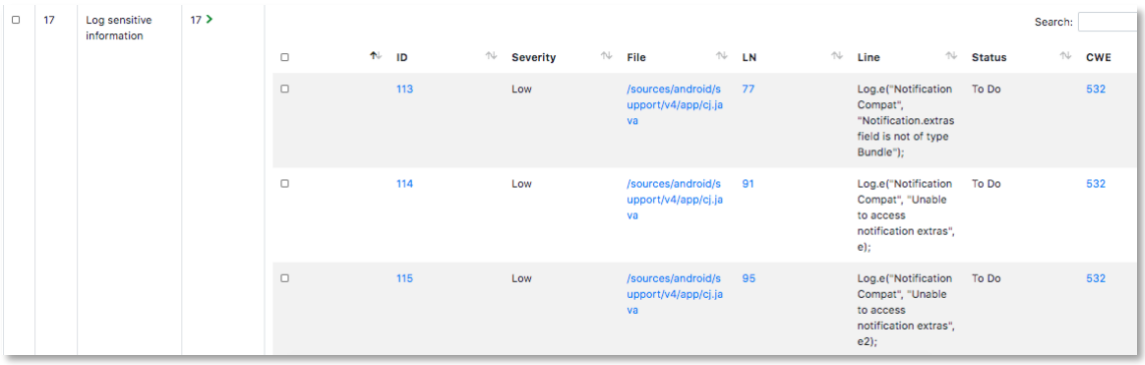
Figura 16 - Ficheros relevantes en escaneo

3.5 Sensitive Information leak

Con bastante relación al anterior riesgo, se realizará una búsqueda por toda la aplicación de posible información sensible hardcodeada en el código fuente de la aplicación.

3.5.1 Sensitive information in log

El riesgo consiste en el guardado de información potencialmente sensible en los logs, por lo que mostraremos como posible finding cualquier escritura en estos, aunque deberá verificarse posteriormente que la información sea o no sensible.



☐	17	Log sensitive information	17 >

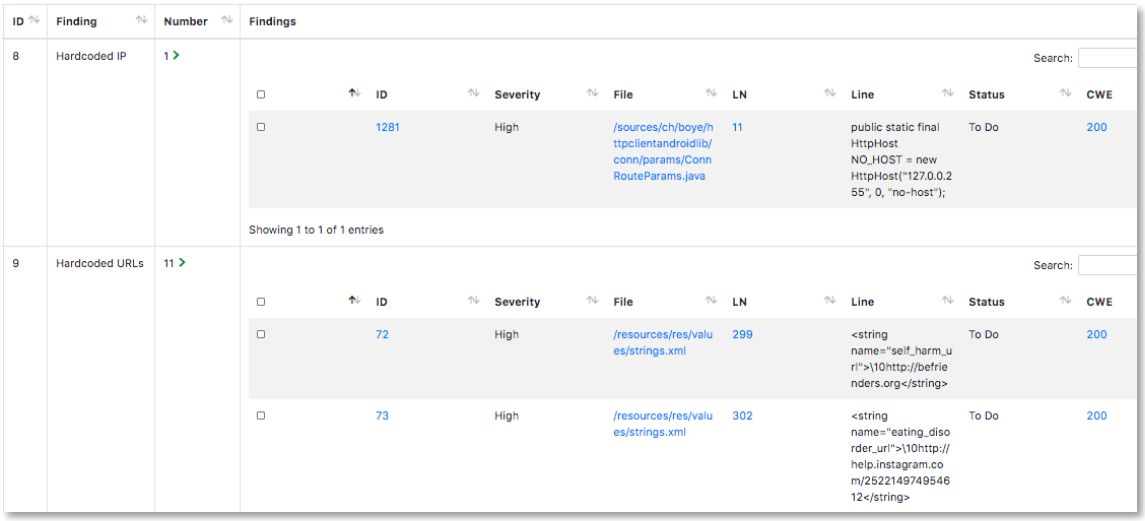
Search:

☐	ID	Severity	File	LN	Line	Status	CWE
☐	113	Low	/sources/android/support/v4/app/cj.java	77	Log.e("Notification Compat", "Notification.extras field is not of type Bundle");	To Do	532
☐	114	Low	/sources/android/support/v4/app/cj.java	91	Log.e("Notification Compat", "Unable to access notification extras", e);	To Do	532
☐	115	Low	/sources/android/support/v4/app/cj.java	95	Log.e("Notification Compat", "Unable to access notification extras", e2);	To Do	532

Figura 17 - Ejemplo de log sensitive information

3.5.2 Hardcoded URLs o IPs

Este riesgo se trata de encontrar URLs o IPs hardcodeadas en el código, lo que podría exponer información sobre la infraestructura con la que se comunica la aplicación.



ID	Finding	Number	Findings
8	Hardcoded IP	1 >	

Search:

☐	ID	Severity	File	LN	Line	Status	CWE
☐	1281	High	/sources/ch/boye/httpclientandroidlib/conn/params/ConnRouteParams.java	11	public static final HttpHost NO_HOST = new HttpHost("127.0.0.255", 0, "no-host");	To Do	200

Showing 1 to 1 of 1 entries

9	Hardcoded URLs	11 >	

Search:

☐	ID	Severity	File	LN	Line	Status	CWE
☐	72	High	/resources/res/values/strings.xml	299	<string name="self_harm_url">\10http://befrienders.org</string>	To Do	200
☐	73	High	/resources/res/values/strings.xml	302	<string name="eating_disorder_url">\10http://help.instagram.com/252214974954612</string>	To Do	200

Figura 18 - Ejemplo de Hardcoded URLs o IPs

3.5.3 Hardcoded keys

En este caso, se buscan posibes usuarios, claves dentro del código, API Keys, contraseñas etcétera, que nos permitan hacer un *bypass* de autenticación en alguno de los sistemas de nuestra aplicación.

Search:

ID	Severity	File	LN	Line	Status	CWE
199	Medium	/resources/res/values/public.xml	2147	<public type="raw" name="sedia_rsa_private_key" id="2131755009" />	To Do	200
200	Medium	/resources/res/values/public.xml	2333	<public type="string" name="google_api_key" id="2131820729" />	To Do	200
202	Medium	/resources/res/values/public.xml	2335	<public type="string" name="google_crash_reporting_api_key" id="2131820731" />	To Do	200

Figura 19 - Ejemplo de hardcoded keys

3.6 Reverse Engineering

Otro de los riesgos en las aplicaciones, se trata de la ingeniería inversa, es decir, que un posible usuario malicioso decompile la aplicación y, conociendo el flujo de esta, pueda modificarlo.

Para evitarlo, identificaremos métodos de evasión de rooting, debugging y/o emuladores como buenas prácticas de seguridad.

3.6.1 Root Detection

En este caso, se buscará a lo largo de la aplicación posibles métodos de detección de rooting en el teléfono o búsqueda de información sobre si el dispositivo tiene acceso root etcétera:

Root detection

The application checks if the device has been rooted

Show 10 entries

Search:

ID	Path	LN	Line
400	/sources/shin2/rootdetector/b.java	25	Process exec = Runtime.getRuntime().exec("su");
402	/sources/shin2/rootdetector/b.java	38	Process exec2 = Runtime.getRuntime().exec("su");
405	/sources/shin2/rootdetector/b.java	57	Runtime.getRuntime().exec("su").destroy();
417	/sources/shin2/rootdetector/c.java	43	String readLine = new BufferedReader(new InputStreamReader(Runtime.getRuntime().exec("su -v").getInputStream())).readLine();
412	/sources/shin2/rootdetector/c.java	20	String[] strArr = {"eu.chainfire.supersu", "eu.chainfire.supersu.pro", "com.koushikdutta.superuser", "com.noshufou.android.su"};

Figura 20 - Ejemplo de Root detection

3.6.2 Debugger Detection

Por último, tambiñen se verificará si hay métodos que eviten el uso de debuggers para evitar la modificación del flujo del código en la aplicación, dado que podría ser peligroso:

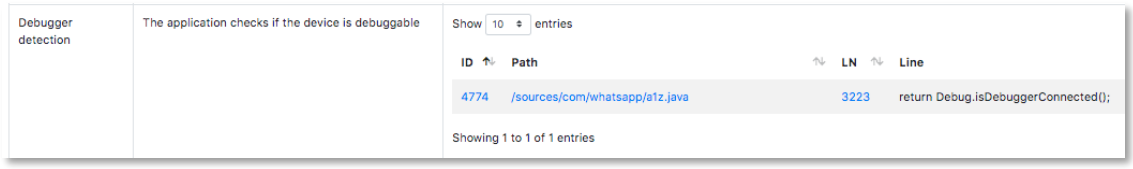


Figura 21 - Ejemplo de debugger detection

CAPÍTULO 4. HERRAMIENTA DE AUDITORÍAS ESTÁTICAS

Como hemos comentado anteriormente, la herramienta realiza auditorías estáticas completas de seguridad en los dispositivos Android: desde el reversing del código hasta la presentación de resultados, haciendo énfasis en la facilidad para el auditor de la realización del triaje de falsos positivos. Durante el desarrollo del siguiente punto, veremos las distintas funcionalidades de las que se compone para llevar a cabo estas auditorías:

4.1 Reglas o Patterns

En primer lugar, la base de la aplicación se compone de una serie de reglas o patterns que realizarán las búsquedas mediante expresiones regulares dentro del código de la aplicación. Estas serán mostradas en un dashboard en el que se podrán activar o desactivar dependiendo de las necesidades. Así, el auditor podrá desactivar las que considere que crea muchos falsos positivos o que no son relevantes en su investigación.

Para acceder a estas se puede hacer en la ruta */patterns* o accediendo mediante el menú haciendo clic sobre el desplegable *Others* como muestra la siguiente imagen.

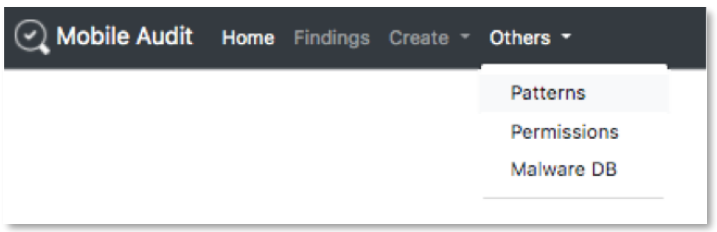


Figura 22 - Menú para acceder a las reglas/patterns

Por otro lado, una vez dentro, veremos la tabla con las diferentes reglas. Estas estarán organizadas por diversas opciones: criticidades, que indican la gravedad de la vulnerabilidad, siendo *Critical* la más alta y *Low* la más baja (en caso de ser una buena práctica se le asignaría la criticidad *None*), descripciones, mitigaciones de cada una de ellas, el *CWE* asociado y el estado (si están activadas o no). Podemos ver un ejemplo de ello en la siguiente figura:

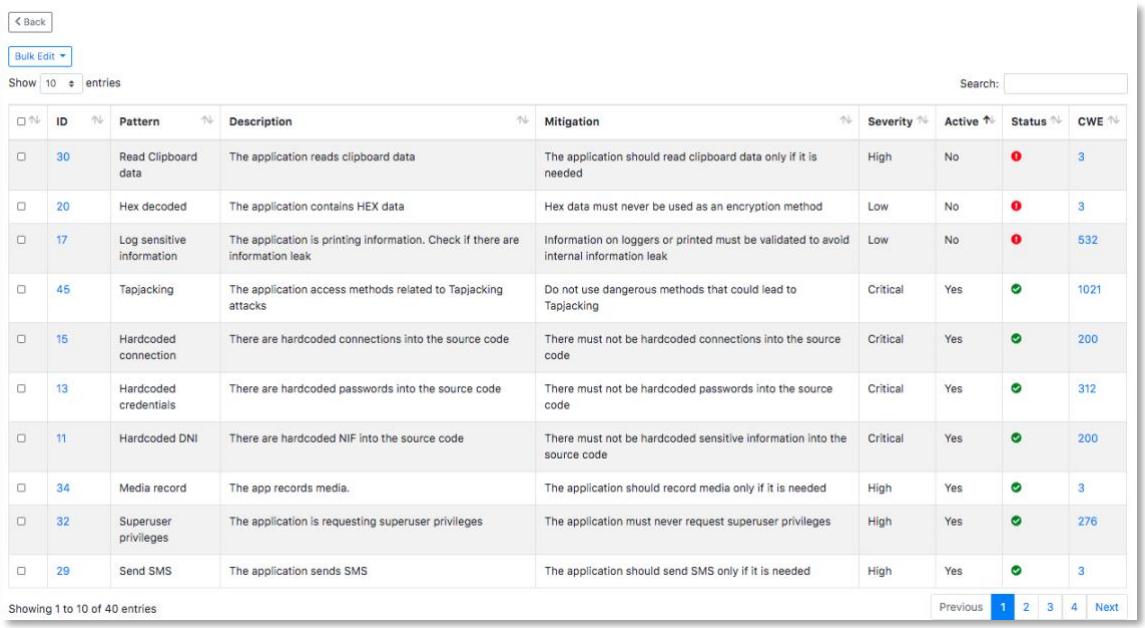


Figura 23 - Lista de reglas/patterns

Para editar el estado de las reglas, podemos seleccionar los *checkbox* de cada una de las reglas que queramos, seleccionar el menú de *Bulk Edit* con el estado al que queremos cambiar y dar a *Edit Patterns*.

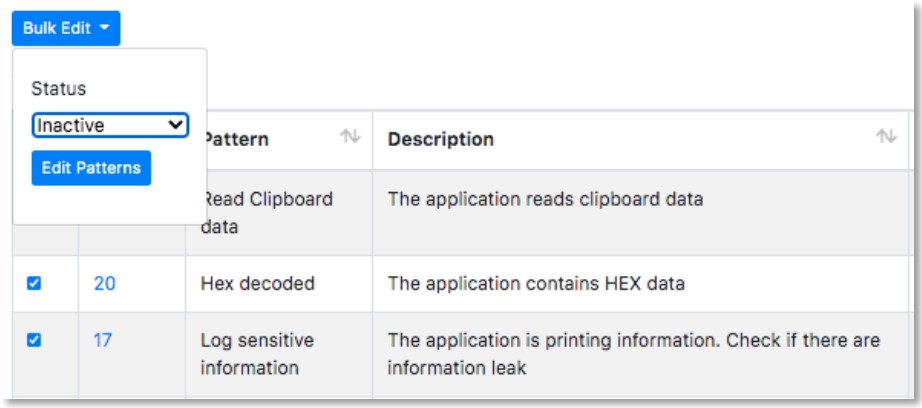


Figura 24 - Editar el estado de las patterns

Una vez editadas, se podrá ver cómo han cambiado el estado en las reglas. Podemos observarlo en la imagen a continuación, donde hemos desactivado las siguientes reglas:

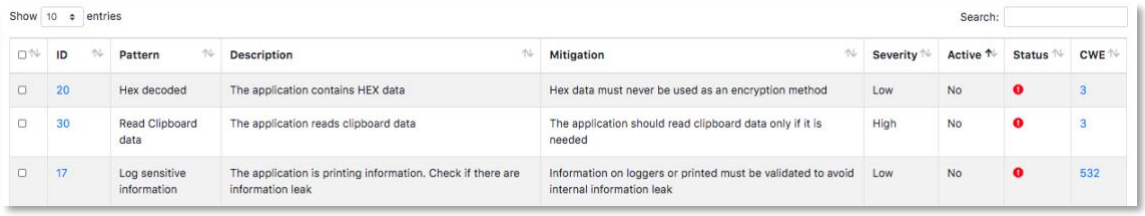


Figura 25 - Cambio de estado en las reglas

4.2 Creación de la auditoría

Por otro lado, una vez activas las reglas que nos interesen, para comenzar la auditoría, se procederá a crear una aplicación en la plataforma, para ello, hay dos opciones:

- Desde el menú principal, se hará clic sobre *New App*

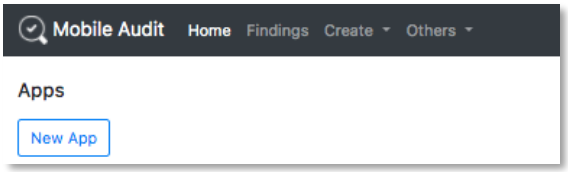


Figura 26 - Botón de creación de una aplicación

- Desde el menú de creación, haciendo clic en *Application*

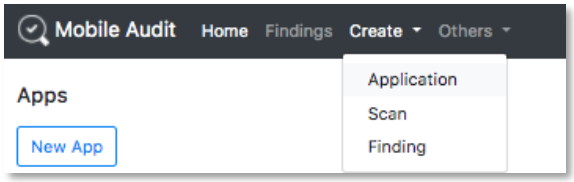


Figura 27 - Menú de Creación

Esta aplicación representará nuestro producto a analizar. En nuestro caso, utilizaremos la aplicación Radar Covid para realizar la auditoría de seguridad y, para ello, en primer lugar, rellenaremos los datos de nombre y descripción:

Figura 28 - Creación de la aplicación a auditar

Una vez realizado, procederemos a crear el escaneo de una versión concreta de nuestro producto mediante la subida de la apk correspondiente. Además, en caso de que esté activada la opción de integración con Defect Dojo, se podrá incluir el ID del Test al cual se subirán los findings de esta auditoría.

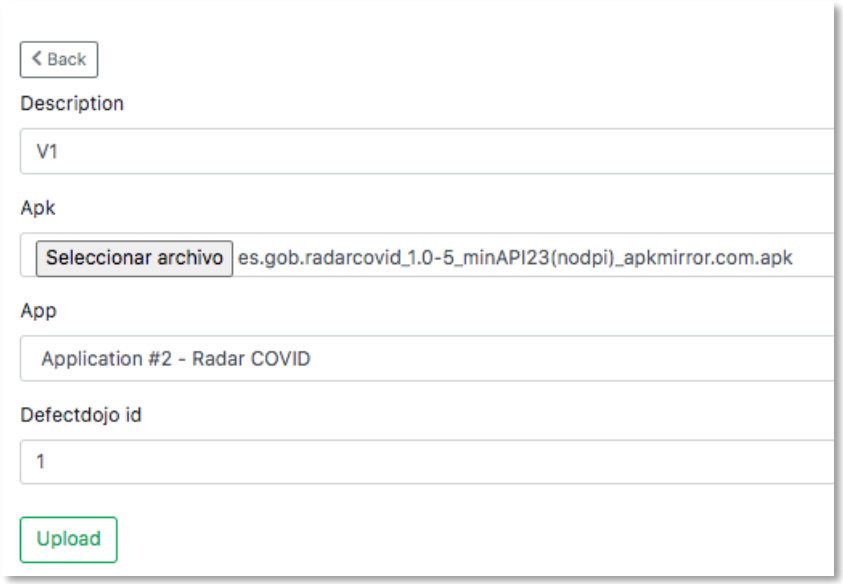


Figura 29 - Creación de un escaneo de la aplicación a auditar

En el dashboard de la aplicación web (*/* ó */home*), se podrá observar el estado de los diferentes productos junto a sus análisis. Además, se incluirá información de progreso, número de findings por categorías, detecciones con Virus Total (si está habilitado) etcétera.

Por otro lado, también se podrá crear nuevas aplicaciones y escaneos en estas:

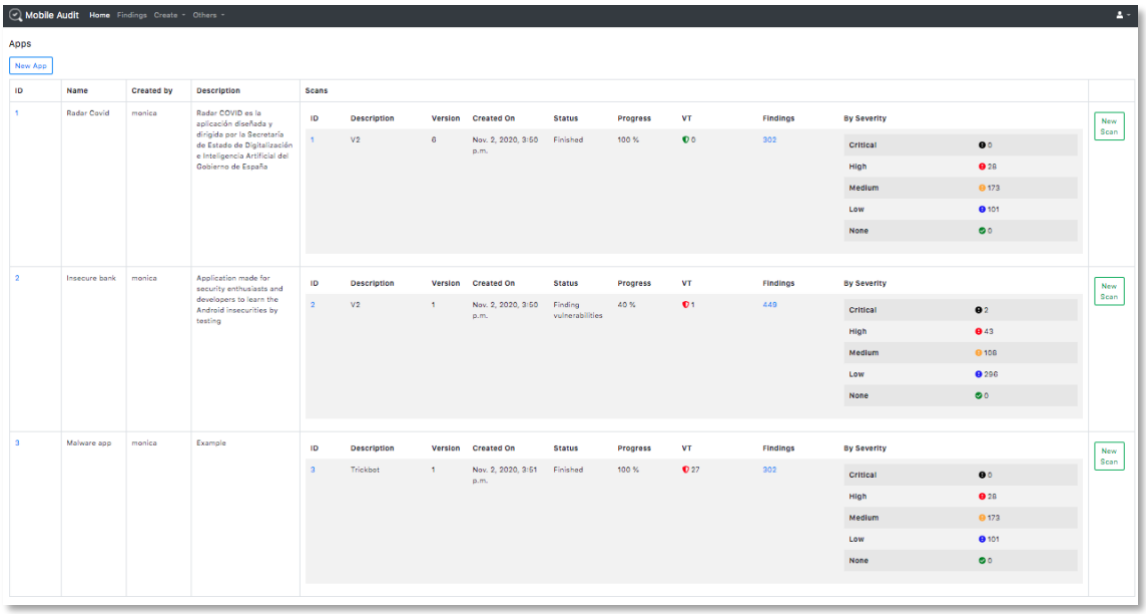


Figura 30 - Dashboard principal de la herramienta

Por otro lado, mientras se va realizando la auditoría, se irá observando la información que se va recopilando y, para acceder a la información extraída en cada escaneo desde el dashboard, haremos clic en el link del identificador del este.

Por último, dando clic al identificador de cada aplicación, podemos acceder al dashboard propio de la aplicación, que nos mostrará información similar pero sólo de la aplicación correspondiente.

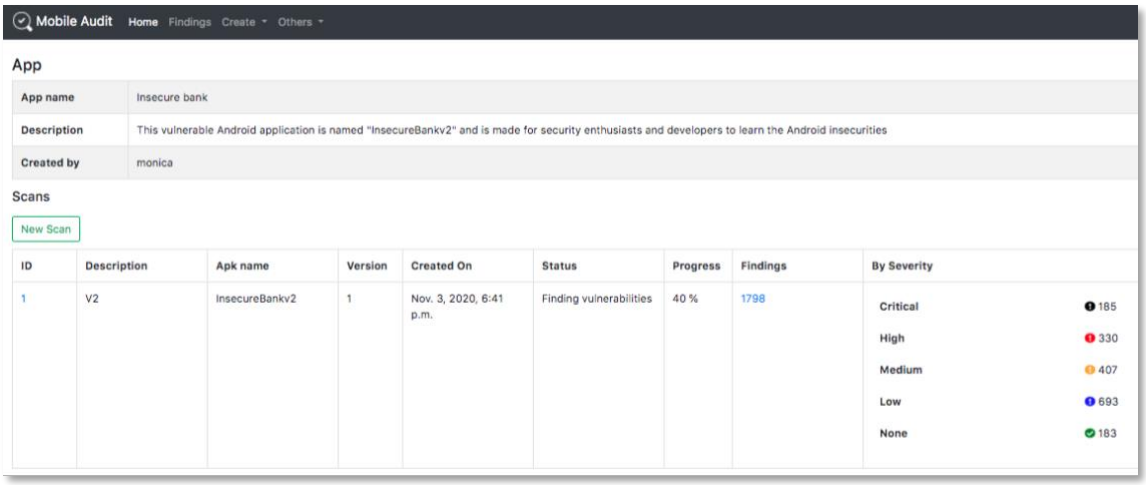


Figura 31 - Dashboard de una aplicación

4.3 Información recopilada durante el escaneo

En primer lugar, una vez estemos en el scan, podremos ir actualizando la página conforme se vaya recopilando nueva información para poder visualizarla haciendo uso del botón de *Refresh*.

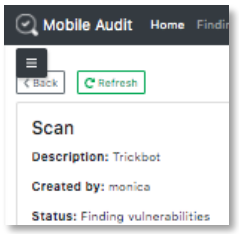


Figura 32 - Botón de refresh del scan

Además, para acceder de una manera más sencilla a toda la información que se describa a continuación, tenemos un menú lateral que se despliega haciendo *clic* en el icono 

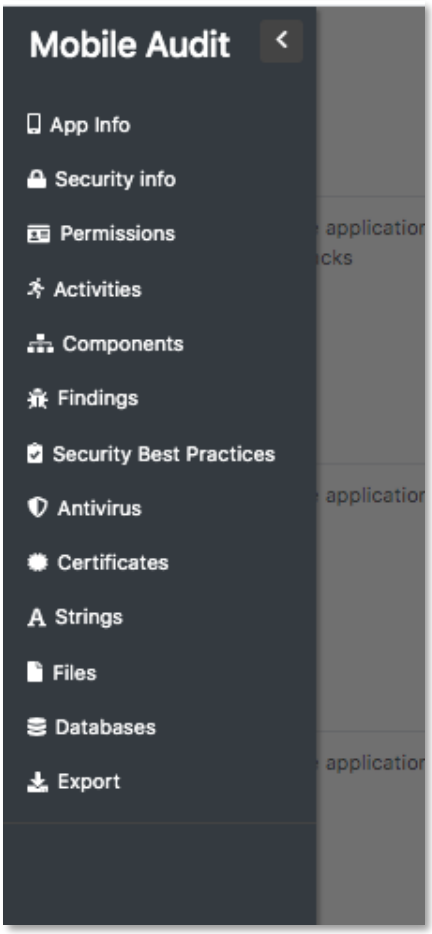


Figura 33 - Menú lateral con información del escaneo

4.3.1 Información de la aplicación

En este apartado, se incluirá información básica de la aplicación, que incluirá el estado del scan e información obtenida de la apk:

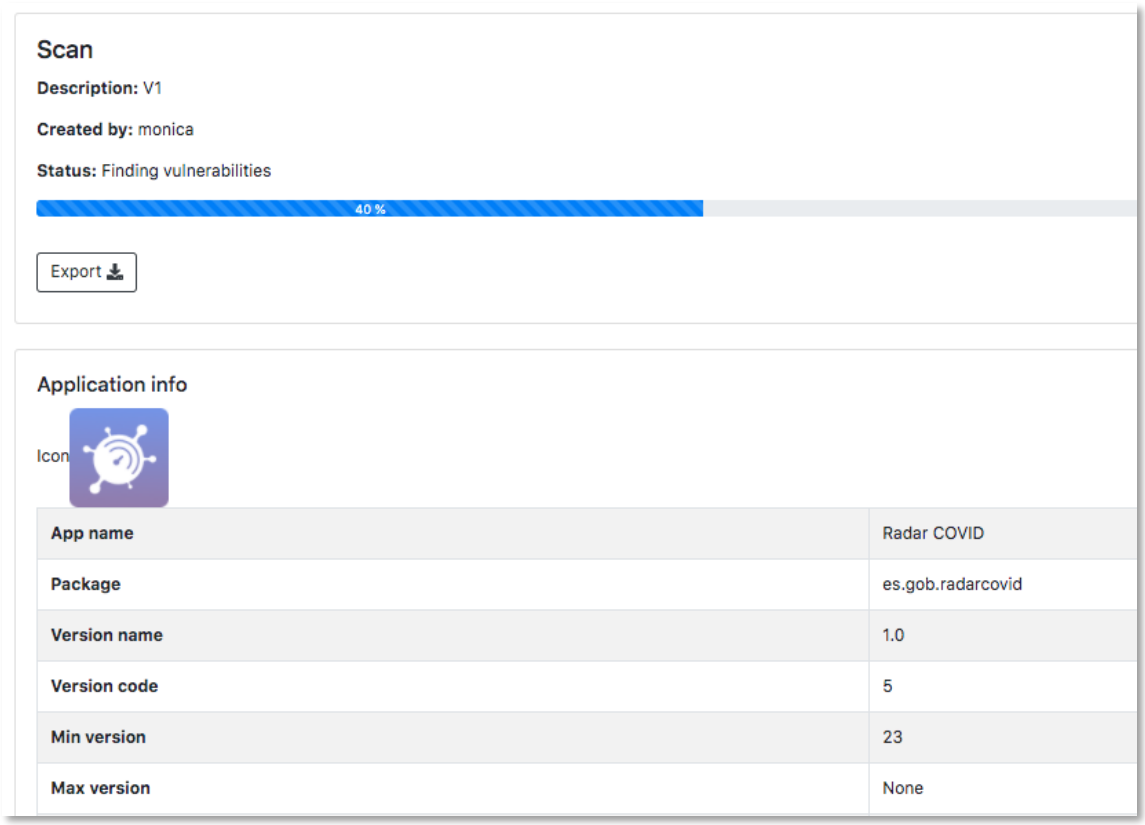
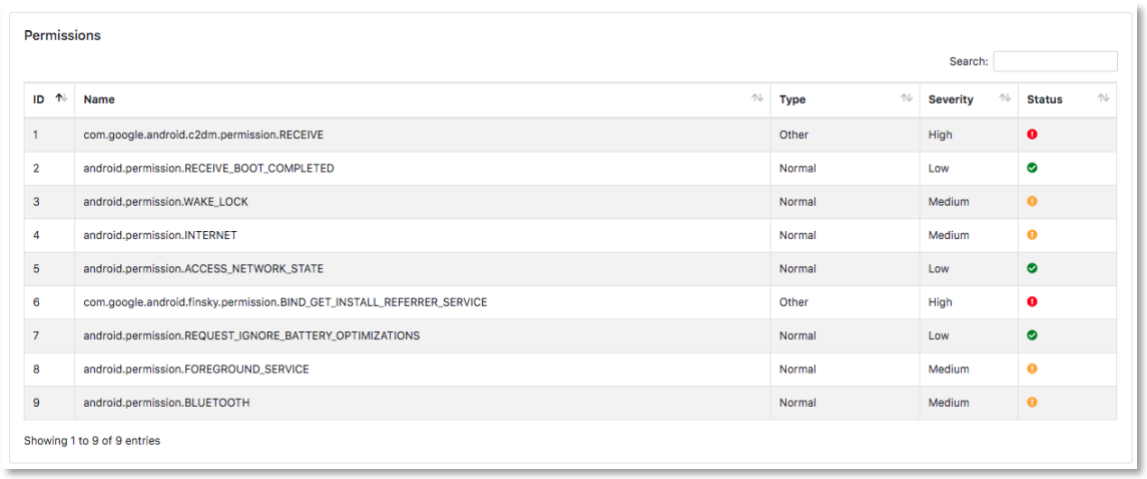


Figura 34 - Información de la aplicación

4.3.2 Permissions

Permisionado clasificado por tipo (ya sea peligrosa, normal, u otra, en caso de que no se tenga guardada en nuestra base de datos y no pueda ser clasificada automáticamente). Además, el estado nos indicará si debemos tener cuidado (en caso de que esté en rojo), si se debe observar con detenimiento (en caso de que sea naranja) o, por el contrario, si aparentemente no tiene peligro (en caso de que esté en verde).



Permissions

Search:

ID	Name	Type	Severity	Status
1	com.google.android.c2dm.permission.RECEIVE	Other	High	
2	android.permission.RECEIVE_BOOT_COMPLETED	Normal	Low	
3	android.permission.WAKE_LOCK	Normal	Medium	
4	android.permission.INTERNET	Normal	Medium	
5	android.permission.ACCESS_NETWORK_STATE	Normal	Low	
6	com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE	Other	High	
7	android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS	Normal	Low	
8	android.permission.FOREGROUND_SERVICE	Normal	Medium	
9	android.permission.BLUETOOTH	Normal	Medium	

Showing 1 to 9 of 9 entries

Figura 35 - Ejemplo de permisos de una apk

Además, dentro del menú de *Others->Permissions*, podemos ver el listado con todos los permisos que tiene la base de datos:

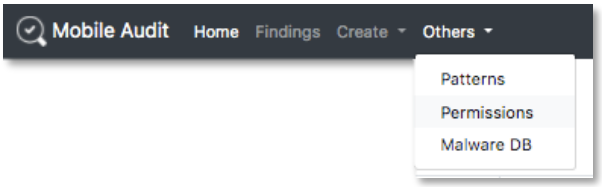
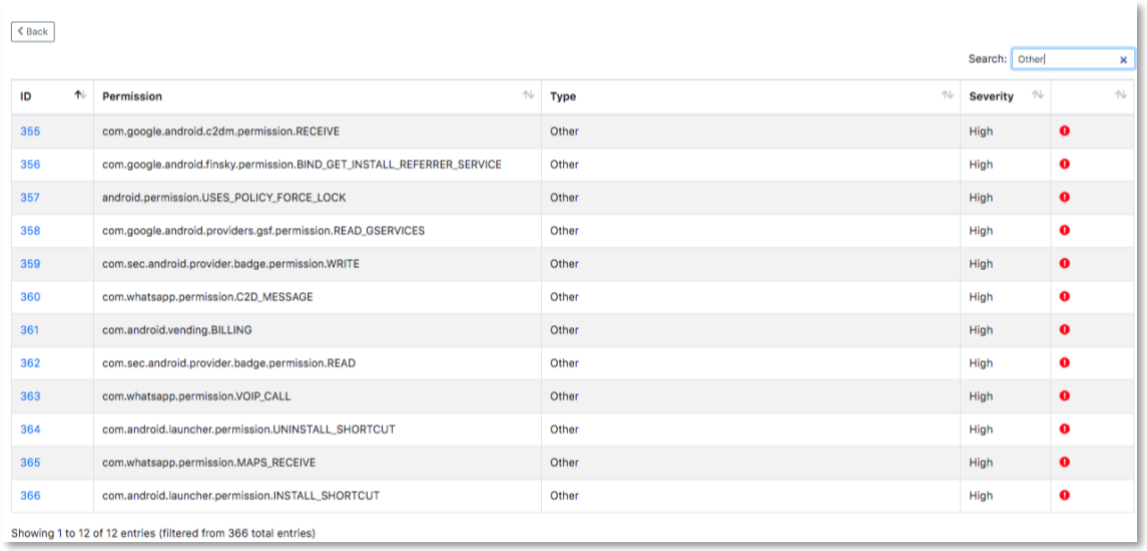


Figura 36 - Menú base de datos de Permissions

Por otro lado, este listado incluirá los nuevos que se vayan añadiendo conforme se hagan auditorías, los cuales, al no ser conocidos, tendrán categoría de *Other* y tendrán criticidad *Alta*.



Back

Search: Other

ID	Permission	Type	Severity	
355	com.google.android.c2dm.permission.RECEIVE	Other	High	
356	com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE	Other	High	
357	android.permission.USES_POLICY_FORCE_LOCK	Other	High	
358	com.google.android.providers.gsf.permission.READ_GSERVICES	Other	High	
359	com.sec.android.provider.badge.permission.WRITE	Other	High	
360	com.whatsapp.permission.C2D_MESSAGE	Other	High	
361	com.android.vending.BILLING	Other	High	
362	com.sec.android.provider.badge.permission.READ	Other	High	
363	com.whatsapp.permission.VOIP_CALL	Other	High	
364	com.android.launcher.permission.UNINSTALL_SHORTCUT	Other	High	
365	com.whatsapp.permission.MAPS_RECEIVE	Other	High	
366	com.android.launcher.permission.INSTALL_SHORTCUT	Other	High	

Showing 1 to 12 of 12 entries (filtered from 366 total entries)

Figura 37 - Other Permissions

4.3.3 Security Info

Se realizará un resumen básico de los findings encontrados clasificados por criticidad y, además, posibles detecciones en VirusTotal (en caso de que esté activada la integración).

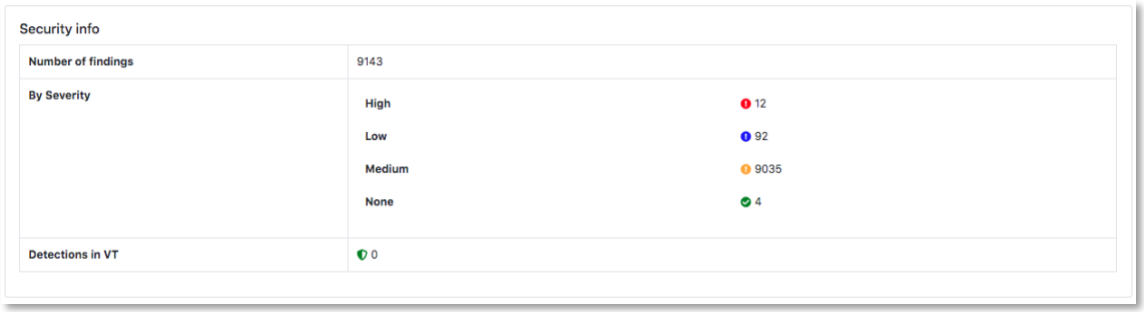


Figura 38 - Security Info de la aplicación

4.3.4 Activities

Se provee un listado de las Activities del proyecto, así como una indicación de cual es la actividad principal.

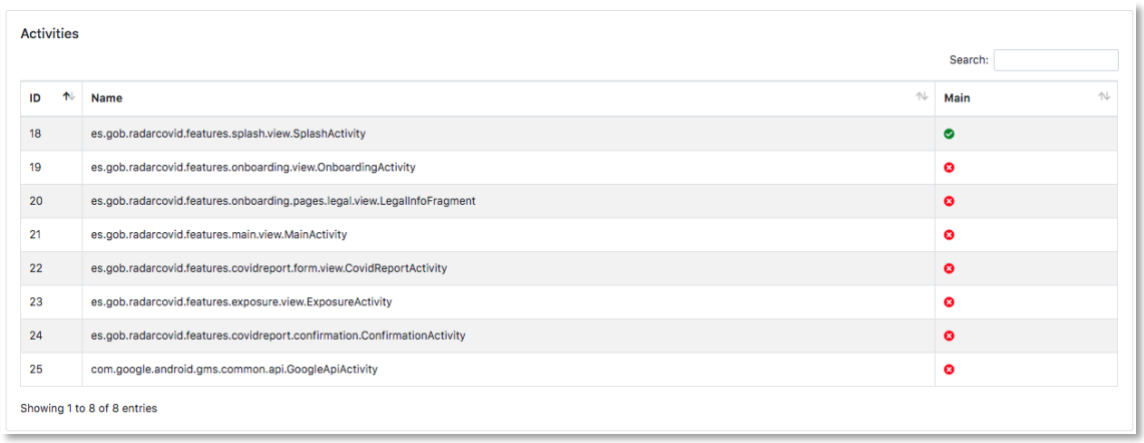


Figura 39 - Listado de activities

4.3.5 Components

Enumerará los diferentes componentes de la aplicación, incluyendo los diferentes Intents de estos. Incluyen: Activities, Services, Broadcast Receivers y Providers.

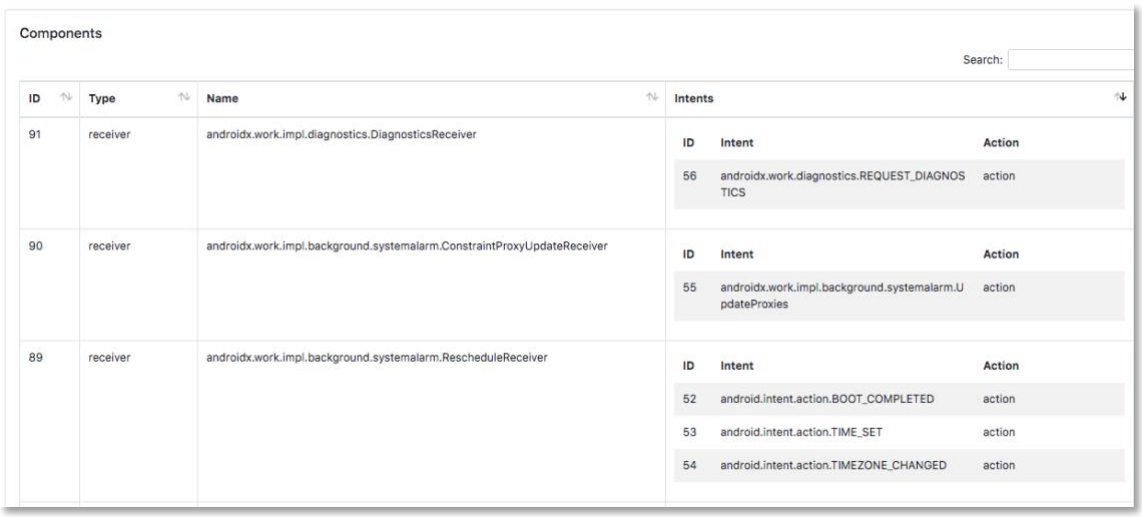


Figura 40 - Listado de components

4.3.6 Certificates

Información de los distintos certificados de la aplicación, incluyendo versiones de estos, quién es la entidad certificadora, los algoritmos, número de serie etcétera.

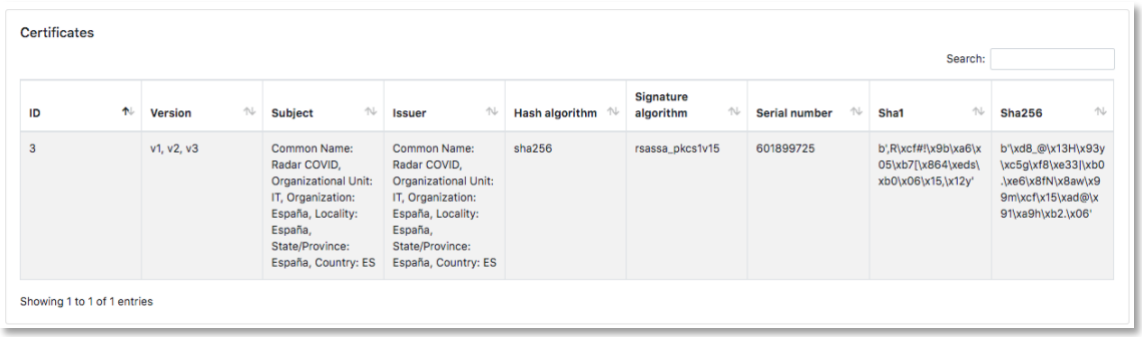


Figura 41 - Certificados

4.3.7 Strings

Este apartado incluye las cadenas interesantes que contiene la aplicación, como por ejemplo: URLs, IPs, credenciales e información sensible entre otros.

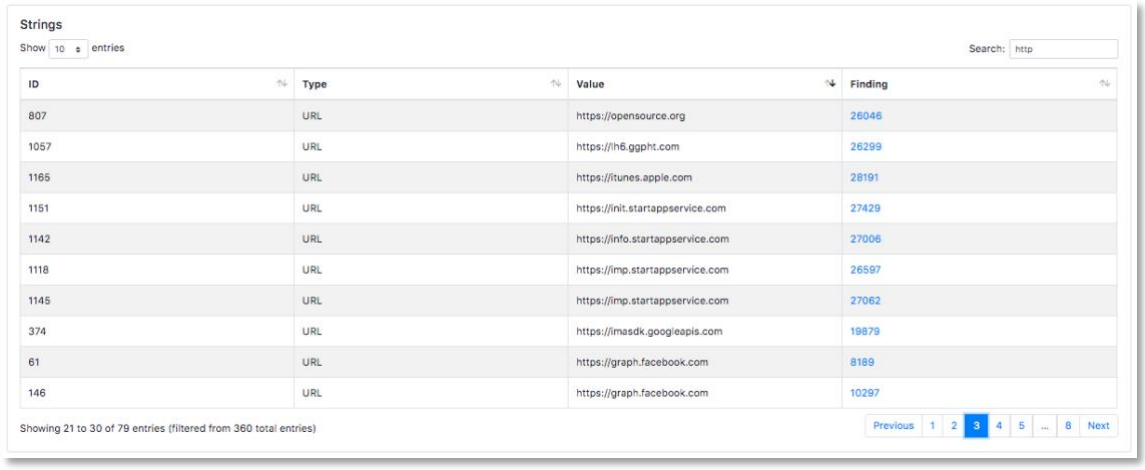


Figura 42 - Strings de una aplicación

Por otro lado, cada una de las URLs encontradas, se buscarán dentro de la base de datos obtenida de Malware DB, para obtener el máximo de información sobre ellas, dado que esta incluye geolocalización, IPs etcétera. Para ver toda la información que contiene, podemos ir al menú *Others->MalwareDB*.

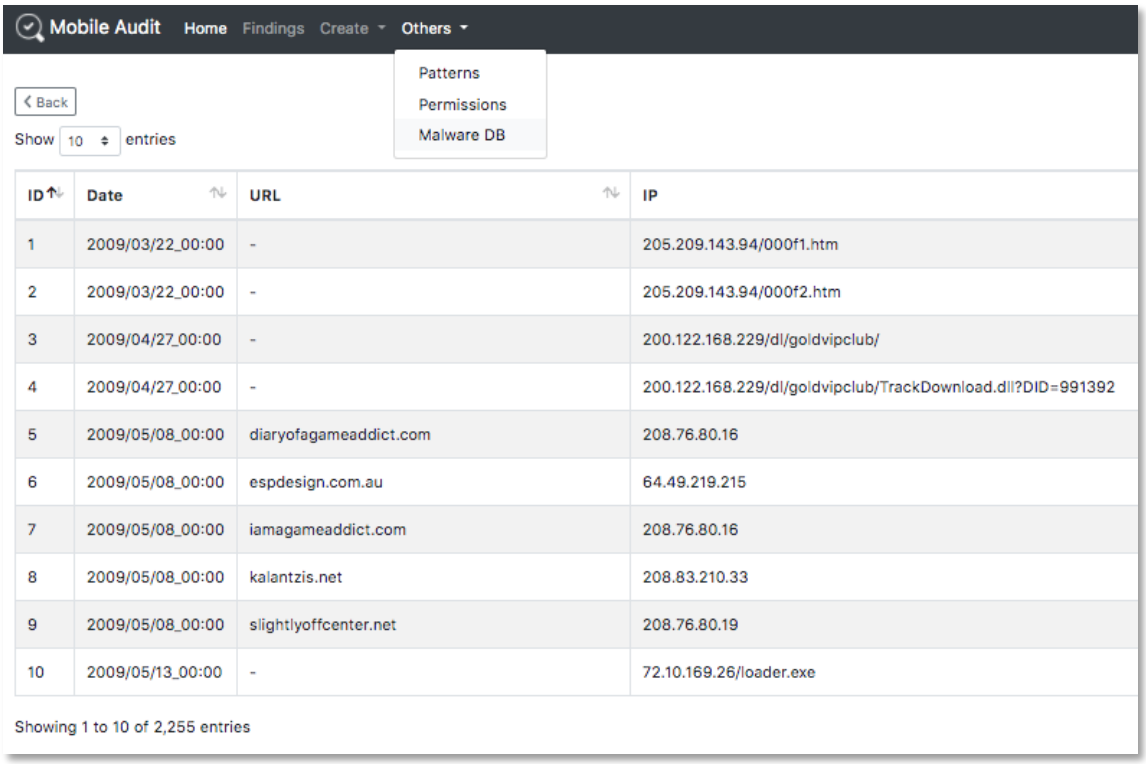
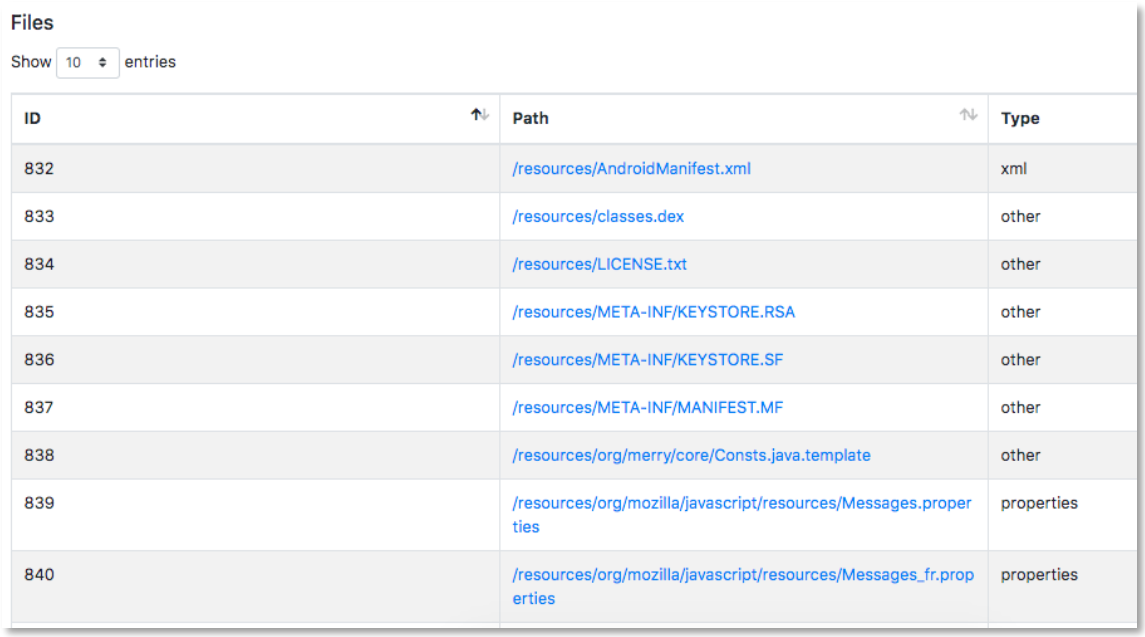


Figura 43 - Malware DB

4.3.8 Files

La herramienta permite recopilar todos los ficheros que se han encontrado dentro de la aplicación (que no sean el propio código fuente *.java* o *.kt*) y categorizarlos por tipo dentro de una tabla como la que vemos a continuación:


Files

Show 10 entries

ID	Path	Type
832	/resources/AndroidManifest.xml	xml
833	/resources/classes.dex	other
834	/resources/LICENSE.txt	other
835	/resources/META-INF/KEYSTORE.RSA	other
836	/resources/META-INF/KEYSTORE.SF	other
837	/resources/META-INF/MANIFEST.MF	other
838	/resources/org/merry/core/Consts.java.template	other
839	/resources/org/mozilla/javascript/resources/Messages.properties	properties
840	/resources/org/mozilla/javascript/resources/Messages_fr.properties	properties

Figura 44 - Files de una aplicación

Por otro lado, la aplicación permite acceder al contenido del fichero haciendo clic sobre el link asociado, como por ejemplo en el *Manifest.xml* que se muestra a continuación:

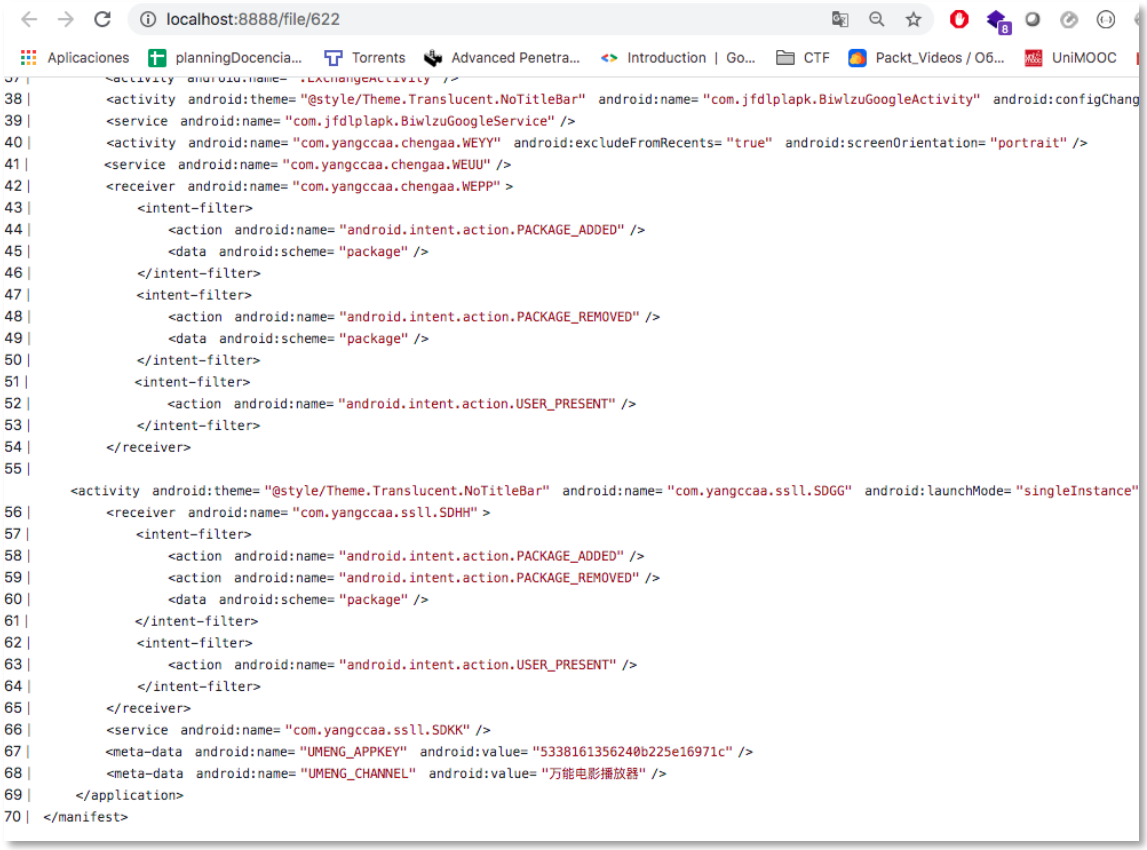


Figura 45 - Visualización del Manifest.xml

Nota: los ficheros HTML no se visualizan puesto que se renderizarían en la propia template de la página y se podría ejecutar código malicioso.

4.3.9 Información sobre VirusTotal

Esta herramienta nos provee de mucha información sobre las detecciones de las firmas de herramientas antivirus que posee, por lo que nos será de gran ayuda a la hora de verificar si nuestra aplicación contiene código malicioso y/o es detectada como malware. Para ello, mandará una petición a la API de la herramienta con el hash de la aplicación, para ver si ha sido detectado con anterioridad y, en caso contrario, si se habilita la subida a Virus Total (no lo está por defecto), subirá el apk para que sea analizado por ella y, posteriormente, mostrará la información recabada por pantalla:

ID	f84fdc32343e070ea7b347bdcf184232ddf994fec1c0e22bde902acef5b125ab	
Data	**Type**	file
	Magic	Zip archive data
	Reputation	0
	Uploaded	No
Date	**Last scan date**	Oct. 19, 2020, 8:51 a.m.
	First seen	None
	First submission	Sept. 18, 2020, 10:25 a.m.
	Last submission	Oct. 19, 2020, 8:51 a.m.
Last scan	**Malicious**	0
	Harmless	0
	Suspicious	0
	Undetected	64
	Timeout	0
	Unsupported	0

Figura 46 - Información VT sobre Radar Covid

Además, podemos ver las detecciones de las distintas firmas de antivirus con su correspondiente información:

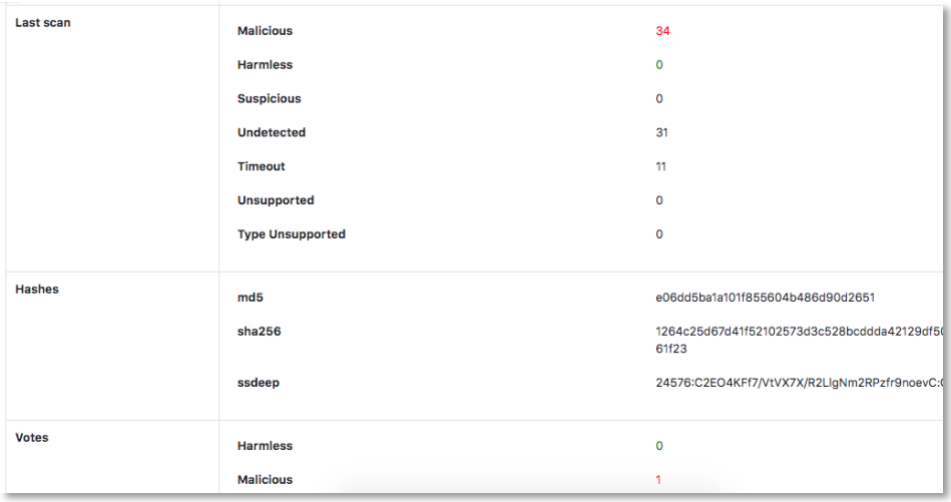
ID	Antivirus	Version	Category	Result	Update	Detected	Status
1	ALYac	1.1.1.5	undetected	None	20201019	No	
2	APEX	6.85	type-unsupported	None	20201016	No	
3	AVG	18.4.3895.0	undetected	None	20201019	No	
4	Acronis	1.1.1.78	type-unsupported	None	20200917	No	
5	Ad-Aware	3.0.16.117	undetected	None	20201019	No	
6	AegisLab	4.2	undetected	None	20201019	No	
7	AhnLab-V3	3.18.2.10046	undetected	None	20201019	No	
8	Alibaba	0.3.0.5	undetected	None	20190527	No	
9	Antiy-AVL	3.0.0.1	undetected	None	20201019	No	
10	Arcabit	1.0.0.881	undetected	None	20201019	No	

Showing 1 to 10 of 75 entries

Previous 1 2 3 4 5 ... 8 Next

Figura 47 - Información Antivirus Radar Covid

Por otro lado, mostraremos un ejemplo en una muestra de malware, donde podemos observar la información básica del último escaneo:



Last scan	Malicious	34
	Harmless	0
	Suspicious	0
	Undetected	31
	Timeout	11
	Unsupported	0
	Type Unsupported	0
Hashes	md5	e06dd5ba1a101f855604b486d90d2651
	sha256	1264c25d67d41f52102573d3c528bcddda42129df5061f23
	ssdeep	24576:C2EO4KFf7/VtVX7X/R2LlgNm2RPzfr9noevC:
Votes	Harmless	0
	Malicious	1

Figura 48 - Información VT malware

A continuación, podemos ver una tabla con las diferentes herramientas y las detecciones que han tenido sobre esta muestra:

ID	Antivirus	Version	Category	Result	Update	Detected	Status
626	ALYac	1.1.1.5	undetected	None	20200622	No	
627	APEX	6.40	type-unsupported	None	20200622	No	
628	AVG	18.4.3895.0	malicious	Android:Agent-IUK [Trj]	20200622	Yes	
629	Acronis	1.1.1.76	type-unsupported	None	20200603	No	
630	Ad-Aware	3.0.5.370	undetected	None	20200622	No	
631	AegisLab	4.2	malicious	SUSPICIOUS	20200622	Yes	
632	AhnLab-V3	3.18.0.10004	malicious	Trojan/Android.XBot.76054	20200622	Yes	
633	Alibaba	0.3.0.5	malicious	Backdoor:Android/Haynu.a58007cc	20190527	Yes	
634	Antiy-AVL	3.0.0.1	malicious	Trojan/Generic.ASMalwAD.39D	20200622	Yes	
635	Arcabit	1.0.0.875	undetected	None	20200622	No	

Showing 1 to 10 of 76 entries

Previous 1 2 3 4 5 ... 8 Next

Figura 49 - Información Antivirus malware

Por otro lado, hay un botón que permite acceder al link externo de la herramienta, donde ver la información y, además, un botón de reload que realiza una llamada a la API para actualizar la información, en caso de que se realicen nuevos análisis en el futuro:

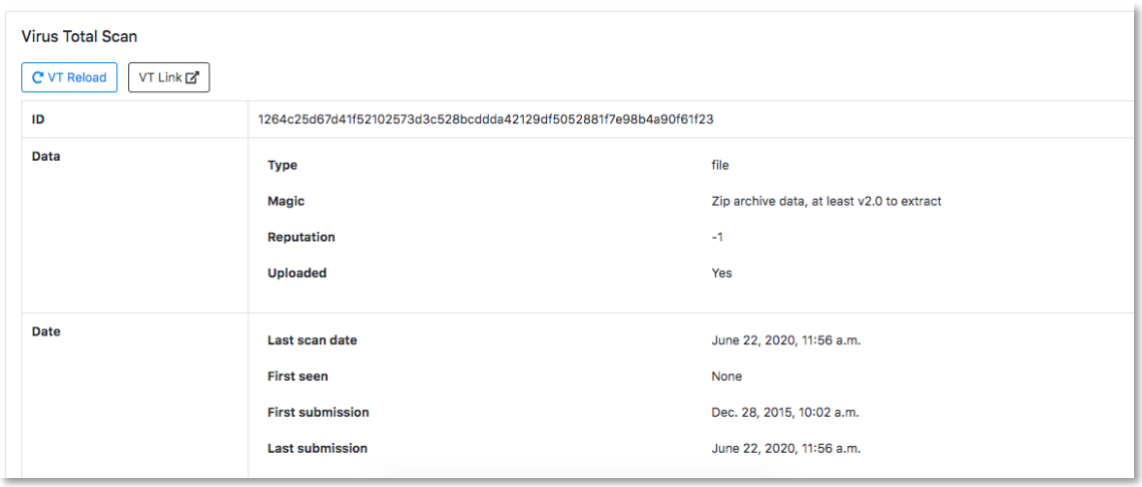


Figura 50 - VT opciones

4.4 Findings

En este apartado, podemos ver la información de los resultados obtenidos ordenados por categorías, según la regla correspondiente.

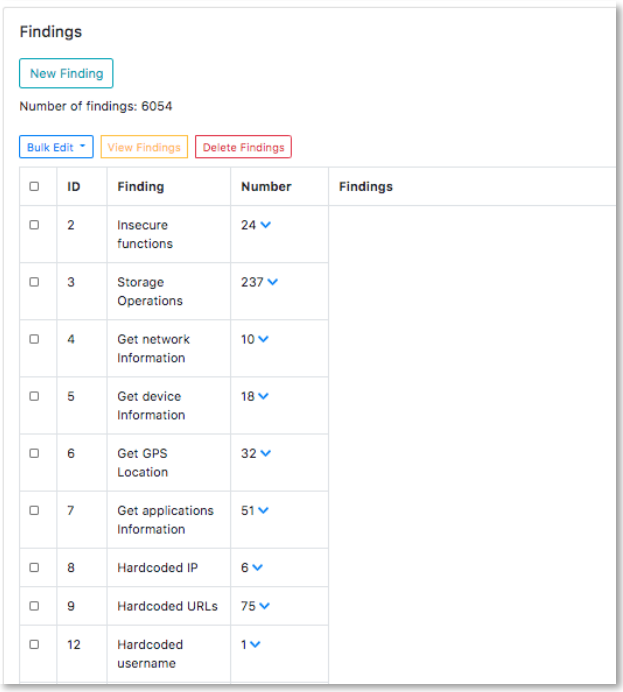


Figura 51 - Findings por categoría de una aplicación

Además, podemos desplegar las categorías y observar los findings de cada categoría:

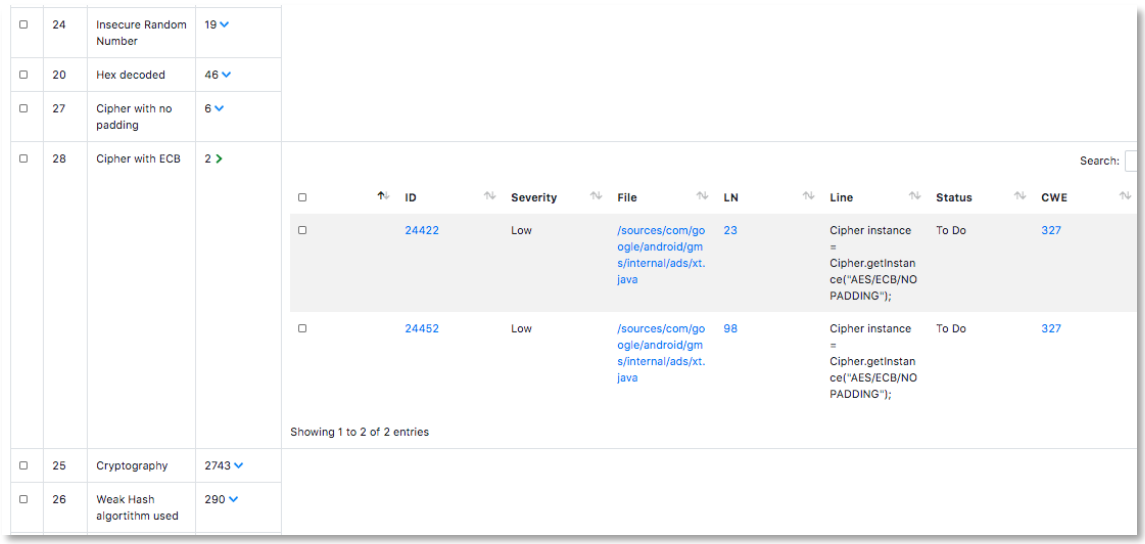


Figura 52 - Findings de una aplicación

4.4.1 Creación de findings de manera manual

Por otro lado, al ser una herramienta que está basada en expresiones regulares, hay vulnerabilidades que no son encontradas, por lo que deberán ser añadidos de manera manual, para ello, tenemos el botón de creación de findings:

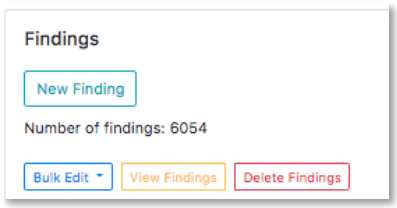


Figura 53 - Creación de findings manuales

Una vez insertada toda la información correspondiente, se guardará en una categoría denominada *Others*, para separarlas del resto de findings:

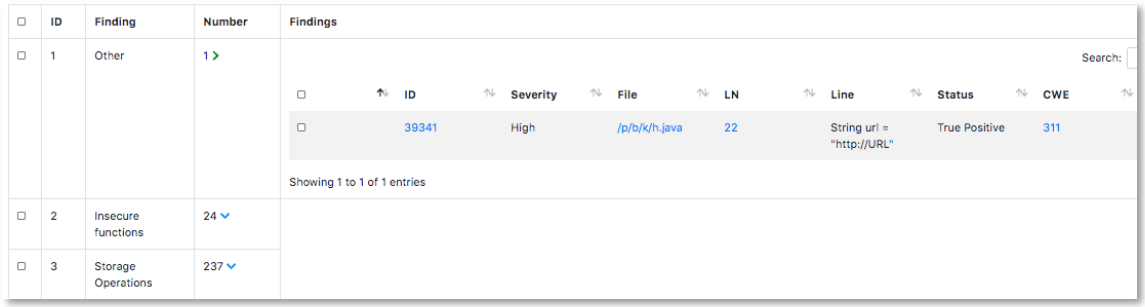


Figura 54 - Visualización de un finding manual

4.5 Triaje de los resultados

En primer lugar, para el triaje o validación de los resultados, se irán seleccionando los findings con el *checkbox* correspondiente y, con el menú de *Bulk Edit*, se podrán editar tanto las criticidades de estos como los diferentes estados.

4.5.1 Editar criticidad

En primer lugar, para editar la criticidad de los findings, seleccionaremos los que queremos editar y, posteriormente, haremos *clic* sobre el menú de *Bulk Edit* y seleccionaremos la criticidad correspondiente. Por último, se hará clic en *Edit Findings* para guardar los cambios.

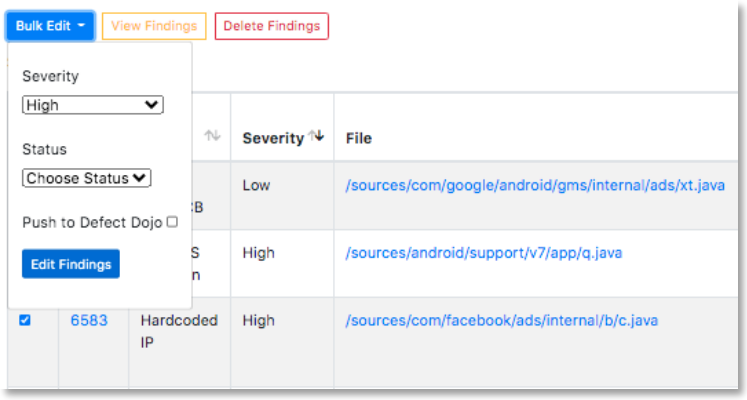


Figura 55 - Edición de criticidad de los findings

4.5.2 Editar estado

Por otro lado, durante el triaje, de la misma manera que hemos comentado el anterior punto, también podremos cambiar el estado de los findings según convenga. Los estados que tenemos son los siguientes:

- To Do: el estado inicial una vez se crean los findings. Correspondería a un estado en el que está pendiente su triaje.
- False Positive: en caso de que el finding no sea explotable.
- True Positive: en caso de que el finding sea una vulnerabilidad explotable.
- Verified: si el finding está verificado por parte de un profesional de seguridad y efectivamente se trata de un True Positive.
- Unknown: si durante el triaje no se conoce si puede ser explotable o no.

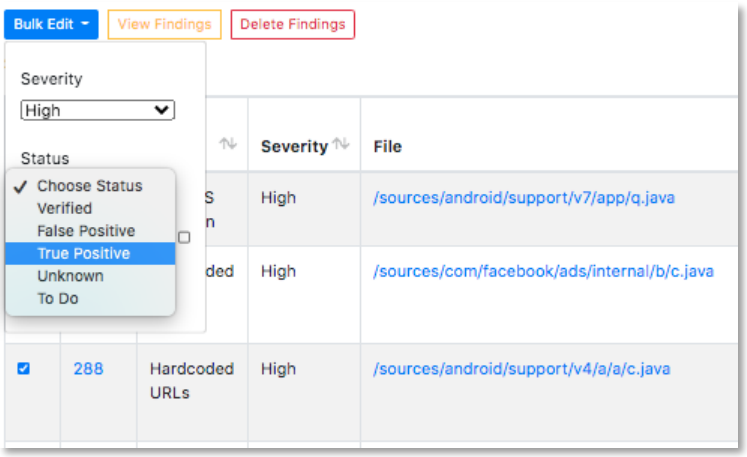


Figura 56 - Edición de estado de los findings

4.5.3 Ver findings

Para facilitar el triaje, podemos seleccionar en Bulk y visualizarlos en una ventana independiente, haciendo clic en *View Findings*:

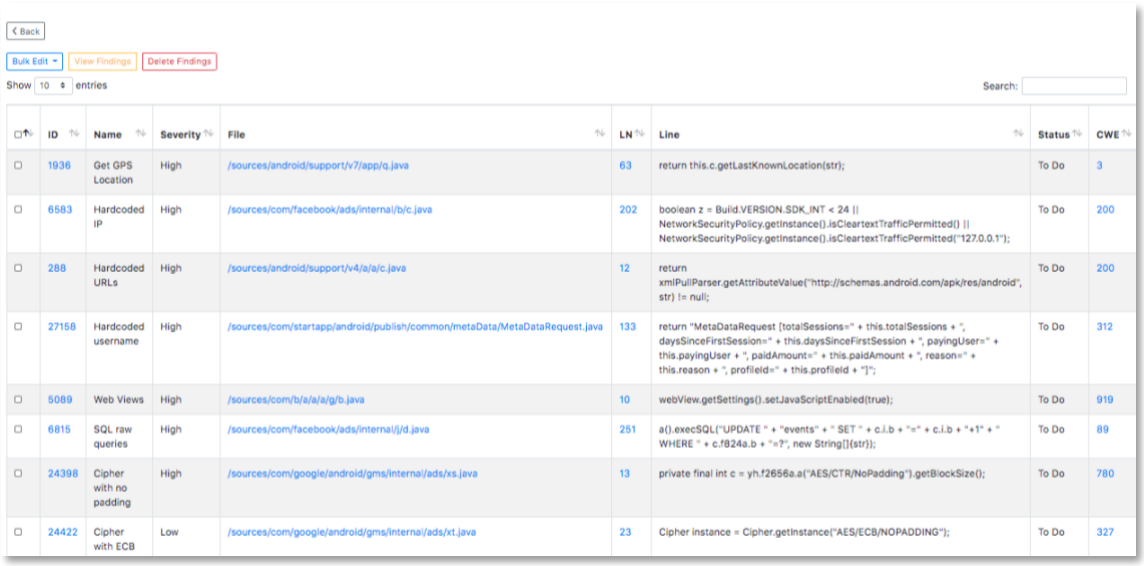


Figura 57 - Visualización de los findings para el triaje

4.5.4 Ver findings

Para facilitar el triaje, al hacer clic sobre cada uno de los identificadores, se puede observar y editar cada uno de los findings de manera individual.

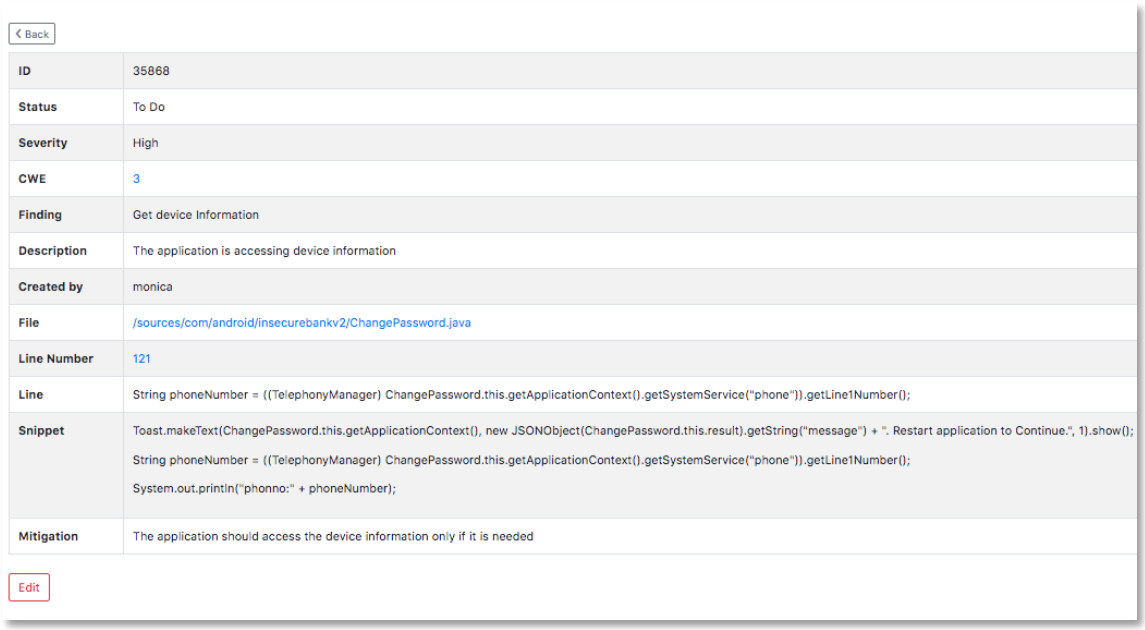


Figura 58 - Visualizar un finding

Por otro lado, nos encontramos links que nos permiten ver el contenido del fichero de esa vulnerabilidad, para poder determinar con mayor precisión si se trata de un verdadero o falso positivo.

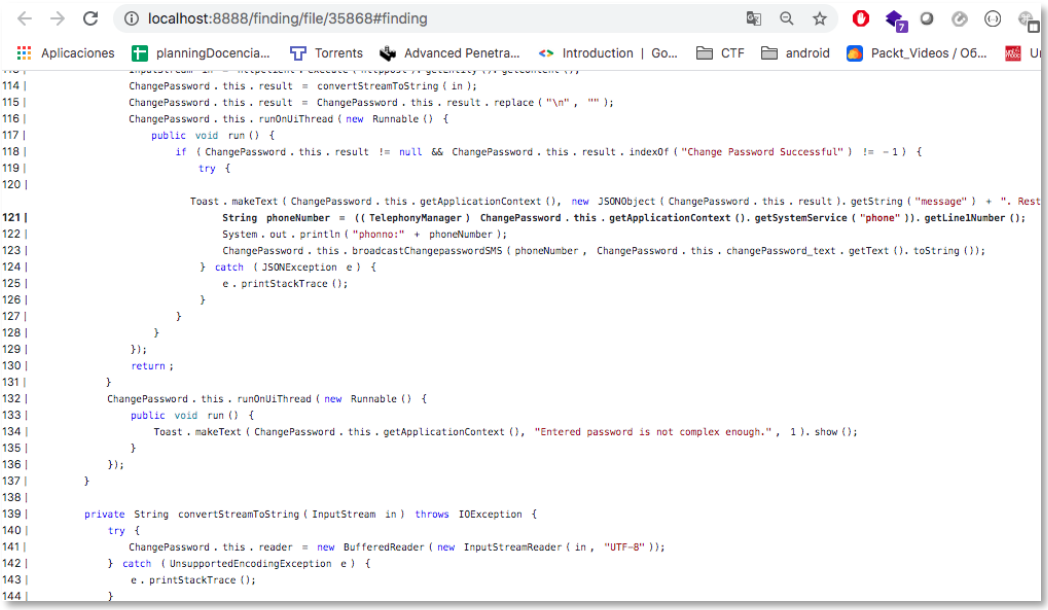


Figura 59 - Visualizar fichero de un finding

4.5.5 Enviar a Defect Dojo

Por último, como hemos comentado anteriormente, se ha integrado con una herramienta de gestión de defectos que es Defect Dojo y, se permite enviar los findings a esta para centralizar todas las vulnerabilidades de las aplicaciones de seguridad.

Para ello, una vez seleccionados los findings a enviar, dentro del menú de *Bulk Edit*, se selecciona el *checkbox* de *Push to DefectDojo* antes de dar a *Edit Findings* y modificar la información.

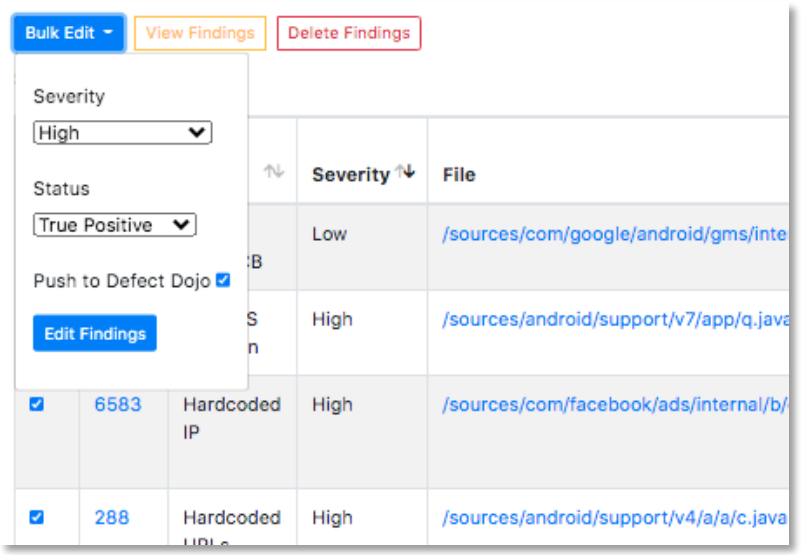


Figura 60 - Enviar findings a Defect Dojo

Una vez los findings sean enviados, se podrá ver el identificador en Defect Dojo con un link que nos llevará al finding en esa herramienta:

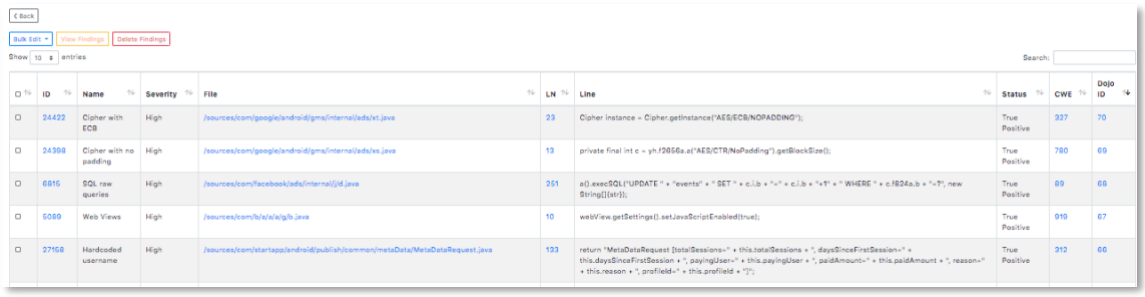


Figura 61 - Visualizar id del finding de Defect Dojo

Además, también se incluirá el botón para ir a la herramienta desde el propio finding:

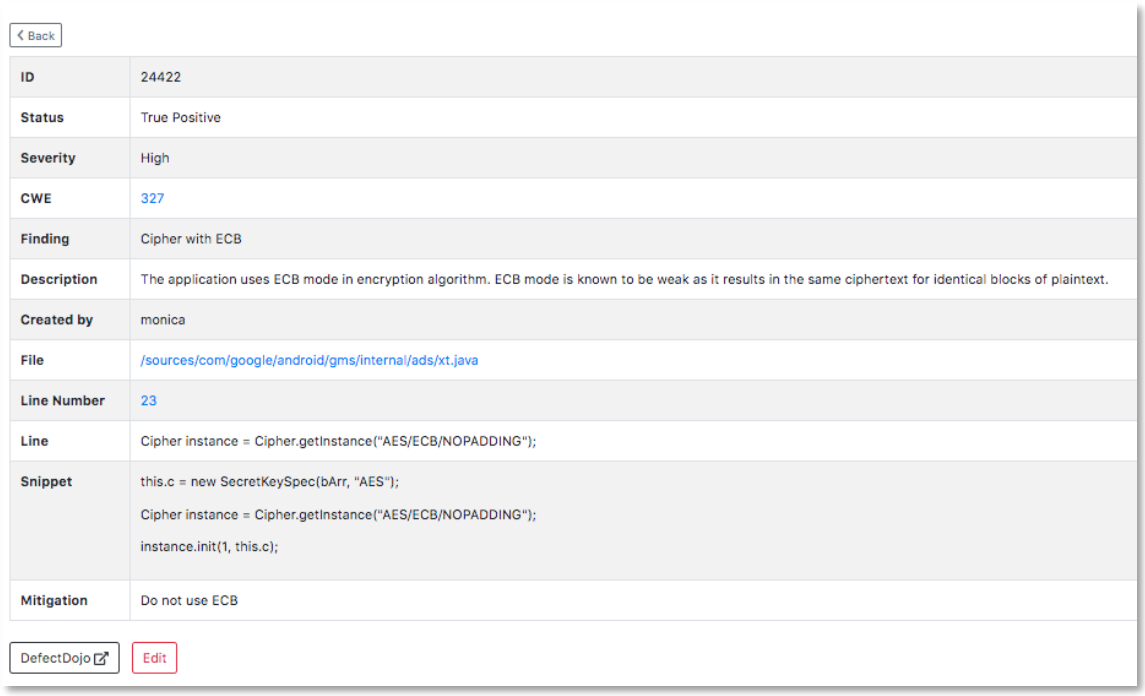


Figura 62 - Visualizar finding creado en Defect Dojo

Por último, una vez visitemos ese link, podremos acceder al finding dentro de la herramienta:

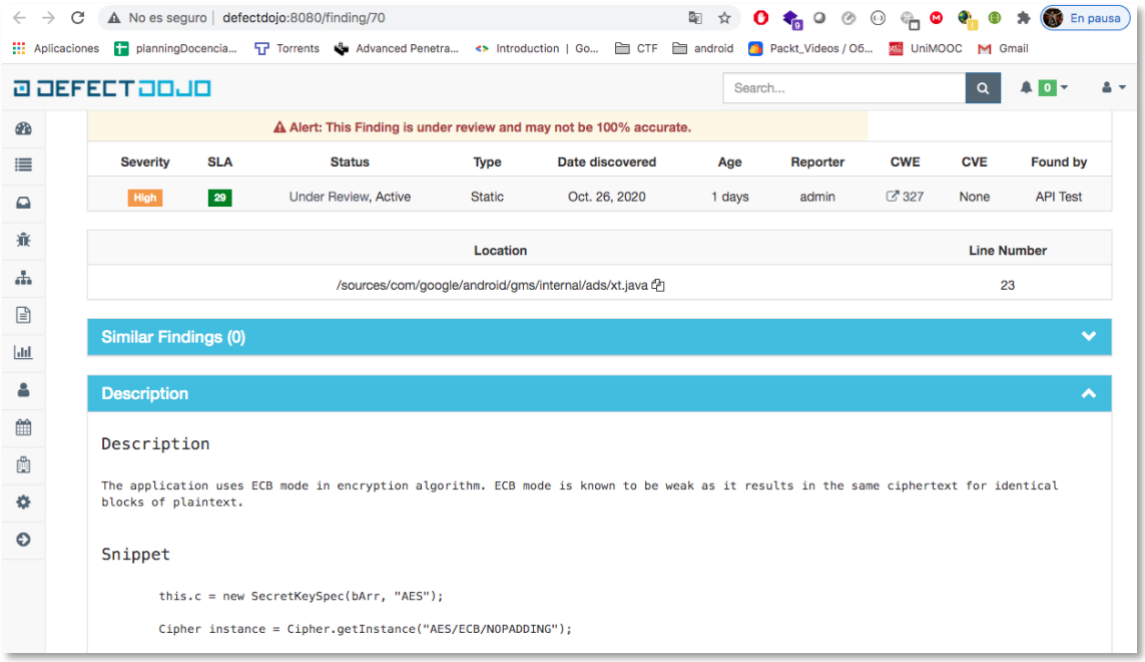
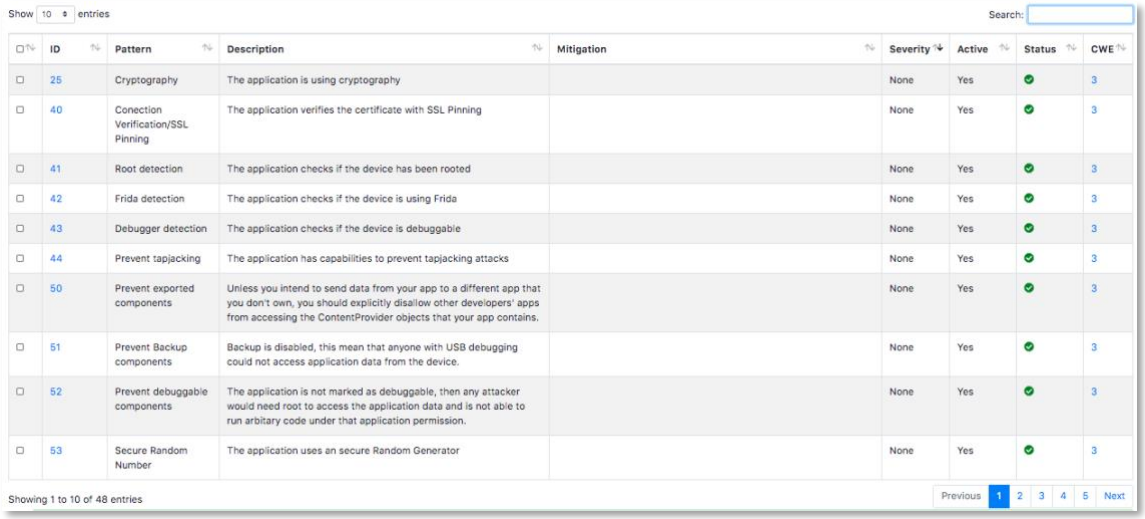


Figura 63 - Visualizar finding creado en la herramienta Defect Dojo

4.6 Mejores prácticas en seguridad

Además de encontrar vulnerabilidades, se ha querido poner en valor las buenas prácticas en la implementación segura del software, por lo que nos encontramos un apartado que describen las mejores prácticas, y se indica las partes del código donde se están realizando de manera correcta. Así, sirve de manera formativa al desarrollador y, en caso de que se sigan esas recomendaciones, se realiza un refuerzo positivo a este. A continuación, podemos ver algunas de las reglas de las mejores prácticas:



Show 10 entries — Search:

ID	Pattern	Description	Mitigation	Severity	Active	Status	CWE
25	Cryptography	The application is using cryptography		None	Yes	✓	3
40	Conection Verification/SSL Pinning	The application verifies the certificate with SSL Pinning		None	Yes	✓	3
41	Root detection	The application checks if the device has been rooted		None	Yes	✓	3
42	Frida detection	The application checks if the device is using Frida		None	Yes	✓	3
43	Debugger detection	The application checks if the device is debuggable		None	Yes	✓	3
44	Prevent tapjacking	The application has capabilities to prevent tapjacking attacks		None	Yes	✓	3
50	Prevent exported components	Unless you intend to send data from your app to a different app that you don't own, you should explicitly disallow other developers' apps from accessing the ContentProvider objects that your app contains.		None	Yes	✓	3
51	Prevent Backup components	Backup is disabled, this mean that anyone with USB debugging could not access application data from the device.		None	Yes	✓	3
52	Prevent debuggable components	The application is not marked as debuggable, then any attacker would need root to access the application data and is not able to run arbitary code under that application permission.		None	Yes	✓	3
53	Secure Random Number	The application uses an secure Random Generator		None	Yes	✓	3

Showing 1 to 10 of 48 entries — Previous 1 2 3 4 5 Next

Figura 64 - Security Best Practices Patterns

Para finalizar, veremos un ejemplo de estas reglas aplicadas en una auditoría, por ejemplo, en este caso se indica dónde se está realizando una verificación de los certificados o SSL Pinning y, por otro lado, en qué partes del código se utilizan números aleatorios seguros (*Secure Random Number*):

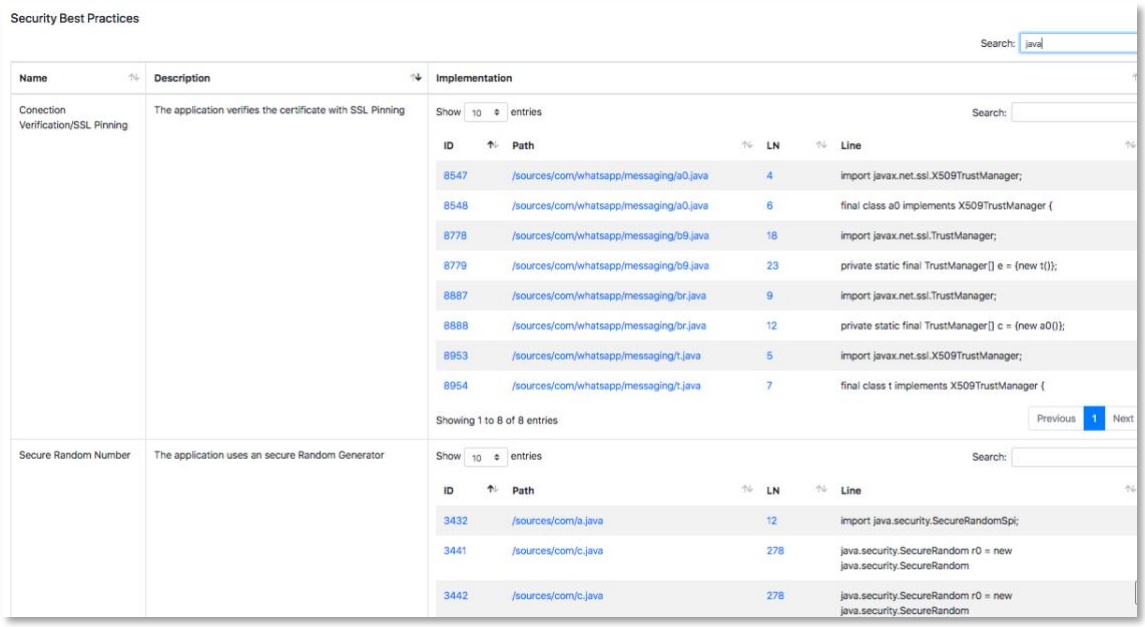
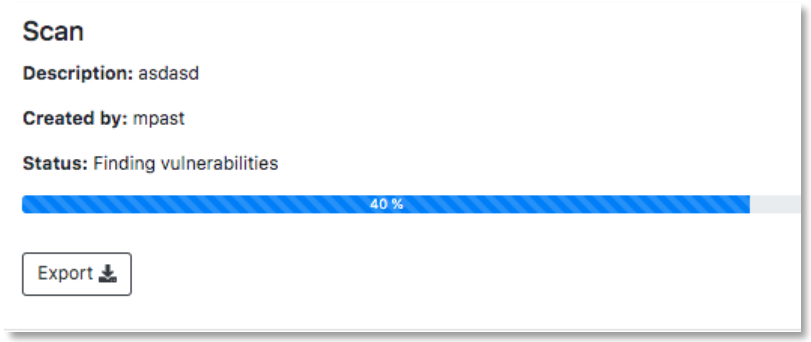


Figura 65 – Best Security Practices de una aplicación

4.7 Informe de resultados

Como último paso dentro de la auditoría, deberemos acceder al informe de resultados. Para ello, en la herramienta hay una funcionalidad de exportar los resultados de un escaneo a formato PDF, para poder visualizarlos en un informe final y así verificar la seguridad de la aplicación y confirmar de manera más clara que no hay malware en la aplicación auditada.



Así, una vez hagamos clic al botón de Export, obtendremos el pdf con el informe del escaneo:

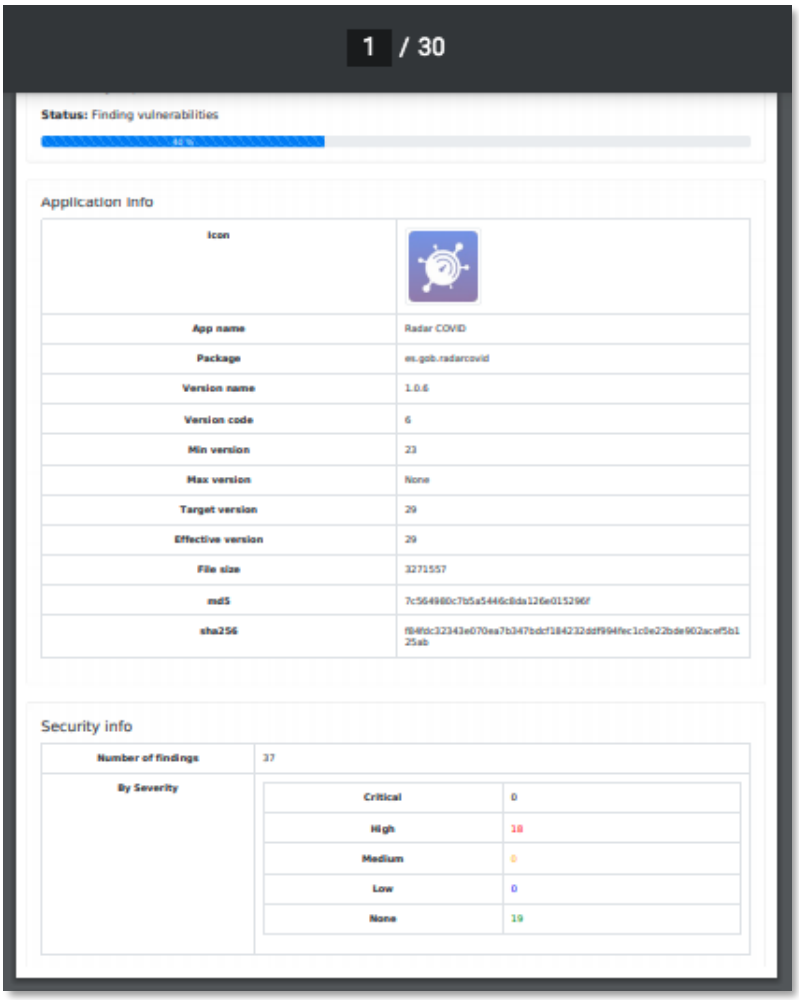


Figura 66 - Informe de resultados de la auditoría

CAPÍTULO 5. DESARROLLO DE LA HERRAMIENTA

Para el desarrollo de la aplicación de auditorías en Android, hemos barajado numerosas tecnologías, frameworks y herramientas para poder integrar y realizar nuestro propósito de la mejor manera y, a continuación, comentaremos la selección que hemos llevado a cabo.

5.1 Herramientas utilizadas

Una vez realizado este estudio, se ha decidido hacer uso de Python, debido a que proporciona numerosas librerías para ayudarnos a poder llevar a cabo la auditoría de manera satisfactoria. Además, para la parte visual de la aplicación, donde se mostrarán los resultados y se permitirá el triaje de estos, se ha elegido el framework Django, por su amplio uso y porque nos permite hacer uso de librerías para gestionar usuarios, sesiones etcétera de manera sencilla. Por último, para facilitar su despligue se ha utilizado la tecnología de contenedores Docker, que nos permite la creación de una imagen con todas las librerías, herramientas etcétera y su despliegue de manera fácil y sencilla en cualquier sitema operativo.

Por lo tanto, nos encontramos con un despliegue de una aplicación que sigue el patrón MVC (Model View Controller) que nos permite separación entre los distintos elementos de nuestra aplicación haciendo uso del framework Django. Además, irá desplegado sobre un servidor de Python denominado uwsgi, que servirá nuestra aplicación. Por otro lado, para servir los archivos estáticos, gestionar certificados etcétera, haremos uso del servidor Nginx como proxy, que redirigirá las peticiones que lleguen a este hacia el servidor uwsgi de nuestra aplicación. Por último, nos encontramos una base de datos PostgreSQL para el modelo de datos y gestionar la persistencia.

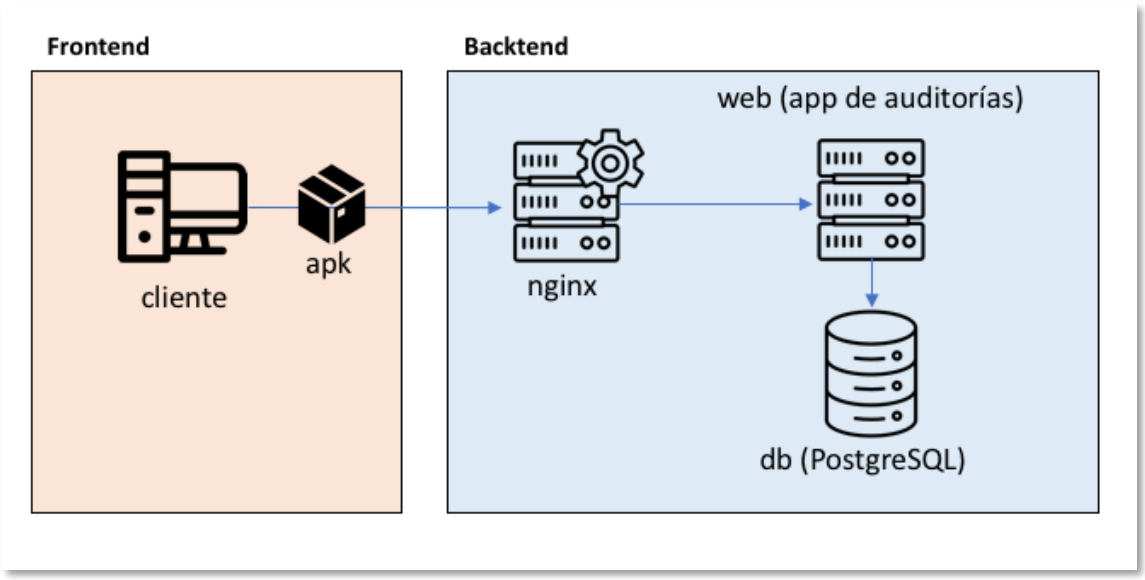


Figura 67 - Arquitectura de la herramienta

5.2 Herramientas para la auditoría

A continuación, se enumerarán las diferentes herramientas utilizadas durante las distintas fases de la auditoría.

5.2.1 Acceso al APK

En primer lugar, para tener acceso al apk que analizaremos, tendremos dos maneras de realizarlo: en caso de que tengamos un dispositivo rooteado y podamos acceder a este mediante la herramienta adb, podremos ejecutar el siguiente comando para extraer el apk desde el dispositivo indicando el nombre de paquete correspondiente a la aplicación que auditaremos:

```
adb pull /data/app/<package name>
```

Por otro lado, en caso de no poder realizar lo anterior, podemos utilizar servicios online que nos permiten descargar las apks, como pueden ser:

- APK downloader: https://apps.evozi.com/apk-downloader/
- APK Pure: https://apkpure.com/es/

Al finalizar este paso que se realiza de manera manual, una vez obtenido el apk, el resto de pasos, los realizará de manera automática nuestra aplicación.

5.2.2 Reversing de la APK

En la primera fase, en la que tenemos el apk, se sube a nuestra aplicación y automáticamente empieza todo el proceso de auditoría. En primer lugar, se utiliza androguard para acceder al máximo de información sobre esta, dado que obtiene los distintos componentes, los permisos de la aplicación, los certificados etcétera.

Por otro lado, se usa jadx para decompilar el código y poder acceder a las clases *.java* de esta. Se barajó la opción de utilizar APK-Tool, que extrae el código *smali* para la realización de las auditorías, pero sería menos legible por parte de los desarrolladores que realizasen la auditoría, dado que ellos programan en Java/Kotlin, por lo tanto, es mejor ofrecerles el mismo lenguaje para poder entender más fácilmente las vulnerabilidades en el código.

5.2.3 Análisis estático

Una vez tenemos acceso al código decompilado, se procederá a la realización de la auditoría. En primer lugar, si está habilitada la opción, se utiliza la API de VirusTotal, en su última versión (v3), para la búsqueda de información sobre el último análisis realizado del fichero apk, en caso de que se haya realizado o, por el contrario, la subida de este apk a la herramienta para la realización de un análisis (en caso de que se haya especificado en los *settings* de la aplicación que se permita la subida a la plataforma online, dado que podemos estar realizando una auditoría a una aplicación sensible que no queramos que se suba información a internet).

A continuación, realiza una búsqueda línea a línea de patrones de vulnerabilidades que se han creado haciendo uso de expresiones regulares o regex. Estas reglas nos permiten una facilidad de creación o modificación de las que tenemos y, además, es una manera rápida de identificar patrones en el código que pueden ser susceptibles de ser vulnerabilidades o código malicioso.

Además, se realiza una búsqueda extensiva de strings e información sensible dentro del código, como podrían ser credenciales, IPs, URLs etcétera. Estas últimas son contrastadas con una base de datos de malware que nos provee www.malwaredomainlist.com en la que verificaremos si se trata de alguna url relacionada con malware y en ese caso, extraeremos toda la información disponible sobre ella (geolocalización, *reverse lookup*…).

Por el contrario, se han creado reglas que nos indiquen si se están realizando las mejores prácticas en seguridad Android, como, por ejemplo, en caso de que se verifique que se está haciendo Certificate Pinning o se verifica que no se esté ejecutando el usuario root etcétera.

Por otro lado, se identificarán ficheros dentro de la aplicación, como pueden ser imágenes, vídeos, ficheros de *properties*… para que puedan ser visualizados por el auditor y encontrar alguna información relevante y, bases de datos, que en este caso se extraerá y visualizará toda la información de esta.

Por último, todo ello será persistido en un modelo de datos muy completo que se ha generado para una correcta identificación y posterior visualización de este.

5.2.4 Triaje de resultados

El auditor tendrá disponible la información de los findings encontrados en tablas categorizadas por tipo de vulnerabilidad. Por lo tanto, deberá ir verificando estos findings y actualizando las criticidades o estados correspondientemente. Para facilitar el triaje, se permite realizar una visualización de los ficheros que contienen la vulnerabilidad y, para ello se ha utilizado la herramienta pygments, que permite dar formato al texto para visualizarlo con la sintaxis correspondiente.

5.2.5 Informe de resultados

Para finalizar, una vez extraído la máxima información disponible, se podrá exportar el informe de resultados en PDF, el cual se ha generado haciendo uso de la herramienta wkhtmltopdf y la librería de python pdfkit utilizando un template de html que se genera de manera dinámica con la información del escaneo.

CAPÍTULO 6. DESPLIEGUE DE LA HERRAMIENTA

En este apartado, el cual será más práctico, en primer lugar, se verán las distintas opciones y configuraciones a la hora de utilizar la herramienta, así como las distintas integraciones de herramientas que ofrece y, por último, se describirán los pasos para realizar el despliegue de la herramienta.

6.1 Configuración de la herramienta

En primer lugar, las variables globales de la aplicación están definidas dentro del un fichero de entorno *.env*, el cual en la instalación, incluiremos un fichero de ejemplo *.env.example*. Estas variables serán las siguientes:

- SECRET_KEY=<String>

  Se trata de una variable que sólo conocerá nuestra aplicación y permitirá realizar operaciones criptográficas de manera segura, como pueden ser: creación de salts, usar identificadores de sesión únicos etcétera-

- DEBUG=0

  Se trata de una variable para desplegar la aplicación en modo debug, en caso de que estemos en entorno de desarrollo y queramos ver mayor información sobre traza de errores etcétera.

  Nota: nunca ponder Debug a 1 en entorno productivo.

- DJANGO_ALLOWED_HOSTS=web localhost 127.0.0.1 [::1]

Serán los hosts sobre los que se permitirán conectar a nuestra aplicación.

- ENV=(PROD|DEV)

  En caso de que el entorno a desplegar sea de desarrollo, el valor será DEV, y se desplegará en una base de datos en memoria SQLite, en cambio, si el entorno es producción PROD, la base de datos será productiva.

- SQL_ENGINE=django.db.backends.postgresql

  Este es el valor correspondiente del backed de nuestra base de datos PostgresSQL.

- SQL_DATABASE=audit

  Variable que indica el nombre de la tabla de base de datos a utilizar.

- SQL_USER=postgres

  Variable que indica el usuario de base de datos.

- SQL_PASSWORD=<String>

  Contraseña de la base de datos.

- SQL_HOST=db

  Host de la base de datos, en nuestro caso, al desplegarse por docker-compose, el host será el mismo que el nombre del servicio, es decir, db.

- SQL_PORT=5432

  Puerto de escucha de la base de datos, al ser PostgreSQL, el puerto por defecto es este.

- LANG=en_US.UTF-8

Todas las variables de entorno necesarias en nuestra aplicación serán accedidas dentro del fichero *app/config/settings.py*, en el que, en caso de que no se definan, se pondrán valores por defecto. Por ejemplo, como podemos ver a continuación, buscará las variables de entorno de Virus Total y, en caso de que no estén definidas, pondrá los valores siguientes:

```
VIRUSTOTAL_ENABLED = env('VIRUSTOTAL_ENABLED', True)
VIRUSTOTAL_URL = env('VIRUSTOTAL_URL', 'https://www.virustotal.com/')
```

6.2 Integraciones con otras aplicaciones

A continuación, indicaremos las diferentes configuraciones de las integraciones que tiene nuestra aplicación.

6.2.1 Defect Dojo

En primer lugar, tenemos la integración con la herramienta de gestión de defectos Defect Dojo.

- DEFECTDOJO_ENABLED=<Boolean>

  En primer lugar, indicaremos si la integración está activada poniendo el valor a *True*.

- DEFECTDOJO_URL=http://defectdojo:8080/finding/

  Indicaremos la url externa a la que accederá para ver en la herramienta el finding correspondiente, en caso de que se hayan enviado a esta.

- DEFECTDOJO_API_URL=http://defectdojo:8080/api/v2/

  Esta será la url externa de la API para la creación de findings, en nuestro caso usaremos la última versión (v2)

- DEFECTDOJO_API_KEY=<String>

  Por último, habrá que indicar el API Key generado en la herramienta Defect Dojo que tenga permisos para la creación de findings.

6.2.2 Virus Total

En primer lugar, tenemos varias opciones para utilizar la API de VirusTotal. Utilizamos la versión v3 de Virus Total para la búsqueda de información sobre el último análisis realizado del fichero apk, en caso de que se haya realizado.

- VIRUSTOTAL_ENABLED=<Boolean>

  Variable que indica si la integración está habilitada.

- VIRUSTOTAL_URL=https://www.virustotal.com/

  Url del servicio de Virus Total.

- VIRUSTOTAL_FILE_URL=https://www.virustotal.com/gui/file/

  Se trata de la url externa que permite la visualización del resultado de un análisis en la herramienta.

- VIRUSTOTAL_API_URL_V3=https://www.virustotal.com/api/v3/

  Url de la última versión de la API, la que nos provee con una mayor información.

- VIRUSTOTAL_API_URL_V2=https://www.virustotal.com/vtapi/v2/

  Url de la versión anterior de la API, para ciertas operaciones que no funcionan en la última versión.

- VIRUSTOTAL_API_KEY=<String>

  Se trata de la API Key generada en VirusTotal para poder acceder a los servicios que nos provee.

- VIRUSTOTAL_UPLOAD=<Boolean>

  En caso de que la apk no se haya analizado anteriormente y que esté habilitado, se permitirá la subida a la plataforma online. Queremos separar el habilitar la integración de la herramienta con la propia subida de información debido a que podemos estar realizando una auditoría a una aplicación sensible que no queramos que se suba información a internet o que sea confidencial.

6.2.3 Malware DB

Por último, la integración con MalwareDB:

- MALWAREDB_ENABLED=<Boolean>

  Indicará si está habilitada la integración.

- MALWAREDB_URL=https://www.malwaredomainlist.com/mdlcsv.php

  La URL correspondiente a Malware DB.

6.3 Componentes de la aplicación

Por otro lado, para hablar de los componentes de la aplicación, en primer lugar, como hemos comentado anteriormente, nos encontramos con un despliegue basado en contenedores, lo cual agiliza enormemente la operación de esta, dado que simplemente

será necesario instalar la herramienta Docker para levantar los contenedores y docker-compose, que viene incluida en la instalación por defecto de la primera, para la orquestación de estos.

Para ello, se puede seguir la documentación oficial para la instalación de esta en el siguiente link.

Durante el despliegue se levantan tres contenedores: en primer lugar, una base de datos PostgreSQL, aunque es compatible con cualquier otra relacional, como por ejemplo MySQL, MariaDB etcétera. Se ha elegido la primera por la rapidez y la customización que permite en el despliegue con Docker. En segundo lugar, un contenedor con el servidor Nginx, que como ya hemos comentado en apartados anteriores, actúa como *proxy* y sirve los ficheros estáticos de nuestra aplicación. Y, por último, un contenedor con una imagen custom que tiene como imagen base una de Python y que se le van añadiendo en las distintas capas, las distintas herramientas o configuraciones que son necesarias. En la siguiente imagen se verá un esquema de la arquitectura de nuestra aplicación (incluyendo las integraciones):

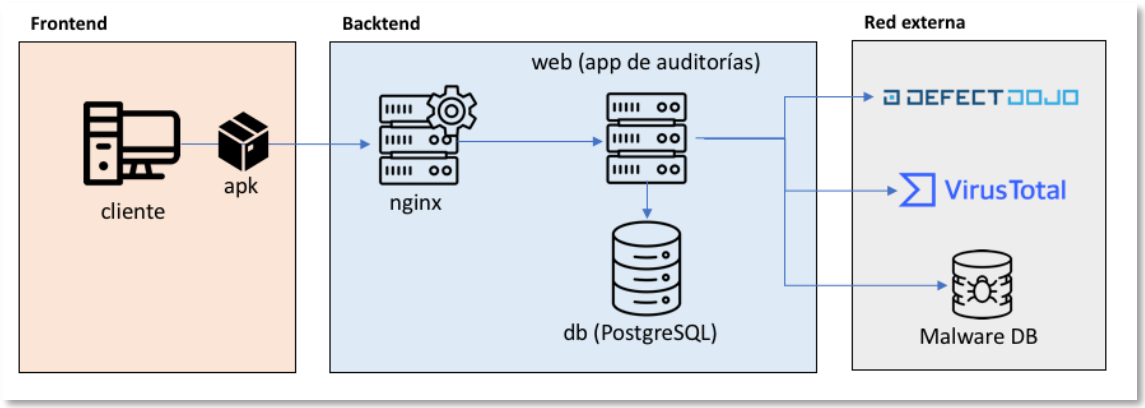


Figura 68 - Esquema de la arquitectura de la aplicación

Una vez visto el esquema, describiremos el fichero *docker-compose.yml* donde en formato YAML incluimos todos los servicios de la arquitectura y configuraciones de estos, las cuales contienen variables de entorno con el formato *${VARIABLE}*, que utilizarán los distintos contenedores durante su ciclo de vida, los cuales veremos a lo largo de los siguientes puntos:

6.3.1 Base de datos PostgreSQL

Como acabamos de comentar, la aplicación en su modo productivo utiliza una base de datos PosgreSQL, en nuestro caso incluye la versión 13 y el hash *sha256* correspondiente para garantizar la integridad de esta. Además, como hemos comentado anteriormente, hace uso del fichero *.env* para las variables del entorno, entre ellas nos encontramos *SQL_DATABASE*, *SQL_USER* y *SQL_PASSWORD* que son las variables de creación de la

base de datos, que incluyen los valores por defecto de esta. Por último, el volumen que comentaremos en un apartado posterior.

Por último, servicio completo lo podemos observar a continuación:

```
db:
  image: postgres:13@sha256:8f7c3c9b61d82a4a021da5d9618faf056633e089302a726d619fa467c73609e4
  env_file:
    - ./.env
  environment:
    POSTGRES_DB: ${SQL_DATABASE:-audit}
    POSTGRES_USER: ${SQL_USER:-postgres}
    POSTGRES_PASSWORD: ${SQL_PASSWORD:-postgres}
  volumes:
    - db-data:/var/lib/postgresql/data
  ports:
    - ${SQL_PORT:-5432}
```

6.3.2 Servidor NGINX

Tal y como hemos visto, el servidor nginx tiene la versión estable actual (1.18.0) y actúa como proxy de nuestra aplicación. Utiliza un volumen para el archivo de configuración del servidor y otro para poder servir los archivos estáticos de la aplicación, como son los estilos, js, imágenes etcétera.

```
nginx:
  image: nginx:1.18.0@sha256:6e4fc428f9f25f1914e43dc2e75ff3be5741417345091112 82a3a050a420d94c
  ports:
    - "8888:8888"
  volumes:
    - .:/app
    - ./nginx/app.conf:/etc/nginx/conf.d/app.conf
  depends_on:
```

```
    - web
```

6.3.3 Herramienta de auditorías

El servicio que tiene la herramienta de auditorías está compuesto del siguiente servicio:

```
  web:
    build:
      context: ./
    command: bash -c "python manage.py makemigrations && python manage.py migrate && python manage.py loaddata data && uwsgi --http 0.0.0.0:8000 --enable-threads --processes 2 --threads 2 --module app.config.wsgi"
    env_file:
      - ./.env
    volumes:
      - .:/app
    expose:
      - "8000"
    depends_on:
      - db
```

Este servicio levanta nuestra aplicación Python con varios comandos que veremos a continuación:

En primer lugar, crear y ejecutar las migraciones:

```
python manage.py makemigrations && python manage.py migrate
```

Después, cargar los archivos json con la información inicial de la aplicación:

```
python manage.py loaddata data
```

Para finalizar, lanza el servidor uwsgi que es el que levanta nuestra aplicación, en el puerto 8000 del contenedor.

Además, como podemos observar, realiza un build en el directorio actual, por defecto la imagen custom que utiliza es el Dockerfile que veremos a continuación.

Imagen de la aplicación

A continuación, se verá el Dockerfile con los comandos que incluye la imagen de Docker para poder configurar correctamente la herramienta:

```
FROM python:3.9.0-buster@sha256:8829824d85665db842f33f6a71960b00f3e3b329f297e499e24c748e29ae19f9

# Update and package installation
RUN apt-get update && \
   apt-get clean && \
   apt-get install -y ca-certificates-java --no-install-recommends && \
   apt-get clean

RUN apt-get update && \
   apt-get install -y openjdk-11-jdk p11-kit wkhtmltopdf && \
   apt-get install -y  && \
   apt-get clean && \
   update-ca-certificates -f

# Get JADX Tool
ENV JADX_VERSION 1.1.0

RUN \
   wget "https://github.com/skylot/jadx/releases/download/v$JADX_VERSION/jadx-$JADX_VERSION.zip" && \
   unzip "jadx-$JADX_VERSION.zip"

# Create a directory in the container in /app
RUN mkdir /app

# Copy the requirements to that directory
COPY requirements.txt /app

# Use /app as the workdir
WORKDIR /app
```

```
# Install python dependencies
RUN pip install -r requirements.txt

# Copy all to /app directory
COPY . /app/

# Encoding configuration
ENV LANG en_US.UTF-8
ENV LANGUAGE en_US:en
ENV PYTHONIOENCODING utf8

# Run the container as 1001 user
USER 1001

# Expose the 8000 port
EXPOSE 8000
```

Subida de apks

En nuestra aplicación, por defecto, subimos las apks al directorio *app/media/apk*, para modificar esta parte, deberemos cambiar el directorio de subida dentro de la clase Apk en el fichero *app/models.py*.

Logs de la aplicación

La configuración de los logs la podemos encontrar dentro de *app/config/settings.py* dentro del diccionario *LOGGING*. Por defecto, se muestran por consola y además se guardan dentro del fichero *app/logs/debug.log*.

6.3.4 Volúmenes

Como hemos comentado anteriormente, hacemos uso de volúmenes en Docker para garantizar la persistencia de los datos de nuestra aplicación. En esencia, es un archivo o directorio que permanece más allá de la vida útil de un contenedor, el cual, por la propia naturaleza de este, no permite guardar información y su tiempo de vida suele ser corto, debido a que se suelen escalar con frecuencia. Por lo que, para poder guardar la información de la herramienta, tenemos declarados los siguientes volúmenes:

- db-data

  Guarda toda la información de la base de datos.

- /nginx/app.conf:/etc/nginx/conf.d/app.conf

  El fichero de configuración de nginx se mapea desde el directorio en local *app/nginx* hasta el directorio de configuración de nginx por defecto en una máquina Linux que es */etc/nginx/conf.d*

- .:/app

  Se mapea toda la información local de la aplicación (desde la ruta raíz) a el directorio */app* del contenedor.

6.4 Despliegue de la aplicación

Para realizar el despliegue automático, se ha generado un script en el que se incluyen todos los comandos, pero vamos a verlos de uno en uno para realizar una instalación manual.

En primer lugar, podremos evitar este paso la primera vez que lancemos la aplicación, pero, en caso de que queramos modificar la imagen de la herramienta de auditorías a posteriori, deberemos hacer el build de esta mediante el siguiente comando:

```
docker-compose build
```

El comando principal que realizará el build (en caso de que no se haya ejecutado el comando anterior) y levantará los contenedores para que se inicialice la aplicación será el siguiente:

```
docker-compose up
```

Esto lo que hará es la creación de una red para que los contenedores puedan comunicarse, creará los volúmenes correspondientes y, una vez se termine la inicialización, la aplicación estará levantada en el puerto 8888 de nuestra máquina, por lo que podremos acceder a ella desde el navegador poniendo:

```
http://localhost:8888/
```

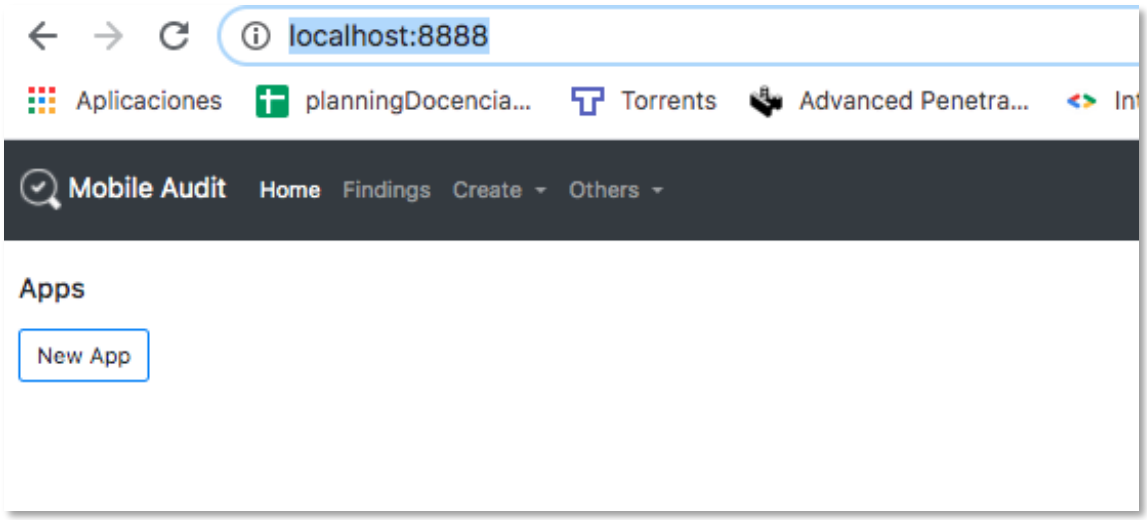


Figura 69 - Dashboard de inicio

Una vez desplegada, deberemos registrarnos en la aplicación, para ello podemos hacerlo desde el menú de la derecha haciendo clic en: *Do you need an account?*

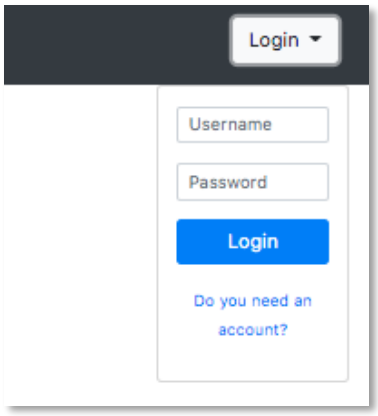


Figura 70 - Menú de login

La información necesaria para el registro es la que podemos ver en el siguiente formulario:

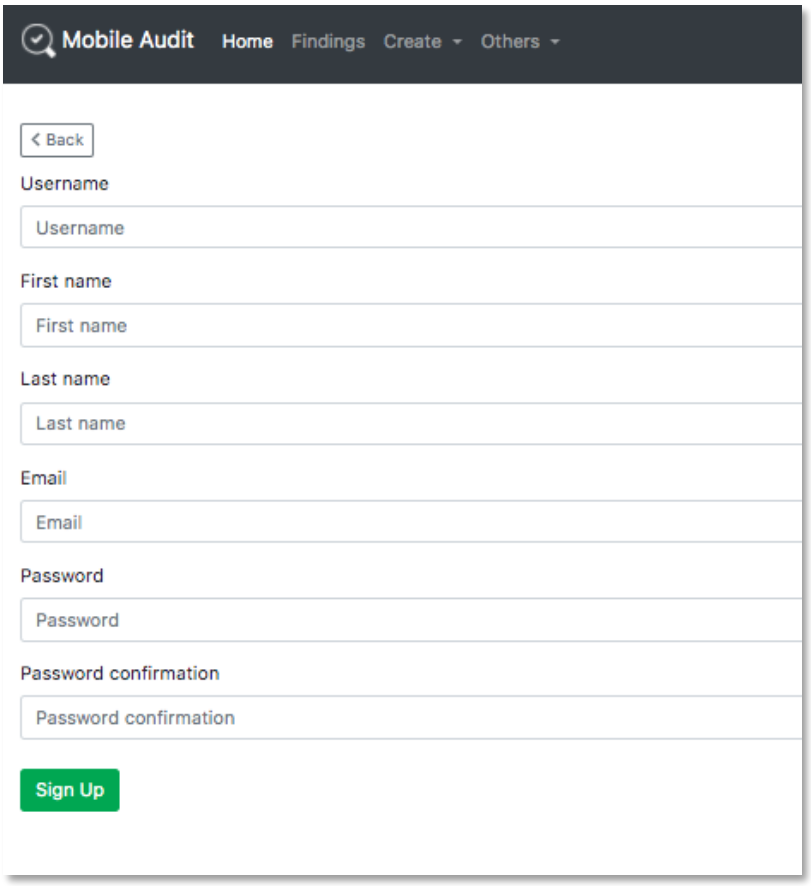


Figura 71 - Datos de registro de la aplicación

Una vez registrados, podremos ver a la derecha un menú para ver y editar nuestro perfil y para poder realizar el logout de la aplicación:

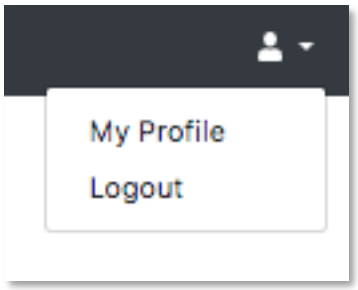


Figura 72 - Menú del usuario

A partir de aquí, ya tendremos la aplicación lista para empezar a hacer auditorías estáticas, para más información sobre las funcionalidades de la aplicación, podemos verlas a lo largo del capítulo 4

CAPÍTULO 7. CONCLUSIONES Y PROPUESTAS

En este capítulo comprobaremos si se han cumplido los objetivos propuestos y se realizará una valoración final de todo lo aprendido durante el desarrollo de la herramienta y, por último, se realizarán las propuestas de ampliación de cara a futuro.

7.1 CONCLUSIONES

Al inicio del trabajo, se plantearon unas premisas y unos objetivos bastante ambiciosos en cuanto a la herramienta, dado la idea a desarrollar era muy novedosa puesto que actualmente no había nada parecido en el mercado. Y, una vez desarrollado el trabajo, tras ver el resultado, ha sido bastante bueno respecto a las metas propuestas.

En cuanto a funcionalidades, la herramienta contiene todo lo que se planteó inicialmente e incluso se han ido añadiendo mejoras que se han identificado conforme avanzaba el desarrollo, por lo que se han creado esos nuevos casos de uso y se han implementado de manera eficiente.

Una de las conclusiones principales en cuanto a la implementación ha sido que Android es una plataforma muy usada y a la vez muy poco protegida por defecto. Durante el desarrollo, se han realizado numerosas auditorías de aplicaciones, incluyendo muestras de malware y aplicaciones conocidas y, podemos verificar, que, con la herramienta desarrollada se pueden encontrar numerosas evidencias para verificar la seguridad de estas.

Por lo tanto, con todo ello, el trabajo realizado ha sido satisfactorio, tanto en aprendizaje, sobre los numerosos riesgos de estas aplicaciones, como en cómo evitarlos mediante la implementación de las mejores prácticas de codificación segura.

7.2 Trabajo futuro y posibles ampliaciones

Para finalizar con este último capítulo, comentaremos algunas de las propuestas que se podrían añadir, al trabajo actual, en trabajos futuros:

- En primer lugar, es necesario modificar la aplicación para que se cargue de manera dinámica, así evitar esperas y la necesidad de que el usuario tenga que recargar la página con frecuencia.
- Por otro lado, el modelo de datos, reglas, mitigaciones puede ser pulido y ampliado para emplear más y mejores reglas y así evitar la numerosa cantidad de falsos positivos.
- Además, se ha puesto un empeño muy grande en la generación del modelo de datos, para que pueda ser escalable y se guarde la mayor información posible, por lo que sería interesante permitir la exportación de esta en más formatos, como pueden ser CSV, json, doc etcétera.
- Para finalizar, otra de las funcionalidades a implementar de cara a futuro es añadir un API para la interacción de otras herramientas con nuestra aplicación, y proveer los resultados de manera que puedan ser consumidos de esta de manera sencilla.

CAPÍTULO 8. BIBLIOGRAFÍA

[1] «Cuota de mercado de Android en España,» [En línea]. Available: https://gs.statcounter.com/os-market-share/mobile/spain.

[2] «Cuota de mercado global Android,» [En línea]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[3] «Componentes principales en Android,» [En línea]. Available: https://developer.android.com/guide/components/fundamentals.html.

[4] «Application Fundamentals,» [En línea]. Available: https://developer.android.com/guide/components/fundamentals.

[5] «Androguard Documentation,» [En línea]. Available: https://androguard.readthedocs.io/en/latest/.

[6] «Android Intent Filters,» [En línea]. Available: https://developer.android.com/guide/components/intents-filters.

[7] «Malware Domain List,» [En línea]. Available: https://www.malwaredomainlist.com/.

[8] «APK Pure,» [En línea]. Available: https://apkpure.com/es/.

[9] «APK Downloader,» [En línea]. Available: https://apps.evozi.com/apk-downloader/.

[10] https://docs.djangoproject.com/en/3.1/, «Django Documentation,» [En línea].

[11] «VirusTotal,» [En línea]. Available: https://www.virustotal.com/.

[12] «wkhtmltopdf docs,» [En línea]. Available: https://wkhtmltopdf.org/docs.html.

[13] «pdfkit,» [En línea]. Available: https://pypi.org/project/pdfkit/.

[14] «Listado de permisos en Android,» [En línea]. Available: https://gist.github.com/Arinerron/1bcaadc7b1cbeae77de0263f4e15156f.

[15] «Listado de permisos en Android,» [En línea]. Available: https://gist.github.com/Arinerron/1bcaadc7b1cbeae77de0263f4e15156f.

[16] «Manifest Permission,» [En línea]. Available: https://developer.android.com/reference/android/Manifest.permission.

[17] «OWASP Mobile Top 10,» [En línea]. Available: https://owasp.org/www-project-mobile-top-10/.

CAPÍTULO 9. CONTENIDO DEL ENTREGABLE

En el contenido del entregable que acompaña a la memoria podemos encontrar los siguientes recursos:

- Memoria del trabajo en los formatos PDF, DOCX y DOC dentro del directorio Memoria.
- Código fuente del trabajo dentro del directorio Código.
- Páginas Web que han servido de bibliografía. Las podemos encontrar dentro de la memoria: Bibliografia.
- Manual o Readme de usuario de la aplicación, que podemos encontrar en el directorio Manual, en formato Markdown y en inglés (el manual en español lo encontraríamos en esta misma memoria).