Análisis de Vulnerabilidades en Aplicaciones de Rastreo de Covid-19 basadas en el Protocolo DP3T

Vulnerabilities Analysis in COVID-19 Tracking Applications based on the DP3T Protocol

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA DE COMPUTADORES
CURSO 2020–2021

Alonso Martín Zurdo
Pablo Izquierdo Garbajosa

Directores
Francisco Javier Crespo Yáñez
Luis Javier García Villalba

Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

Madrid, Junio de 2021

Índice General

Índice de Figuras

Índice de Tablas

Lista de Acrónimos

API	*Application Programming Interface*
APK	*Android Application Package*
APT	*Advanced Persistent Threat*
BLE	*Bluetooth Low Energy*
C2D	*Cloud to Device Messaging*
COVID-19	*Corona Virus Disease*
CVSS	*Common Vulnerability Scoring System*
CWE	*Common Weakness Enumeration*
CWSS	*Common Weakness Scoring System*
DNS	*Domain Name Service*
DoS	*Denial of Service*
DP-3T	*Decentralized Privacy-Preserving Proximity Tracing*
EphID	*Ephimeral Identifier*
FIRST	*Forum of Incident Response and Security Teams*

GPS	*Global Positioning System*
ID	Identificador
IP	*Internet Protocol*
MASVS	*Mobile Application Security Verification Standard*
MobSF	*Mobile Security Framework*
OWASP	*Open Web Application Security Project*
PACT	*Private Automated Contact Tracing*
PEPP-PT	Rastreo Paneuropeo de Proximidad para Preservar la Privacidad
PEPP-PT-NTK	*Pan-European Privacy-Preserving Proximity Tracing Need To Know*
ROBERT	*ROBust and privacy-presERving proximity Tracing*
SQL	*Structured Query Languaje*
TCN	*Temporary Contact Numbers*
TI	Tecnología de la Información
UE	Unión Europea
URL	*Uniform Resource Locator*
XSS	*Cross-Site Scripting*

Abstract

Due to the global pandemic caused at the beginning of 2020 by the *Corona Virus Disease* (COVID-19), we have seen the raise of the number of people infected by the virus growing at a very high speed, provoking multiple saturations in hospitals, many deaths and leading many countries to impose home confinement for months. To control the infections and act quickly in case of infections by the virus and prevent its spread, a key job arised, the tracker, which is a person who is dedicated to investigating the close contacts a person who has been tested positive for COVID-19 so they can be warned to do a home quarantine and carry out the corresponding tests. This manual scan becomes very ardous work due to the rapid spread os the virus, becoming ineffective. For this reason, the idea of creating a mobile application to make more effiecient digital contact tracing arosed. However, tracking through a mobile application arouses in users concern and fear that with this system their privacy an intimacy wil be violated and so their data will be exposed to potencial hackers, causing users not to use this system and so this will make a loose of effectiveness of contact tracing. In this work, an analysis of vulnerabilities present in mobile contract tracing applications based on the *Decentralized Privacy-Preserving Proximity Tracing* (DP-3T) protocol is carried out using an automatic vulnerability analysis tool for mobile applications. Specifically, a series of analyzes (static and dymamic) are carried out on the applications based on DP-3T protocol developed in the Unión Europea (UE). A total of 18 applications have been selected according to their availability and possibility of instalation. The results of the analysis had been quite improvable in most of applications with a 47.7 of average score where more than 90 % have turned out to be insecure in the version this analysis has been done. All of this can be as a result of the short development time developers may have had leading to some decisions not have been the most appropiate when it comes to preserving security but as making it functional applications.

Keywords: dynamic analysis, static analysis, mobile applications, COVID-19, DP-3T, security, vulnerabilities.

Resumen

Debido a la pandemia mundial provocada por el COVID-19 a principios de 2020, vimos como el número de personas contagiadas por este virus crecía a una velocidad muy alta, causando la saturación de los hospitales, muchos fallecimientos y llevando a muchos países a imponer un confinamiento domiciliario durante meses. Para controlar los contagios y actuar de manera rápida ante un caso de una persona contagiada por el virus y evitar su expansión, surge un trabajo clave que es el rastreador, que es una persona que se dedica a investigar los contactos cercanos que ha tenido una persona que ha dado positivo en COVID-19 para avisarles que hagan una cuarentena domiciliaria y se realicen las pruebas correspondientes. Este rastreo manual se vuelve un trabajo muy arduo debido a la rápida expansión del virus dejando de ser efectivo. Por este motivo surge la idea de crear una aplicación móvil para hacer un rastreo digital de contacto más efectivo. Sin embargo, el rastreo mediante una aplicación móvil, despierta en los usuarios la preocupación y el temor de que con este sistema se violen su privacidad e intimidad y que sus datos queden expuestos a posibles hackers, provocando que los usuarios no usen el sistema y con ello se pierda la efectividad del rastreo de contactos. En este trabajo se realiza un análisis de las vulnerabilidades presentes en las aplicaciones móviles de rastreo de contactos que se basan en el protocolo DP-3T utilizando una herramienta de análisis automático de vulnerabilidades para aplicaciones móviles. Específicamente, se realiza una serie de análisis (estático y dinámico) a las aplicaciones basadas en el protocolo DP-3T desarrolladas en la UE. En total se han seleccionado un total de 18 aplicaciones según su disponibilidad y posibilidad de instalación. El resultado de los análisis ha sido bastante mejorable en la mayoría de las aplicaciones con una media de puntuación de 47.7 donde más del 90 % han resultado ser inseguras en la versión con la que se ha realizado este trabajo. Todo esto puede venir a raíz del escaso tiempo de desarrollo que puede haber llevado a la toma de decisiones que no hayan sido las más adecuadas a la hora de preservar la seguridad tanto como de hacer una aplicación funcional.

Palabras Clave: Análisis dinámico, análisis estático, Aplicaciones móviles, COVID-19, DP-3T, seguridad, vulnerabilidades.

Capítulo 1

Introducción

1.1. Motivación

Debido a la pandemia mundial provocada por el COVID-19 a principios de 2020, vimos como el número de personas contagiadas por este virus crecía a una velocidad muy alta, causando la saturación de los hospitales, muchos fallecimientos y llevando a muchos países a imponer un confinamiento domiciliario durante meses.

Para controlar los contagios y actuar de manera rápida ante un caso de una persona contagiada por el virus y evitar su expansión, surge un trabajo clave que es el rastreador, que es una persona que se dedica a investigar los contactos cercanos que ha tenido una persona que ha dado positivo en COVID-19 para avisarles y que hagan una cuarentena domiciliaria y se realicen las pruebas correspondientes.

Debido a la rápida expansión del virus, el rastreo manual se vuelve un trabajo muy arduo e imposible de conseguir, por lo que deja de ser efectivo, ya que, por ejemplo, si una persona contagiada ha estado usando el transporte público es imposible contactar con todas esas personas con las que ha compartido vagón. Por este motivo surge la idea de crear una aplicación móvil para hacer un rastreo digital de contacto más efectivo.

Pero con el rastreo mediante una aplicación móvil, surge, en las personas, la preocupación y el temor de que con este sistema se viole su privacidad e intimidad y sus datos queden expuestos a posibles hackers, provocando que los usuarios no hagan uso del sistema y con ello se pierda la efectividad del rastreo de contactos.

1.2. Objetivos

El objetivo de este trabajo es analizar que tipo de vulnerabilidades pueden tener las aplicaciones móviles para el rastreo de contactos que se basan en el protocolo DP-3T. Para ello se hace uso de una herramienta de análisis automático de vulnerabilidades para aplicaciones móviles, con la que se hace un análisis estático y dinámico de las aplicaciones que nos ayudará a comprobar los diferentes tipos de vulnerabilidades que pueden tener

dichas aplicaciones y si se pone en riesgo la seguridad y privacidad de los usuarios que las usan.

1.3. Plan de Trabajo

En la Tabla 1.1 se resumen las tareas realizadas a lo largo del desarrollo del proyecto. Adicionalmente, se adjunta en la Figura 1.1 un diagrama de Gantt que ilustra las tareas y su temporalidad.

Identificador	Tarea
1	Investigación
1.1	Lectura e Investigación de artículos académicos
2	Documentación
2.1	Busqueda de información sobre el protocolo DP-3T
2.2	Busqueda de información sobre el análisis de penetración y distintas herramientas
3	Busqueda del entrono de trabajo
3.1	Prueba de distintas herramientas
3.2	Configuración del entorno de trabajo final
4	Experimentación
4.1	Obtención de APK de las aplicaciones
4.2	Análisis estático
4.3	Análisis de los resultados
5	Memoria
5.1	Anotaciones, citas y recopilación de artículos
5.2	Redacción de la memoria

Tabla 1.1: Tareas Realizadas a lo largo del Proyecto

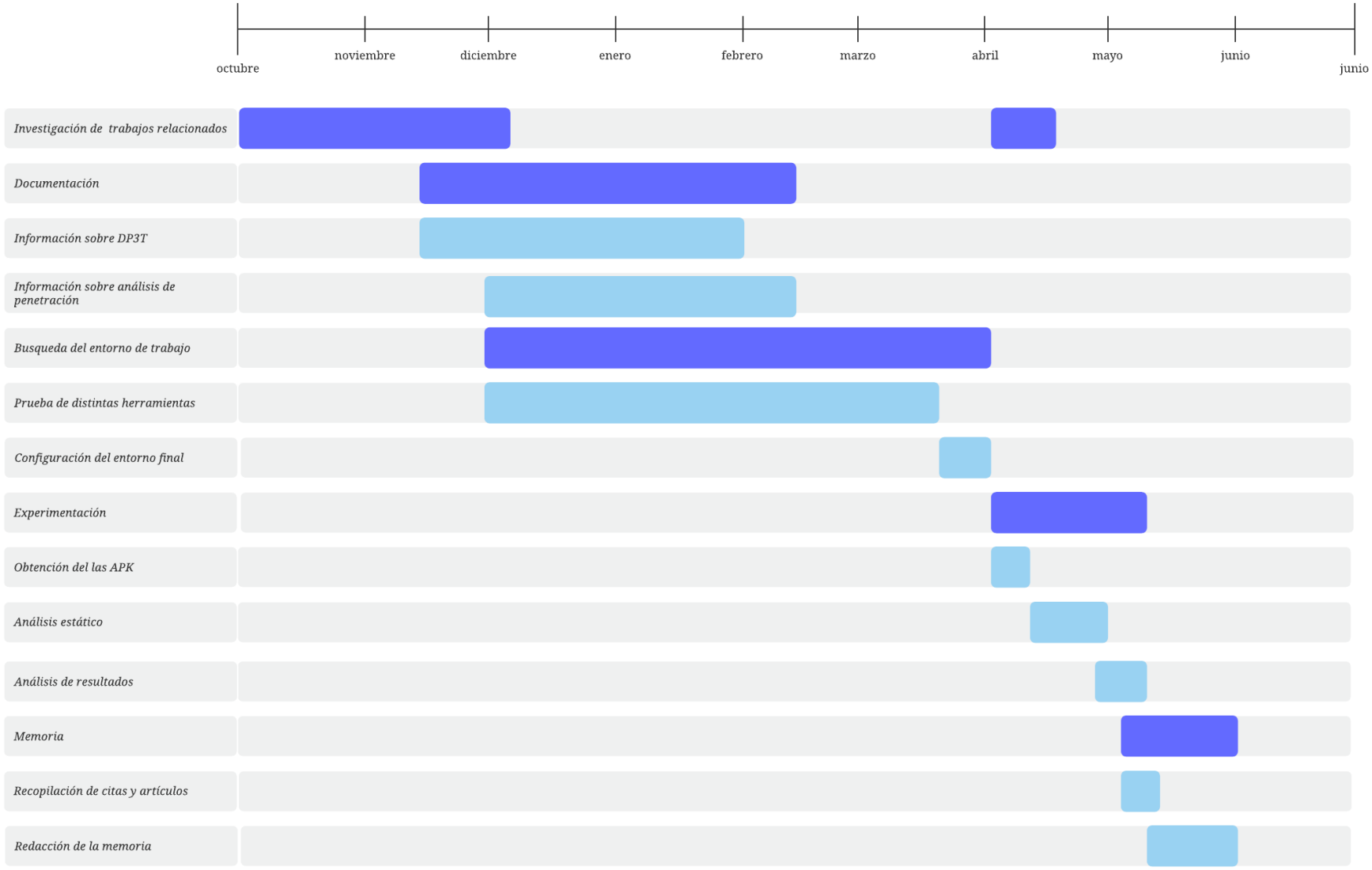


Figura 1.1: Diagrama de Gantt

1.4. Estructura del Trabajo

El resto del trabajo está organizado en siete capítulos con la estructura que se comenta a continuación:

El Capítulo 2 contextualiza e introduce brevemente los orígenes, conceptos y bases teóricas de las aplicaciones de rastreo de contactos de manera general.

En el Capítulo 3 se habla más en detalle del protocolo de rastreo de proximidad descentralizado. Se describen sus tipos y su funcionamiento además de analizar y comparar su privacidad y seguridad entre los distintos modelos que existen.

En el Capítulo 4 contextualiza al funcionamiento de las herramientas de análisis de vulnerabilidades. Se repasan los distintos tipos y se explican en detalle los estándares que usa la herramienta utilizada para hacer el análisis de las aplicaciones.

El Capítulo 5 presenta distintos trabajos relacionados con este, explicando los resultados que se obtuvieron.

En el Capítulo 6 se detallan los resultados de los análisis realizados a las distintas aplicaciones europeas y como está configurado el laboratorio de análisis.

El Capítulo 7 resume las aportaciones individuales de cada integrante del equipo de trabajo.

El Capítulo 8 concluye este trabajo: resumiendo brevemente la experiencia del proyecto y analizando los resultados de los análisis. Se presenta el trabajo futuro y las posibles líneas de investigación.

Los Capítulos 9 y 10 son traducciones al inglés de los Capítulos 1 y 8, respectivamente.

Capítulo 2

Aplicaciones de Rastreo de Contactos

El rastreo de contactos es uno de los métodos utilizados para contener una epidemia médica. Al rastrear a los humanos expuestos a una persona infectada, se puede reducir la propagación de infecciones si esas personas potencialmente infectadas pueden aislarse de la población restante. Además, el rastreo de contactos ayuda a rastrear las áreas que están expuestas a una infección [HWMF21]. A raíz del brote pandémico de la COVID-19 surgieron diferentes enfoques que presentaban soluciones tecnológicas para mejorar la detección y rastreo temprano de posibles brotes de COVID-19. Sobre éstas tecnologías surgieron muchas dudas sobre la seguridad y la privacidad de los usuarios. Esto abrió un debate sobre el tipo de modelo a seguir, por una parte un modelo centralizado y por otra parte un modelo descentralizado. En la Sección 2.1 se habla del origen de las aplicaciones de rastreo de contacto, en la Sección 2.2 se describe el objetivo y los tipos de aplicaciones de rastreo de contactos, las características de los modelos descentralizados de las aplicaciones de rastreo de contactos se describe en la Sección 2.3, las características de los modelos centralizados se describen en la Sección 2.4. Finalmente, en la Sección 2.5, se presentan los principales tipos de adversarios que se pueden encontrar.

2.1. Origen

La pandemia de la COVID-19 se ha extendido muy rápidamente. Muy pocos países han logrado mantenerlo bien controlado, pero una de las principales herramientas que utilizan varios de estos países es el rastreo de contactos. Específicamente, siempre que una persona es diagnosticada de COVID-19, cada persona que estuvo en contacto con esa persona infectada es avisada y se le dice que se ponga en cuarentena. Al principio el rastreo de contactos podía hacerse manualmente, pero con miles de casos el rastreo de contactos manual se vuelve mucho más difícil [CIY20]. A raíz de esta dificultad, los gobiernos de los países, para ayudar a las autoridades sanitarias nacionales, decidieron implementar

aplicaciones móviles de rastreo de contactos para automatizar y facilitar la labor de rastrear a las personas que hayan estado en contacto con una persona contagiada.

2.2. Tipos de Aplicaciones de Rastreo

El principal objetivo de este tipo de aplicaciones es detectar lo antes posible nuevas fuentes potenciales de infección, de modo que la propagación de COVID-19 pueda mitigarse rápidamente. Hasta ahora se pueden encontrar dos tipos de aplicaciones móviles [MKHR+20].

- **Aplicación de seguimiento de contactos**: Se basa en los marcos de rastreo de contactos digitales descentralizados y centralizados que se basan en tecnología inalámbrica de proximidad como *Bluetooth Low Energy* (BLE). Cuando dos usuarios están físicamente cerca, los smartphones se envían su identidad en términos de *Ephimeral Identifier* (EphID) entre sí. Cada smartphone registra todos sus encuentros que ocurrieron dentro de un período de tiempo.

- **Aplicación para compartir ubicación**: Se basa en la información de posicionamiento del smartphone, es decir, mediante rastreo GPS o mapeo de torres de telefonía móvil. Para una aplicación de este tipo, el usuario debe aceptar que su smartphone envíe de forma regular su posición a un servidor central, que puede asignar, durante un período de tiempo ilimitado, a cada usuario. Si un usuario informa una infección por COVID-19, entonces se advierte de la situación a todos los usuarios que estuvieron cerca del usuario infectado.

2.3. Modelos Descentralizados

Actualmente el debate sobre los sistemas de rastreo de contactos se centra en soluciones basadas en tecnología Bluetooth. En todos los sistemas, la aplicación emite frecuentemente un EphID, al mismo tiempo, la aplicación recopila los EphID que recibe de la aplicación de otros usuarios cercanos, gestionando dos listas de identificadores, los enviados y los recibidos. Los identificadores se almacenan indicando cuándo se enviaron y recibieron, con bastante frecuencia, los identificadores antiguos son eliminados de las listas [Vau20].

En los modelos descentralizados todo se genera localmente en el smartphone del usuario. Cuando un usuario es diagnosticado, recibe de la autoridad sanitaria un token que le permite cargar información en un backend central. Esta operación requiere su consentimiento y el usuario puede tener un control sobre qué cargar [Vau20]. Algunos de los métodos descentralizados son:

- ***Private Automated Contact Tracing* (PACT)(MIT)**: PACT es una colaboración liderada por el Laboratorio de Ciencias de la Computación e Inteligencia

Artificial del MIT. Es un enfoque simple y descentralizado para el uso de dispositivos de comunicación digital personal para la automatización de la detección de exposición mediante la señalización BLE [RWI+20].

- ***Temporary Contact Numbers* (TCN) Coalition**: Es una comunidad global de tecnólogos unidos por la misión de apoyar aplicaciones de notificación de exposición durante la pandemia de COVID-19 [JD20].
- **DP-3T**: Este sistema proporciona una base tecnológica para ayudar a frenar la propagación del COVID-19 simplificando y acelerando el proceso de notificación de personas que podrían haber estado expuestas al virus para que puedan tomar las medidas para romper su cadena de transmisión [TPH+20]. Este protocolo se detalla más ampliamente en el capítulo 3.
- **OpenCovidTrace**: Su objetivo es integrar los protocolos de rastreo de contactos más populares basados en BLE y proporcionar funciones adicionales además de ellos. Tales protocolos incluyen DP-3T, Google/Apple, Exposure Notification y BlueTrace. Sigue las especificaciones DP-3T originales [MKHR+20].
- **Whisper Tracing**: Las identificaciones temporales se generan periódicamente en el smartphone del usuario y se intercambia con smartphones cercanos a través de BLE. Cuando un usuario está infectado, la aplicación carga a un servidor central las semillas que se utilizaron para producir los Identificador (ID) temporales de las últimas dos semanas.

2.4. Modelos Centralizados

Al contrario que en el modelo descentralizado, es el servidor central el que estima la probabilidad de exposición, en lugar de ser el smartphone localmente. Es el servidor el que notifica a los usuarios del riesgo o éstos consultan el servidor sobre su estado. El servidor central tiene un pseudo-identificador a largo plazo para cada usuario y lo usa para derivar los EphID que se envían a los smartphones. En caso de un diagnóstico positivo, los usuarios pueden dar permiso para que su móvil envíe la lista registrada de observaciones al servidor para permitir el rastreo de proximidad. A continuación se describen los protocolos más conocidos [TPH+20]:

- ***Pan-European Privacy-Preserving Proximity Tracing Need To Know* (PEPP-PT-NTK) y OpenTrace**: Ejecutan el proceso de rastreo de proximidad después de que un usuario diagnosticado haya subido su lista de observaciones [(EphID, tiempo, duración)] para la ventana contagiosa. El backend recupera los pseudo-identificadores a largo plazo de los usuarios en riesgo de los EphID informados y activa un proceso para notificarles si su exposición es lo suficientemente alta, como se muestra en la Figura 2.1.

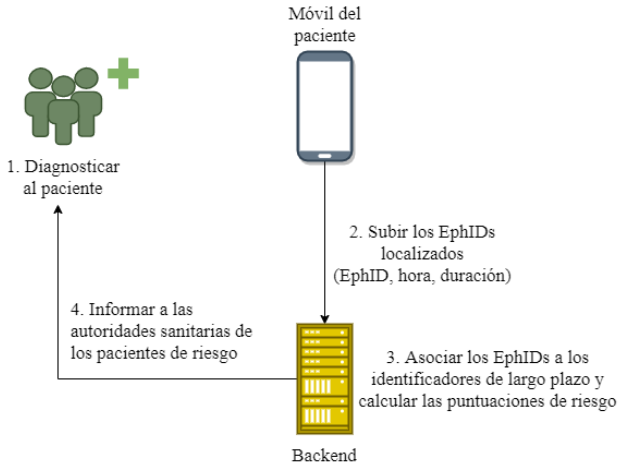


Figura 2.1: Rastreo de proximidad para diseños centralizados PEPP-PT-NTK y OpenTrace.

- ***ROBust and privacy-presERving proximity Tracing* (ROBERT)**: El backend rastrea la exposición de cada usuario del sistema. El servidor asocia estos EphID observados a largo plazo con pseudo-identificadores de usuarios en riesgo. No notifica a los pacientes, los usuarios deben consultar el servidor regularmente para saber su estado de exposición.

- **DESIRE**[CBB+20]: Generan identificadores en el smartphone, sin dejar de estimar la exposición de forma centralizada. En lugar de transmitir identificadores efímeros e independientes, los smartphones en DESIRE transmiten claves públicas efímeras que, cuando se combinan con las claves públicas de otros smartphones, producen el EphID. En caso de un diagnóstico positivo, los usuarios pueden dar permiso para que su smartphone envíe una versión de los EphID observados al backend para permitir el rastreo de proximidad.

2.5. Tipos de Adversarios

Los adversarios son personas, grupos u organizaciones que intentan comprometer la seguridad/privacidad del servicio de rastreo de contactos o interrumpir su funcionamiento. Los adversarios podrían intentar las siguientes acciones para atacar un sistema de rastreo de contratos digitales [MKHR+20]:

- Interceptar, bloquear, modificar, inyectar o reproducir cualquier mensaje en el canal de comunicación pública.

- Utilizar la aplicación móvil para acceder al sistema y habilitar o deshabilitar el servicio de notificación de la aplicación a voluntad.

- Cuando corresponda, el mismo adversario puede intentar registrarse en el servicio varias veces, es decir, crear varios perfiles.
- El mismo adversario puede llevar varios dispositivos e instalar la(s) aplicación(es) en cada uno de ellos.
- El mismo adversario puede intentar instalar la aplicación legítima junto con las personalizadas en el mismo dispositivo.
- Instalar antenas de alta potencia para amplificar su señal de recepción y transmisión para cubrir un área más amplia con el propósito de aumentar su capacidad de seguimiento o transmisión.
- Acceder a los datos almacenados en el dispositivo.
- Causar *Denial of Service* (DoS), conmoción en el sistema, acoso al usuario final o contaminar los datos proporcionados al sistema.
- Configurar y operar su propio servidor backend malicioso.
- Tener acceso al código fuente del servidor backend.
- Aliarse con otros usuarios finales, personas que trabajan para las autoridades sanitarias o policiales, o administradores del backend.
- Comprometer un servidor backend y la infraestructura de la Tecnología de la Información (TI) subyacente.
- Engañar/atraer a los usuarios finales para que instalen malware en sus dispositivos.

Tanto en los modelos centralizados como en los modelos descentralizados se pueden encontrar diferentes tipos de adversarios maliciosos que se describen a continuación [TPH+20]:

- **Usuarios regulares**: Miran exclusivamente la información disponible a través de la interfaz de usuario de la aplicación para inferir información privada sobre otros usuarios.
- **Tech-savvy user**: Tiene acceso al sistema a través de la aplicación móvil. Además, puede configurar (BT, WiFi y smartphones) para escuchar a escondidas localmente. Finalmente, puede descompilar/modificar la aplicación y tener acceso al código fuente del backend.
- **Espía**: Puede observar la comunicación de la red y/o mensajes de difusión e intentar rastrear a las personas.
- **Autoridad sanitaria**: Tienen información de personas en riesgo sólo cuando la misma persona en riesgo lo notifica.

- **Desarrolladores**: Pueden acceder a los datos de los servidores. También pueden cambiar el software backend y el de las aplicaciones móviles.
- **Adversario a nivel estatal**: Combina las capacidades del espía y del Tech-savvy user. Su objetivo es obtener información sobre la población o individuos particulares.
- **Adversario de presupuesto ilimitado**: Por ejemplo, grandes organizaciones y estados nacionales extranjeros. Tiene las capacidades del tech-savvy user pero a una escala mayor. Los objetivos pueden ser aprender información sobre la población o un ataque DoS. Una forma de alterar el sistema es mediante el envío de notificaciones falsas.

Capítulo 3

Protocolo de Rastreo de Proximidad Descentralizado para Preservar la Privacidad

El 1 de Abril de 2020 nace un consorcio llamado Rastreo Paneuropeo de Proximidad para Preservar la Privacidad (PEPP-PT) para ofrecer soluciones de rastreo de proximidad centralizadas y descentralizadas. Como solución descentralizada se crea un proyecto llamado Seguimiento de Proximidad Descentralizado para Preservar la Privacidad (DP-3T), dirigido por un equipo de expertos liderado por Carmela Troncoso. Pasado un tiempo, el equipo de DP-3T abandona el proyecto PEPP-PT alegando que no eran transparentes. En este capítulo se presenta el protocolo DP-3T. En la Sección 3.1 se describe su funcionamiento, las características más relevantes de privacidad y seguridad de este protocolo se analizan en la Sección 3.2. Finalmente, se presentan las principales diferencias del protocolo DP-3T con los distintos modelos existentes [TPH+20].

3.1. Funcionamiento de DP-3T

El protocolo DP-3T se basa en la transmisión de EphID a través de BLE desde el smartphone del usuario, con lo que se pueden recibir y almacenar los EphID de usuarios cercanos. Cuando un usuario da positivo en COVID-19 envía sus EphID al backend y éste a los usuarios del sistema, se compararan con los EphID almacenados y si coincide con alguno, el sistema avisa al usuario de que ha estado en contacto con una persona positiva en COVID-19 [MKHR+20]. Se proponen 3 protocolos [TPH+20].

3.1.1. Rastreo de Proximidad Descentralizado de Bajo Consumo

Este protocolo tiene buenas propiedades de privacidad y pequeños requisitos de ancho de banda. Cada día el smartphone genera una semilla aleatoria y junto un hash se generar el EphID. Para cada señal recibida se guarda el EphID Bluetooth recibido, la medida de exposición y el día que se recibió. Como se pueden recibir varias veces el EphID, para un almacenamiento eficiente se agrupan por EphID. Cuando un usuario es positivo, envía al servidor la semilla y el día en el que se considera que empezó a ser contagioso. Después de reportarlo, el móvil lo borra. El servidor guarda una dupla con el EphID y el día, y ésta dupla es la que se descargan los smartphones para comprobar la exposición con un positivo, como se muestra en la Figura 3.1.

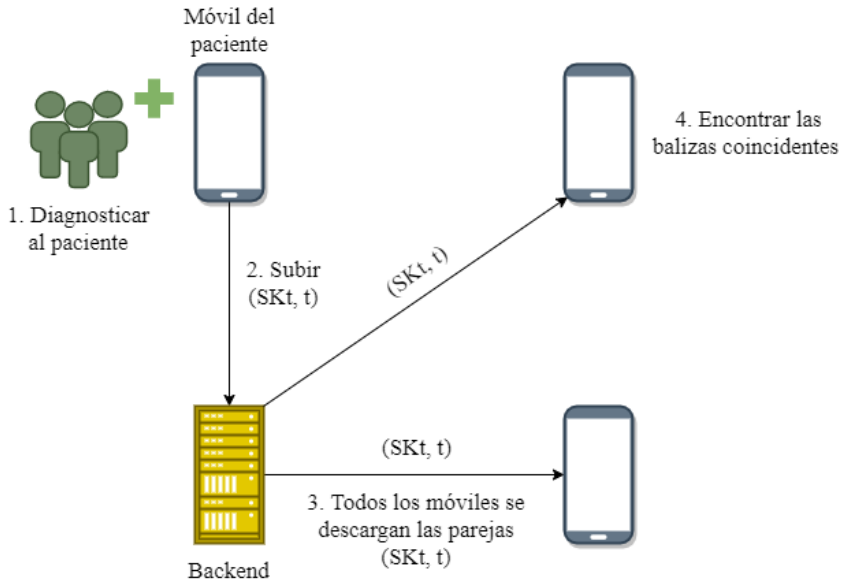


Figura 3.1: Proceso de rastreo de proximidad

3.1.2. Ratreo de Proximidad Descentralizado no Vinculable

Este diseño ofrece mejor privacidad que el de bajo consumo, pero a costa de aumentar el ancho de banda. Utiliza un filtro Cuckoo para almacenar los EphID. Este diseño previene que un atacante vincule los EphID de los positivos y permite al usuario positivo descartar lugares y periodos de tiempo donde sabe que no ha tenido contacto con nadie. Periódicamente el servidor crea un nuevo filtro Cuckoo. El filtro Cuckoo tiene muy baja probabilidad de falsos positivos. A diferencia del diseño de bajo consumo, cuando un usuario da positivo, puede excluir periodos de tiempo donde sabe que no ha tenido contacto con nadie. Este diseño necesita más banda ancha y almacenamiento que el diseño de bajo consumo. El costo computacional es probablemente menor que en el de bajo consumo.

3.1.3. Rastreo de Proximidad Descentralizado Híbrido

Este diseño combina ideas de los otros dos diseños. Los móviles generan una semilla aleatoria por cada ventana de tiempo y la usa de forma similar al diseño de bajo consumo. Los usuarios sólo cargan semillas si la exposición con otros usuarios es relevante. Este diseño es similar al diseño que usan Google y Apple. Los diseños de Google y Apple usan una ventana de tiempo de 1 día, pero nosotros recomendamos que sea de 2 o 4 horas. La información que almacenan los móviles es similar a la del diseño de bajo consumo. Requiere más banda ancha y almacenamiento que el diseño de bajo consumo, pero menos que el diseño no vinculable. En caso de positivo, el funcionamiento es similar al del diseño no vinculable.

3.2. Riesgos de Privacidad y Seguridad

Esta sección analiza las diferentes preocupaciones sobre la privacidad y la seguridad [TPH+20] de los tres protocolos propuestos en la Sección 3.1, teniendo en cuenta los diferentes tipos de adversarios, que se encuentran descritos en la Sección 2.2.

3.2.1. Análisis de Privacidad

Las principales preocupaciones sobre la privacidad son [TPH+20]:

- **El gráfico social**: Describe las relaciones entre usuarios. Un sistema de rastreo de proximidad no necesita proporcionar esta información.
- **El gráfico de interacción**: Refleja interacciones físicas de corto alcance entre usuarios y tampoco debe proporcionar esta información el sistema.
- **La trazabilidad de la ubicación**: Para hacer el rastreo de proximidad tampoco es necesario.
- **Individuos en riesgo**: Las personas en riesgo deben saber que han estado expuestos al virus para que puedan tomar las medidas adecuadas. Ninguna otra parte en el sistema necesita saber esta información.
- **Estado del positivo en COVID-19**: Sólo el usuario y las autoridades sanitarias necesitan saber que el usuario ha sido diagnosticado como positivo.
- **Ubicaciones expuestas**: El sistema no necesita revelar ninguna ubicación que los positivos hayan visitado.

3.2.1.1. Privacidad del Diseño de Bajo Consumo

A continuación se analiza la privacidad del diseño de bajo consumo a partir de las preocupaciones vistas anteriormente.

- **Gráfico social**. Este diseño no revela ninguna información sobre el gráfico social a ninguna entidad.
- **Gráfico de interacción**. El sistema no revela ninguna información sobre la interacción entre dos usuarios a ninguna entidad.
- **Trazabilidad de la ubicación**. En cuanto a la trazabilidad de la ubicación, los espías, los Tech-savvy user y los adversarios a nivel estatal pueden hacer, localmente, un seguimiento de pacientes infectados durante la ventana en la que el usuario ha notificado el positivo.
- **Individuos en riesgo**. No da información sobre personas en riesgo a ninguna parte que no sean las personas en riesgo.
- **Estado del positivo en COVID-19**. Un usuario malicioso que sólo usa la interfaz estándar de la aplicación no puede saber que contacto ha dado positivo porque el sistema no revela esa información salvo que la haya expuesto el propio usuario contagiado. Sólo puede saberlo de forma externa a la aplicación, como que el usuario le informe o el usuario malicioso le vea entrando en el hospital.Para revelar el estado de un individuo positivo el adversario debe registrar a quien vio y cuando, registrar muchas cuentas en el sistema y utilizar estas cuentas en un periodo corto de tiempo. Debido a la vinculabilidad de EphID de este diseño, un atacante puede combinar observaciones en diferentes momentos para identificar quien informo de un positivo al sistema. El adversario puede vincular la dirección *Internet Protocol* (IP) de los usuarios, para reducir el riesgo recomendamos que no se registren las direcciones IP. En la configuración actual, los atacantes retroactivos pueden vincular las balizas recibidas en tiempos diferentes y aproximados para ayudar a identificar a los usuarios positivos de COVID-19. La cantidad de información disponible para dicho atacante se puede reducir ejecutando el rastreo de proximidad dentro de un módulo de nivel de sistema operativo privilegiado o dentro de una ejecución confiable local (TEE).
- **Ubicaciones expuestas**. Un Tech-savvy user puede saber si un usuario positivo ha visitado una ubicación en un pequeño radio de 50 metros. También puede saber cuántas personas han visitado dicha ubicación.

3.2.1.2. Privacidad del Diseño no Vinculable

A continuación se analiza la privacidad del diseño no vinculable a partir de las preocupaciones vistas anteriormente.

- **Gráfico social y de interacción**. Brindan el mismo nivel de protección para el gráfico social y el gráfico de interacción.
- **Trazabilidad de la ubicación**. Los EphID permanecen desvinculados contra un atacante local. Sin embargo, un servidor malintencionado puede cargar EphID efímeros.
- **Individuos en riesgo**. Al igual que el diseño de bajo consumo, las semillas de un usuario positivo son independientes de sus contactos. Por lo tanto, no dan información sobre personas en riesgo.
- **Estado del positivo en COVID-19**. Para el estado de un positivo, este diseño reduce la vinculabilidad de los EphID de los usuarios positivos. Esto reduce que los adversarios expertos en tecnología puedan identificar cuál de sus contactos ha dado positivo.Los atacantes retroactivos pueden extraer poca información de los registros almacenado en lo móviles. Sólo pueden tener un recuento total de las balizas positivas de cada día. Un atacante proactivo y conocedor de las tecnologías puede modificar la aplicación para saber cuándo ha estado cerca de un usuario positivo.
- **Ubicaciones expuestas**. Para las ubicaciones expuestas, como los EphID no se pueden vincular a un solo dispositivo, es más difícil para los atacantes saber cuantos casos distintos visitaron la ubicación.

3.2.1.3. Privacidad del Diseño Híbrido

Este diseño tiene dos diferencias importantes con los otros dos diseños:

1) Controla la vinculabilidad de los EphID reportados por usuarios positivos y restringe la vinculabilidad a un periodo de ventana corto.
2) Permite a los usuarios positivos redactar EphID para ventanas de tiempo específicas.

Estas dos diferencias afectan las propiedades de privacidad del diseño híbrido en lo siguiente.

- **Estado del positivo en COVID-19**. Para el estado de un positivo, en comparación con el diseño desvinculable, El diseño híbrido proporciona una protección ligeramente menor contra atacantes proactivos y retroactivos. Como en este diseño el usuario puede decidir no compartir determinadas ventanas de tiempo, esto reduce más la probabilidad de éxito de ataques de identificación.
- **Ubicaciones expuestas**. Un atacante es menos probable que sepa el número de casos positivos que visitaron una ubicación específica debido al vinculación restringida.

3.2.2. Análisis de Seguridad

Las principales preocupaciones sobre la seguridad son [TPH+20]:

- **Eventos de exposición falsa**: Podría hacer creer a un usuario que está en riesgo.
- **Suprimir contactos de riesgo**: Que un usuario positivo o el backend eviten que otras personas se enteren de que están en riesgo.
- **Evitar el descubrimiento de contactos**: Un atacante podría perturbar el sistema y evitar el descubrimiento de contactos.

3.2.2.1. Seguridad del Diseño de Bajo Consumo

A continuación se analiza la seguridad del diseño de bajo consumo a partir de las preocupaciones vistas anteriormente.

- **Eventos de exposición falsa**. Un atacante con una antena potente puede desencadenar falsas alertas de exposición. Como resultado, otros dispositivos cercanos pueden interactuar con el dispositivo del atacante. Para completar el ataque, el atacante debe asegurarse que éstas interacciones se marquen como eventos de exposición. Para ello, el atacante puede:

  1) Él mismo da positivo y lleva su dispositivo al hospital.
  2) Sobornar a una persona positiva para que lleve el dispositivo del atacante al hospital.
  3) Secuestrar/sobornar a la autoridad sanitaria que autoriza a los positivos activar el rastreo.
  4) Comprometer el servidor backend.

  Como los EphID derivan de una semilla usando una función hash y una función pseudoaleatoria, es inviable que un atacante sepa la semilla del usuario al observar sus transmisiones.

- **Suprimir contactos de riesgo**. Los usuarios infectados pueden decidir no informar de su positivo o dejar de transmitir las
- **Evitar el descubrimiento de contactos**. Cualquier sistema con este diseño es susceptible a ataques de bloqueo, haciendo dejar de funcionar el sistema.

3.2.2.2. Seguridad del Diseño no Vinculable

A continuación se analiza la seguridad del diseño no vinculable a partir de las preocupaciones vistas anteriormente.

- **Suprimir contactos de riesgo y evitar el descubrimiento de contactos**. Tiene las mismas propiedades que el diseño de bajo consumo respecto a suprimir contactos de riesgo y evitar el descubrimiento de contactos.

- **Eventos de exposición falsa**. Un atacante puede causar falsas alarmas mediante ataques de extensión de rango BLE. Para crear eventos falsos, el atacante debe recibir y transmitir los EphID dentro del mismo tiempo. Como en el diseño de bajo consumo, la generación del EphID es desvinculable y evita que un atacante reclame como propio el EphID de otro usuario.

3.2.2.3. Seguridad del Diseño Híbrido

A continuación se analiza la seguridad del diseño híbrido a partir de las preocupaciones vistas anteriormente.

- **Suprimir contactos de riesgo y evitar el descubrimiento de contactos**. Tiene las mismas propiedades que el diseño de bajo consumo respecto a suprimir contactos de riesgo y evitar el descubrimiento de contactos.

- **Eventos de exposición falsa**. Un atacante puede causar falsas alarmas mediante ataques de extensión de rango BLE. Para crear eventos falsos, el atacante debe recibir y transmitir los EphID dentro de la misma ventana de tiempo. Como en el diseño de bajo consumo, la generación del EphID es desvinculable y evita que un atacante reclame como propio el EphID de otro usuario.

3.2.3. Privacidad en Modelos Centralizados vs Descentralizados

A continuación se comparan los resultados del análisis de la privacidad entre el modelo centralizado y el modelo descentralizado.

- **Gráfico social**. El servidor backend puede reconstruir el gráfico social de los usuarios a partir de la información compartida por los usuarios positivos. El servidor puede unir subgrafos de diferentes casos positivos para obtener una imagen completa del verdadero gráfico social. ROBERT y DESIRE proponen evitar la filtración del gráfico social mediante el uso de una red de comunicación anónima para cargar identificadores observados de una manera no vinculable.

- **Gráfico de interacción**. En un sistema en el que tanto los identificadores como la exposición al COVID-19 se calculan de forma centralizada y los identificadores observados cargados están vinculados, el servidor backend siempre puede asociar identificadores de transmisión efímeros cargados a pseudo-identificadores permanentes para dispositivos individuales. En PEPP-PT-NTK, OpenTrace y ROBERT, los identificadores observados tienen una marca de tiempo. Por lo tanto,

el servidor backend no solo puede reconstruir un gráfico social, sino que también puede reconstruir un gráfico de interacción.

- **Trazabilidad de la ubicación**. El diseño descentralizado limita el potencial de seguimiento de la ubicación a los usuarios que han recibido un diagnóstico positivo y durante el transcurso del período contagioso. En los sistemas centralizados en los que las claves se generan en el servidor, el acceso a las claves del lado del servidor permite vincular los EphID al identificador permanente de la aplicación correspondiente. Esto permite rastrear/identificar personas en función de los EphID observados en el pasado, así como rastrear los movimientos futuros de las personas.

- **Estado del positivo en COVID-19**. Los sistemas de rastreo de proximidad centralizados y descentralizados comparten la limitación de privacidad inherente de que pueden ser explotados por un usuario experto en tecnología para revelar qué personas de su lista de contactos podrían estar infectadas. Sin embargo, los diseños centralizados ocultan cuándo y con qué frecuencia el usuario estuvo en contacto con un paciente con COVID-19 positivo. Como resultado, los atacantes expertos en tecnología no pueden beneficiarse de la vinculación entre EphID y la información de tiempo para amplificar su ataque. En cambio, necesitan depender de varias cuentas.

3.2.4. Seguridad en Modelos Centralizados vs Modelo Descentralizados

A continuación se compara el análisis de la seguridad entre el modelo centralizado y el modelo descentralizado.

- **Eventos de exposición falsa**. Desencadenar falsas alarmas es fácil en todos los diseños centralizados excepto DESIRE. DESIRE requiere un intercambio activo entre usuarios para desencadenar un evento de exposición falso

- **Suprimir contactos en riesgo**. Es posible en cualquier sistema.

- **Evitar el descubrimiento de contactos**. Todos los sistemas son susceptibles a ataques de bloqueo.

Capítulo 4

Técnicas de Análisis de Vulnerabilidades

Para conseguir mitigar el impacto de los ciber ataques, varias empresas crearon una multitud de técnicas y estándares con las que conseguir distinta información sobre las vulnerabilidades en distintos sistemas. En este capítulo exploraremos que son y como se llevan acabo los ataques simulados en la Sección 4.1 y algunos de los estándares mas usados para las vulnerabilidades en aplicaciones móviles en la Sección 4.3.

4.1. Prueba de Penetración

La prueba de penetración, conocida como "pen test" es un ciber ataque simulado contra tu propio equipo para comprobar si existen vulnerabilidades que se pudieran explotar. Esto mitiga las debilidades potenciales del sistema antes de que ocurra un ataque real.

4.1.1. Fases

Las fases de esta prueba, como aparecen en la Figura 4.1, son las siguientes [IMP]:

1. **Planificación y reconocimiento**: Se define el alcance de al prueba y sus objetivos, además de ganar comprensión del sistema que se va a probar, para poder intuir posibles vulnerabilidades.

2. **Escaneado**: Entender como responderá la aplicación a distintos intentos de intrusión. Esto se hace mediante dos tipos de análisis:

   - **Análisis estático**: Consiste en analizar el código fuente para comprobar como se comporta la aplicación cuando esté en funcionamiento.
   - **Análisis dinámico**: También se analiza el código fuente, pero esta vez mientras está en ejecución, siendo esta una manera más práctica de escanear la aplicación, pues provee al "pen tester" de una visión a tiempo real de su funcionamiento.

3. **Acceso**: Se utilizan ataques del tipo inyecciones *Structured Query Languaje* (SQL), puertas traseras o *Cross-Site Scripting* (XSS) para ganar privilegios, interceptar tráfico, robar datos para entender el daño que se puede hacer a la aplicación.

4. **Persistencia del acceso**: La meta de la fase es comprobar si se puede utilizar alguna vulnerabilidad para mantener el acceso en el sistema el tiempo suficiente como para poder acceder a las zonas con los datos mas sensibles. La idea es imitar una amenaza *Advanced Persistent Threat* (APT).

5. **Análisis de resultados**: Los resultados del test se detallan en un informe que contiene las vulnerabilidades específicas explotadas, los datos sensibles a los que se ha tenido acceso y el tiempo que el "pen tester" ha estado en el sistema sin ser detectado.

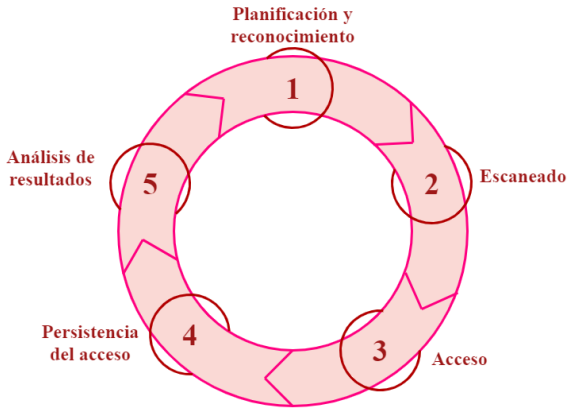


Figura 4.1: Fases del Análisis de Penetración

4.2. Tipos de Pruebas

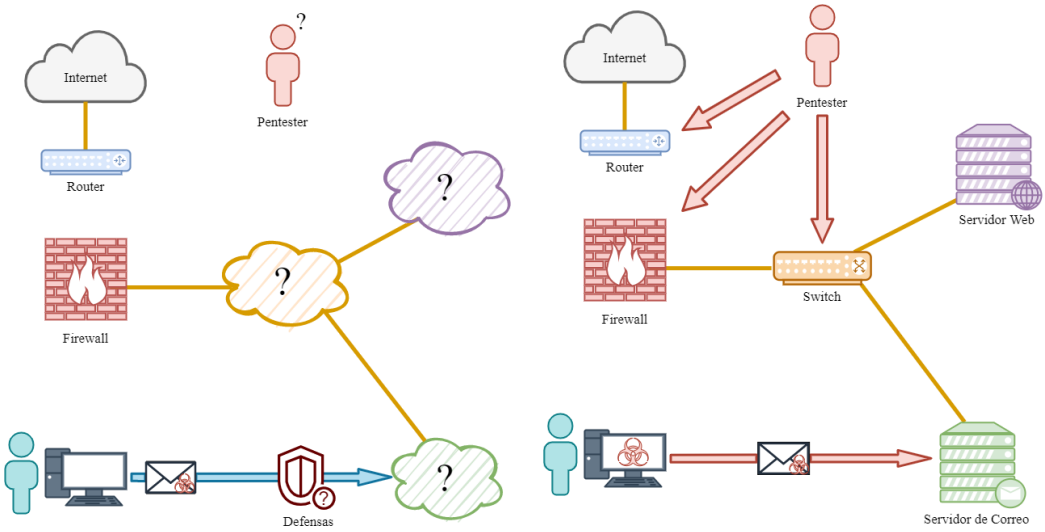
Las pruebas se clasifican dependiendo del objetivo y del escenario inicial [LR]:

- **Test externo**: Los objetivos son las partes del sistema que son visibles en internet, como pueden ser la aplicación web, el email o el *Domain Name Service* (DNS).

- **Test interno**: El objetivo es ganar acceso al sistema mas allá del firewall del sistema, simulando un atacante con acceso privilegiado, no tiene que simular necesariamente a un trabajador, pero si un atacante que haya robado las credenciales de un empleado mediante un ataque de phishing.

- **Test ciego**: En este tipo de test externo solo se le proporciona al "pen tester" el nombre de la empresa que tiene que atacar, lo que proporciona datos reales del tiempo que tardaría un atacante en perpetuar un asalto a la aplicación.

- **Doble test ciego**: El test es similar al test ciego, pero con la diferencia de que el personal de seguridad no es informado de que el ataque simulado va a tener lugar. Este tipo de test tiene algunos riesgos, como que se pongan varios sistemas en cuarentena con intención de parar el 'ataque'.
- **Test de caja negra**: Es un escenario similar al test ciego, en le que el tester tiene algo más de información sobre el sistema que tiene que atacar, como la *Uniform Resource Locator* (URL) de la web de la compañía, o su dirección IP, como se muestra en la Figura 4.2(a).
- **Test de caja blanca**: Se trata de un escenario en le que el "tester" tiene información detallada, como el código fuente, configuraciones o documentación del sistema para se encuentre la mayor cantidad de vulnerabilidades en el menor tiempo como se muestra en la Figura 4.2(b). La diferencia fundamental con el test interno es que no se dispone ningún tipo de credencial para ninguna cuenta de la empresa.
- **Test dirigido**: En este escenario, tanto el tester, como el equipo de seguridad trabajan juntos, manteniéndose informados de sus movimientos. Este ejercicio de prueba, da una visión al equipo de seguridad del punto de vista de un atacante potencial a tiempo real.



(a) Test de Caja Negra (b) Test de Caja Blanca

Figura 4.2: Diagrama de funcionamiento de los test de caja negra y de caja blanca

4.3. Estándares

Para ayudar a cuantificar la severidad y el impacto de las vulnerabilidades, algunas organizaciones han elaborado unos estándares de clasificación. Algunos de ellos son: El

Sistema de Puntuación de Vulnerabilidades Comunes, los Estándares de Verificación de Seguridad en Aplicaciones Móviles y la Enumeración de Debilidades Comunes.

4.3.1. Sistema de Puntuación de Vulnerabilidades Comunes

La métrica de evaluación por sistema de puntuación de vulnerabilidades comunes (del inglés, *Common Vulnerability Scoring System* (CVSS)) es un framework abierto creada por la organización *Forum of Incident Response and Security Teams* (FIRST) que es una asociación de entidades que proporcionan mecanismos de buenas prácticas, herramientas y elementos para la clasificación y respuesta a incidentes de seguridad [PM07]. FIRST ha estado trabajando en la renovación del sistema CVSS desde su lanzamiento en febrero de 2005 hasta la salida de su última versión en junio de 2019. El sistema de puntuación CVSS se compone de tres grupos principales de métricas (Base, Temporal y de Entrono) Que, a su vez, contienen otros conjuntos de métricas [PM07].

Las métricas usadas en la versión 2.0 se reúnen en tres grupos [PM07]:

1. **Métricas Base**: Contiene las cualidades intrínsecas de una vulnerabilidad, descritas en la Tabla 4.2.

2. **Métricas temporales**: Contiene las características de la vulnerabilidad que cambian con el tiempo, listadas en la Tabla 4.3.

3. **Métricas Entorno**: Contiene las características que se relacionan con el usuario, mostradas en la Tabla 4.4.

En las Tablas 4.1 se describen los grupos de métricas mencionadas anteriormente y en las Tablas 4.2, 4.3 y 4.4 los valores que pueden tomar dichas métricas. Estos se usan como se indica en la Figura 4.3 para calcular el valor final [PM07]. Como se observa en la figura, el CVSS se calcula dando un valor a cada métrica expuesta anteriormente. Además, la única métrica obligatoria es la Base y opcionalmente se pueden evaluar las métricas temporales y de entorno.

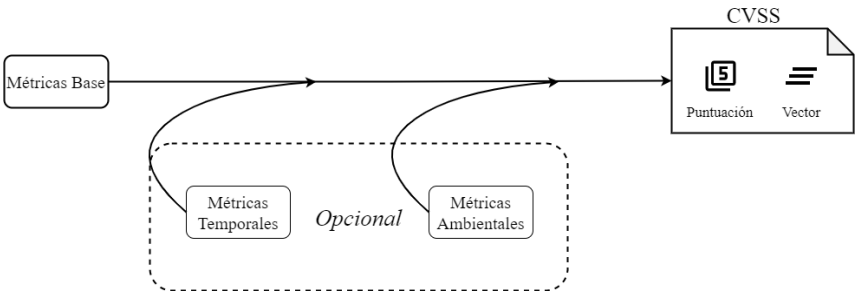


Figura 4.3: Uso de métricas para el cálculo del valor CVSS

Tabla 4.1: Descripción de los grupos de métricas de CVSS V2

Grupo	Métrica	Descripción	Valores
Métricas Base	Vector de acceso (AV)	Refleja como se explota la vulnerabilidad, cuanto mas remoto pueda estar el atacante, mayor será la puntuación	- Local - Red Adyacente - Red
	Complejidad de acceso (AC)	Mide la complejidad de ataque necesaria para explotar una vulnerabilidad cuando el atacante ya ha obtenido acceso al sistema. Cuanto menor sea la complejidad necesaria, mayor la puntuación	- Alto - Medio - Bajo
	Autentificación (Au)	Mide la cantidad de veces que un atacante necesita autentificarse para aprovechar la vulnerabilidad. Cuantas menos autentificaciones sean necesarias, mayor será la puntuación	- Múltiple - Único - Ninguno
	Impacto de confidencialidad (C)	Mide el impacto de confidencialidad de un ataque realizado con éxito. La confidencialidad se refiere a la divulgación de información solo a los usuarios autorizados	- Ninguno - Parcial - Completo
	Impacto de integridad (I)	Mide el impacto en la integridad del sistema después de una vulnerabilidad explotada con éxito. La integridad se refiere a la veracidad de la información	- Ninguno - Parcial - Completo
	Impacto de disponibilidad (A)	Mide el impacto en la disponibilidad del sistema después de una vulnerabilidad explotada con éxito. La disponibilidad se refiere a la accesibilidad de los recursos. Los ataques consumen ancho de banda en la red, ciclos de reloj o espacio en el disco	- Ninguno - Parcial - Completo
Métricas Temporales	Explotabilidad (E)	Mide el estado actual de las técnicas de explotación o la disponibilidad del código. Cuanto más disponible está el código de explotación, mas aumenta el numero de atacantes potenciales, incluso aquellos que no están calificados.	- No probado - Prueba de concepto - Funcional - Alto - No definido
	Nivel de remedio (RL)	Mide la urgencia con la que tratar la vulnerabilidad, cuanto menos oficial y permanente sea una solución, mayor será su puntuación.	- Arreglo oficial - Arreglo temporal - Solución alternativa - No disponible - No definido
	Informe de confianza (RC)	Mide le grado de confianza en la existencia de la vulnerabilidad y de la credibilidad de los detalles técnicos conocidos. Cuanto mas valida sea por el proveedor u otras fuentes acreditadas, mayor es su puntuación.	- Sin confirmar - Sin corroborar - Confirmado - No definido
Métricas de Entorno	Daño colateral potencial (CDP)	Mide el potencial de perdida de vidas o activos físicos por daño o robo. Esta métrica puede medir la pérdida económica de productividad o ingresos.	- Ninguno - Bajo - Bajo-Medio - Medio-Alto - Alto - No definido
	Distribución de objetivo (TD)	Mide la proporción de sistemas vulnerables, actúa como indicador del entorno para aproximar el porcentaje de sistemas que podrían verse afectados.	- Ninguno - Bajo - Medio - Alto - No definido
	Requisitos de seguridad	Esta métrica sirve a los analistas para marcar la importancia de los Requisitos de Confidencialidad (CR), Integridad (IR) y Disponibilidad (AR) aumentando o disminuyendo su peso en la puntuación final.	- Bajo - Medio - Alto - No definido

Tabla 4.2: Métricas del grupo Base

Métrica	Valor	Descripción
Vector de acceso (AV)	Local (L)	Solo se puede explotar con acceso físico al sistema o una cuenta local
	Red Adyacente (A)	Se requiere que el atacante tenga acceso al dominio de colisión (difusión) del software vulnerable
	Red (N)	El software vulnerable esta vinculado a la pila de red y el atacante no requiere acceso local o a la red local
Complejidad de acceso (AC)	Alto (H)	Existen condiciones de acceso específicas. Como puede ser que el atacante pueda tener privilegios elevados, la configuración vulnerable se ve raramente en la practica o que el atacante dependa de métodos de ingeniería social fácilmente detectables por personas con conocimientos
	Medio (M)	Las condiciones de acceso son algo específicas. Como que la parte atacante esta limitada a un grupo de sistemas o usuarios con algún nivel de autorización, se deba recopilar cierta información previa al ataque o que el ataque requiera ingeniería social que podría engañar a algunos usuarios
	Bajo (L)	No existen condiciones de acceso específicas. Como que el ataque se pueda realizar manualmente y sin necesidad de recopilar información adicional o que el producto afectado requiera acceso a una amplia gama de sistemas y usuarios, probablemente anónimos y poco confiables
Autenticación (Au)	Múltiple (M)	Se requiere que se el atacante se identifique dos o mas veces, incluso si son las mismas credenciales
	Único (S)	La vulnerabilidad requiere que se inicie sesión en el sistema
	Ninguno (N)	No se requiere ninguna autenticación para aprovechar la vulnerabilidad
Impacto de confidencialidad (C)	Ninguno (N)	No hay impacto en la confidencialidad del sistema
	Parcial (P)	Es posible accede a algunos archivos del sistema, pero el atacante no tiene control del alcance de la información que se obtiene
	Completo (C)	Existe una divulgación total, el atacante puede acceder a todos los datos del sistema
Impacto de integridad (I)	Ninguno (N)	No hay impacto en la integridad del sistema
	Parcial (P)	Es posible modificar algunos archivos del sistema, pero el atacante no tiene control sobre que archivos se puede sobrescribir o modificar, por lo que el alcance es limitado
	Completo (C)	Existe una pérdida total de protección del sistema, el atacante puede modificar cualquier archivo el sistema
Impacto de disponibilidad (A)	Ninguno (N)	No hay impacto en la disponibilidad del sistema
	Parcial (P)	El rendimiento esta reducido o existen interrupciones en la disponibilidad de los recursos
	Completo (C)	Hay un cierre total de los recursos afectados, el atacante puede hacer que los recursos no estén disponibles por completo

Tabla 4.3: Métricas del grupo Temporal

Métrica	Valor	Descripción
Explotabilidad (E)	No probado (U)	No existe código de la explotación, o es solamente teórico
	Prueba de concepto (POC)	Existe un código de explotación de una demostración del ataque, pero al no ser funcional en todos los entornos o situaciones, puede requerir una modificación sustancial por parte de un atacante experto
	Funcional (F)	El código de la explotación esta disponible y es funcional en la mayoría de las situaciones
	Alto (H)	El código funciona en todas las situaciones o se entrega de forma activa a través de un agente autónomo móvil (virus o gusano)
	No definido (ND)	Este valor sirve para omitir esta métrica cuando se calcule la puntuación
Nivel de remedio (RL)	Arreglo Oficial (OF)	El proveedor ha emitido un parche oficial con una solución completa
	Arreglo temporal (TF)	Hay una solución oficial de manera temporal
	Solución alternativa (W)	Hay una solución no oficial. En la mayoría de casos, la entidad afectada creará un parche propio para mitigar la vulnerabilidad
	No disponible (U)	No hay una solución disponible, o es imposible de aplicar
	No definido (ND)	Este valor sirve para omitir esta métrica cuando se calcule la puntuación
Informe de confianza (RC)	Sin confirmar (UC)	Existe una única fuente no confirmada, o varios informes contradictorios
	Sin corroborar (UR)	Hay varias fuentes no oficiales. Puede haber detalles técnicos contradictorios o hay alguna ambigüedad
	Confirmado (C)	La vulnerabilidad ha sido reconocida por el proveedor o autor de la tecnología afectada
	No definido (ND)	Este valor sirve para omitir esta métrica cuando se calcule la puntuación

Tabla 4.4: Métricas del grupo de Entorno

Métrica	Valor	Descripción
Daño colateral potencial (CDP)	Ninguno (N)	No hay posibilidad de perdida de vidas, activos físicos, productividad o ingreso
	Bajo (L)	Una explotación exitosa puede resultar en daños físicos o materiales leves
	Bajo-medio (LM)	Una explotación exitosa puede resultar en daños físicos o materiales moderados
	Medio-alto (MH)	Una explotación exitosa puede resultar en daños físicos o materiales significativos
	Alto (H)	Una explotación exitosa puede resultar en daños físicos o materiales catastróficos
	No definido (ND)	Este valor sirve para omitir esta métrica cuando se calcule la puntuación
Distribución de objetivo (TD)	Ninguno (N)	No existen sistemas objetivos, solo el 0 % está en riesgo
	Bajo (L)	Existen sistemas objetivos, pero a pequeña escala, solo entre el 1 % y el 25 % está en riesgo
	Medio (M)	Existen sistemas objetivos, pero a pequeña escala, solo entre el 26 % y el 75 % está en riesgo
	Alto (H)	Existen sistemas objetivos, pero a pequeña escala, solo entre el 76 % y el 100 % está en riesgo
	No definido (ND)	Este valor sirve para omitir esta métrica cuando se calcule la puntuación
Requisitos de seguridad	Bajo (L)	La pérdida de la métrica asociada tenga solo un efecto adverso limitado en la organización
	Medio (M)	La pérdida de la métrica asociada tenga solo un efecto adverso grave en la organización
	Alto (H)	La pérdida de la métrica asociada tenga solo un efecto adverso catastrófico en la organización
	No definido (ND)	Este valor sirve para omitir esta métrica cuando se calcule la puntuación

A cada grupo se le asignarán un valor numérico entre 0-10 mediante las Ecuaciones 4.1, 4.4 y 4.5.

Este calculo se hace siguiendo el árbol de decisión de la Figura 4.4 muestra la Figura 4.4 [PM07]. Los pasos de este árbol de decisión se pueden resumir en:

1. Si la puntuación Base no esta definida (Ecuación 4.1) Se devuelve un estado de error.
2. Si sí que esta definida, la puntuación total equivale a la Ecuación 4.1.

$$BaseScore = roundTo1Decimal(((0.6 \cdot Impact) + (0.4 \cdot Exp) - 1.5) \cdot f(Impact)) \quad (4.1)$$

   donde, el impacto y la explotabilidad se definen en las Ecuaciones 4.2 y 4.3 respectivamente.

$$Impact = 10.41 \cdot (1 - (1 - C) \cdot (1 - I) \cdot (1 - A)) \quad (4.2)$$

$$Exp = 20 \cdot AV \cdot AC \cdot Au \quad (4.3)$$

   donde, $f(impact) = 0$ si $Impact = 0, 1.176$ en otro caso
3. Si la puntuación Temporal esta definida entonces la puntuación total equivaldría a la Ecuación 4.4.

$$TemporalScore = roundTo1decimal(BaseScore \cdot E \cdot EL \cdot EC) \quad (4.4)$$

   Donde BaseScore se define en la Ecuación 4.1.
4. Si la puntuación de Entorno esta definida entonces la puntuación total es la que viene definida por la Ecuación 4.5.

$$EnvScore = roundTo1Decimal((AdjTemp + (10 - AdjTemp) \cdot CDP) \cdot TD) \quad (4.5)$$

$$AdjTemp = TemporalScore \quad (4.6)$$

   Recalculado con la Ecuación 4.2 de la puntuación Base reemplazado por la ecuación AdjImpact.

$$AdjImpact = min(10, 10.41 \cdot (1 - (1–C \cdot CR) \cdot (1–I \cdot IR) \cdot (1–A \cdot AR))) \quad (4.7)$$

5. Se devuelve la puntuación total calculada.

Los valores de las variables que se utilizan para calcular el CVSS se reflejan en las Tablas **??**, 4.6 y 4.7.

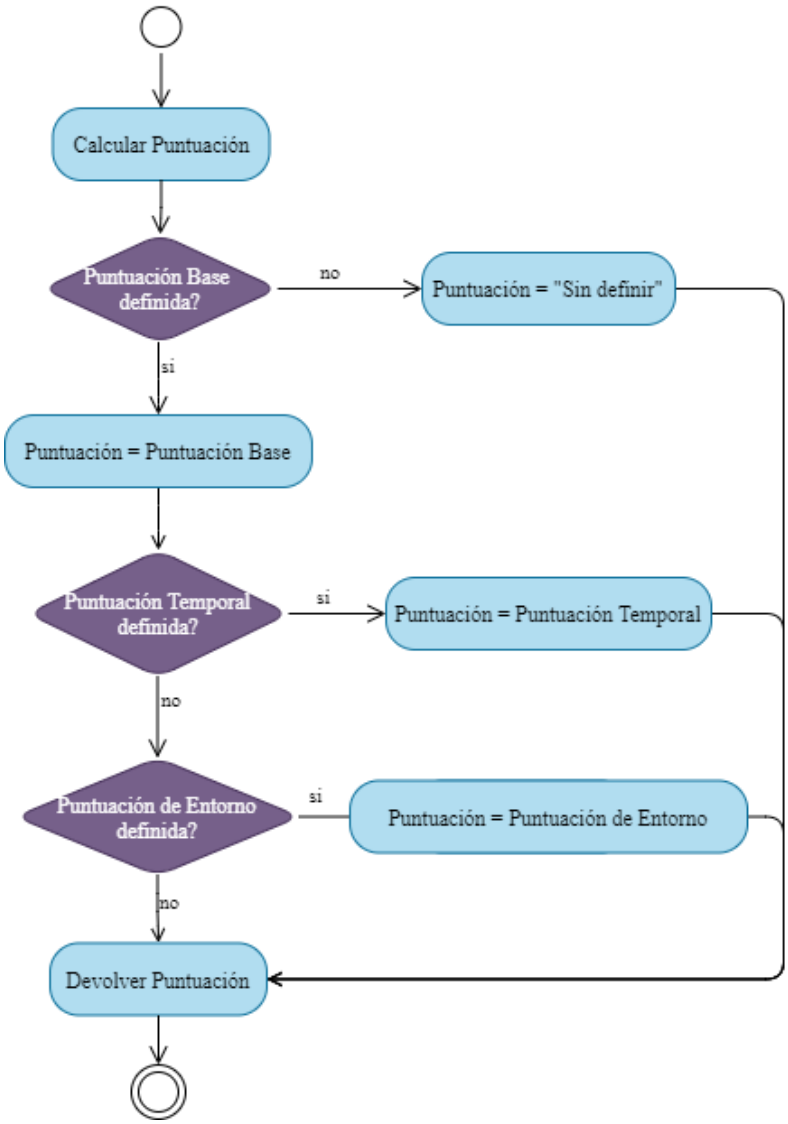


Figura 4.4: Árbol de decisión para calcular en valor CVSS Final

Tabla 4.5: Valor de las métricas de Entorno

Métricas	Sin definir	Ninguno	Bajo	Bajo-medio	Medio	Medio-alto	Alto
Daño colateral potencial (CDP)	0	0	0.1	0.3	NA	0.4	0.5
Distribución de objetivo (TD)	1	0	0.25	NA	0.75	NA	1
Requisito de confidencialidad (CR)	1	NA	0.5	NA	1	NA	1.51
Requisito de integridad (IR)	1	NA	0.5	NA	1	NA	1.51
Requisito de disponibilidad (AR)	1	NA	0.5	NA	1	NA	1.51

Tabla 4.6: Valor de las métricas Base

Métricas	Local	Red Adyacente	Red	Alto	Medio	Bajo	Múltiple	Único	Ninguno	Parcial	Completo
Vector de acceso (AV)	0.395	0.646	1	NA	NA	NA	NA	NA	NA	NA	NA
Complejidad de acceso (AC)	NA	NA	NA	0.35	0.61	0.71	NA	NA	NA	NA	NA
Autentificación (Au)	NA	NA	NA	NA	NA	NA	0.45	0.56	0.704	NA	NA
Impacto de confidencialidad (C)	NA	NA	NA	NA	NA	NA	NA	NA	0	0.275	0.66
Impacto de integridad (I)	NA	NA	NA	NA	NA	NA	NA	NA	0	0.275	0.66
Impacto de disponibilidad (A)	NA	NA	NA	NA	NA	NA	NA	NA	0	0.275	0.66

Tabla 4.7: Valor de las métricas de Entorno

Métricas	Sin definir	No probado	Prueba de concepto	Funcional	Alto	Arreglo oficial	Arreglo temporal	Solución altern.	No disponible	Sin confirmar	Sin corroborar	Confirmado
Explotabilidad (E)	1	0.85	0.9	0.95	1	NA	NA	NA	NA	NA	NA	NA
Nivel de remedio (RL)	1	NA	NA	NA	NA	0.87	0.9	0.95	1	NA	NA	NA
Informe de confianza (RC)	1	NA	NA	NA	NA	NA	NA	NA	NA	0.9	0.95	1

Al valor numérico final se le añade un vector con la evaluación de cada una de las métricas con la sintaxis (métricas: [valor])

Cuyo resultado podría ser: CVSS v2 Base Score: 5.0 (AV:N/ AC:L/ Au:N/ C:P/ I:N/ A:N)

Donde la puntuación asignada seria 5.0 y los valores del grupo Base serian:

- **AV:N** Vector de Acceso: Red
- **AC:L** Complejidad de acceso: Bajo
- **Au:N** Autentificación: Ninguno
- **C:P** Impacto de Confidencialidad: Parcial
- **I:N** Impacto de Integridad: Ninguna
- **A:N** Impacto de disponibilidad: Ninguno

Para acabar, la correspondencia entre la puntuación CVSS y un valor de severidad cualitativo es el siguiente [PM07]:

0 – Nula
0.1 - 3.9 – Baja
4.0 - 6.9 – Media
7.0 - 8.9 – Alta
9.0 - 10.0 – Crítica

4.3.2. Estándares de Verificación de Seguridad en Aplicaciones Móviles del Proyecto Abierto de Seguridad de Aplicaciones Web

El *Open Web Application Security Project* (OWASP) *Mobile Application Security Verification Standard* (MASVS) es un estándar para la seguridad de aplicaciones móviles que define dos niveles de verificación de seguridad (L1 y L2) y un conjunto de requisitos de resistencia contra la ingeniería inversa (R) [SS20].

Es importante saber que este estándar solo cubre la seguridad del lado del cliente en aplicaciones móviles y la comunicación de red entre la aplicación y sus puntos remotos, así como de algunos requerimientos básicos de autentificación de usuario y gestión de sesiones.

Entre las funciones que tiene el OWASP MASVS, las más comunes son [SS20] :

- Servir de guía para detallar una arquitectura segura.
- Servir como reemplazo de las checklists de desarrollo seguro en aplicaciones móviles.
- Servir como base para las metodologías de prueba de seguridad.
- Servir como guía para automatización de pruebas unitarias y de integración.

- Servir de entrenamiento en desarrollo seguro.

Los controles de seguridad son [SS20] :

1. **MASVS-L1**: Contiene los requerimientos genéricos de seguridad recomendado para toda aplicación móvil. Aplicarlos proporciona a la aplicación una seguridad frente a las vulnerabilidades más comunes.

2. **MASVS-L2**: Añade defensa adicional siempre que los controles de seguridad del SO estén intactos y que el usuario de la misma no sea considerado un atacante potencial. Este nivel se debería implementar en las aplicaciones que manejan datos sensibles, como la industria de la salud.

3. **MASVS-R**: Añade la seguridad frente a las amenazas provenientes del lado del cliente. Estos controles de protección de software nunca deberán sustituir a las otras comprobaciones pues esta pensado como un complemento para aquellas aplicaciones que ya cumplan con los requerimientos MASVS-L1 y MASVS-L2.

Con los 3 requerimientos de verificación que se muestran en la Figura 4.5, el resultado serían los siguientes 4 niveles [SS20] ordenados de menor a mayor seguridad:

1. **MASVS-L1**: Este nivel enumera las mejores prácticas de seguridad que se pueden añadir, con un impacto en el coste de desarrollo razonable además de mejorar significativamente la experiencia de usuario, por lo que se recomienda aplicarlo a todas las aplicaciones móviles que no sean aptas para niveles superiores.

2. **MASVS-L1 + R**: Se recomienda a todas las aplicaciones donde la protección de la propiedad intelectual sea un objetivo de la empresa. Los controles extra del MASVS-R proporcionan un aumento en dificultad a la hora de acceder al código fuente con, por ejemplo, ingeniería inversa y así, dificultar la manipulación y el cracking. Un ejemplo de aplicación con estas características es uno relacionado con la industria de los videojuegos, pues dificulta el modding y las trampas en el mismo.

3. **MASVS-L2**: Se aconseja este nivel de verificación para todas las aplicaciones que almacenen datos personales identificables que puedan ser usados para robo de identidad como las relacionadas con el sector de la salud, así como aquellas que tienen acceso a información altamente sensible como las relacionadas con el sector financiero que pueden dar lugar a pagos fraudulentos en caso de robo de datos.

4. **MASVS-L2 + R**: En este nivel se incluyen las aplicaciones ya mencionadas en el nivel MASVS-L2 con le añadido de las aplicaciones que contengan “compras en la aplicación” que idealmente deberían poder estar protegidas con el nivel anterior, pero en la realidad no siempre se puede proteger el lado de servidor con ese tipo de implementación y necesita los controles adicionales del MASVS-R para dificultar el esfuerzo en la manipulación de un potencial autor de malware.

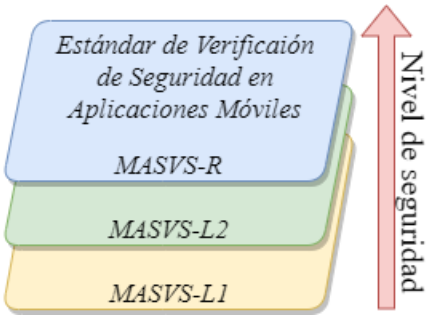


Figura 4.5: Controles de seguridad MASVS

4.3.3. Enumeración de Debilidades Comunes

El *Common Weakness Enumeration* (CWE) es un estándar abierto de debilidades encontradas en software y hardware que sirve como medición en algunas herramientas de software y como una línea base para la identificación de debilidades. El diccionario que lo contiene esta mantenida por la empresa americana MITRE Corporation [OWA17].

Para calcular la severidad de las vulnerabilidades, se utiliza un sistema de puntuación parecido al CVSS, llamado *Common Weakness Scoring System* (CWSS) que ayuda a organizar una lista con las debilidades de sistemas para mitigar su impacto.

Con estas puntuaciones se ha elaborado un ranking con el top 25 debilidades [OWA17]. Las 10 más importantes se presentan en la Tabla 4.8.

Tabla 4.8: Vulnerabilidades más importantes de CWE

Posición	Código	Nombre
1	**CWE-79**	Secuencia de comandos en sitios cruzados (XSS)
2	**CWE-787**	Escritura fuera de los límites
3	**CWE-20**	Validación incorrecta de datos de entrada
4	**CWE-125**	Lectura fuera de los límites
5	**CWE-119**	Restricción indebida de operaciones en los límites del búfer de memoria
6	**CWE-89**	Inyección SQL
7	**CWE-200**	Exposición de información sensible a un agente externo
8	**CWE-416**	Uso después de liberar memoria
9	**CWE-352**	Falsificación de peticiones en sitios cruzados
10	**CWE-78**	Inyección de comandos del Sistema operativo

Capítulo 5

Trabajos relacionados

En [Gvi20], se analiza la seguridad de la tecnología de rastreo de proximidad que desarrollaron Google Inc. y Apple Inc. inspirada en el protocolo DP-3T. Este trabajo se centra en el análisis de seguridad con las especificaciones de ésta tecnología, analizan los ataques más comunes, sus consecuencias y proponen nuevas estrategias para la mitigación de los mismos. Los ataques que analizan son:

- **Ataques de consumo de energía y almacenamiento**: El atacante ataca a un dispositivo cercano enviando un gran volumen de mensajes. Si el mensaje no es válido, el dispositivo lo descarta, pero gastando energía (ataque de consumo de energía). Si el mensaje es válido, la aplicación de rastreo almacena el id (ataque de consumo de almacenamiento), como se muestra en la Figura 5.1.

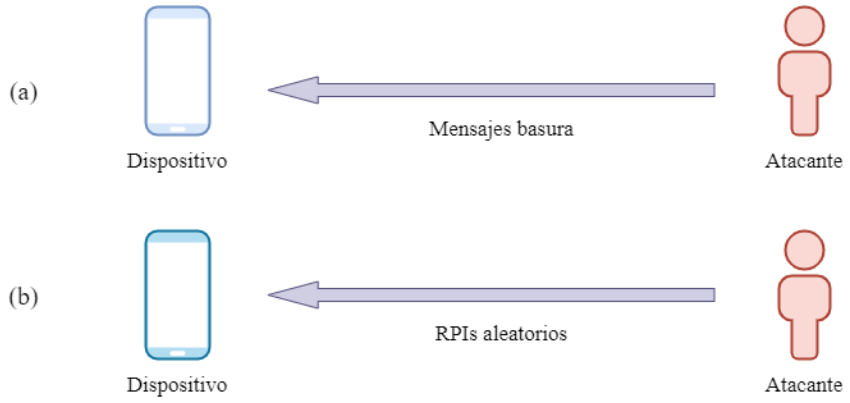


Figura 5.1: (a) Ataque de consumo de energía. (b) Ataque de consumo de almacenamiento.

- **Ataques de retransmisión y repetición**: En un ataque de retransmisión, el atacante utiliza dos dispositivos en dos ubicaciones distintas y transmite desde un dispositivo cualquier mensaje retransmitido desde el otro. En un ataque de repetición, el atacante usa uno o dos dispositivos en diferentes momentos y transmite en el momento posterior un mensaje grabado en el momento anterior, como se muestra en la Figura 5.2.

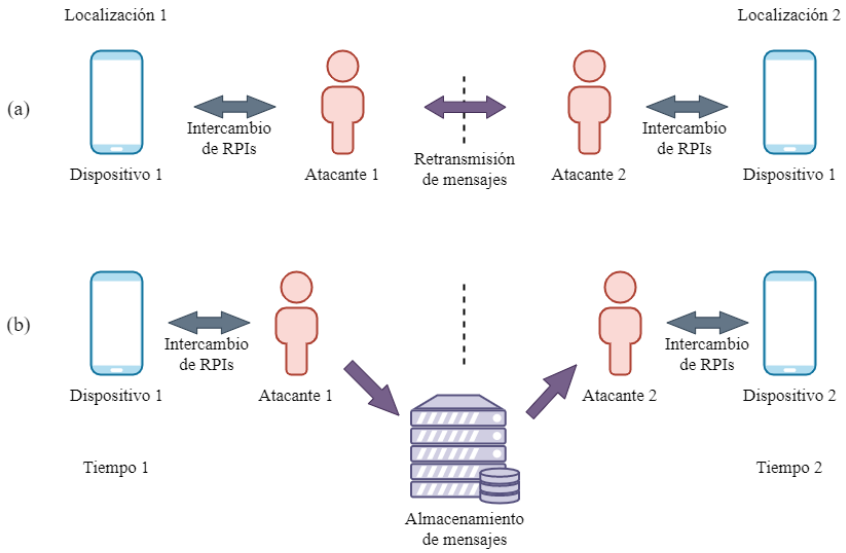


Figura 5.2: (a) Ataque de retransmisión. (b) Ataque de repetición.

- **Ataques de trolling**: El atacante es una persona que ha sido diagnosticada o espera ser diagnosticada positiva en COVID-19 pronto, y está interesada en hacer que las personas desprevenidas teman haber estado expuestas al virus. El atacante conecta su dispositivo móvil a un transporte (como un perro, un automóvil o un dron), que corre y anuncia id de proximidad a dispositivos de personas desprevenidas, como se muestra en la Figura 5.3.

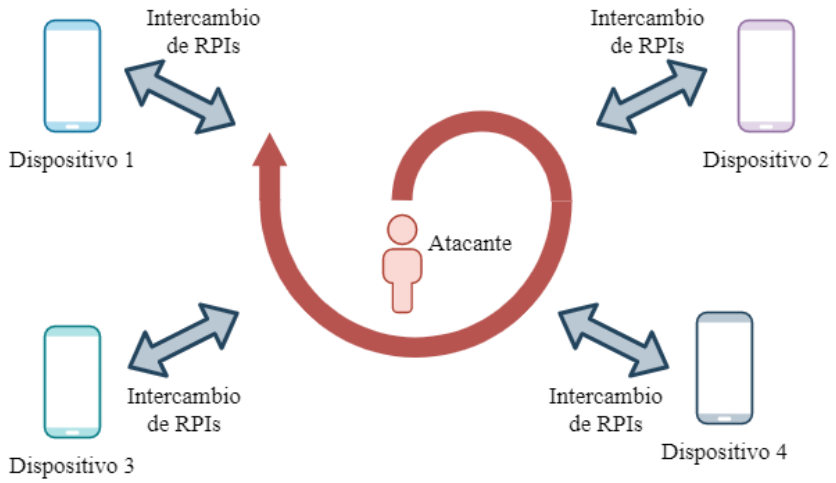


Figura 5.3: Ataque de trolling.

- **Ataques de enlace**: El atacante expone dos mensajes como provenientes del mismo dispositivo objetivo. El atacante permanece en una ubicación fija durante un tiempo suficiente para recibir mensajes de cualquier dispositivo cercano. En un ataque de enlace basado en el tiempo, el atacante infiere que es probable que dos mensajes estén

vinculados cuando su diferencia de tiempo es consistente con la frecuencia regular de mensajes.

- **Ataques de localización de señales**: : El atacante está interesado en exponer la identidad de una o varias personas en una reunión, cuyo dispositivo anuncia mensajes que incluyen ID de proximidad particulares. Con este fin, el atacante rastrea la intensidad de la señal de Bluetooth, junto con el campo de nivel de potencia de transmisión y la ubicación del dispositivo móvil del atacante cuando se recibió cada mensaje, y usa esta información para inferir ubicaciones sobre el tiempo de la persona objetivo con buena precisión. El atacante puede identificar a la persona objetivo observando físicamente quién está presente en las ubicaciones inferidas a lo largo del tiempo.

- **Ataques de singularización**: El atacante acerca un dispositivo a la persona objetivo, registra el ID de proximidad recibido junto con cuándo y dónde, y deja de grabar en ese dispositivo. Si la persona objetivo es diagnosticada positiva, las claves de diagnóstico de la persona objetivo permitirán al atacante hacer coincidir los ID registrados en el dispositivo y señalar a la persona objetivo.

Esta investigación demuestra que las especificaciones en el momento de la investigación pueden introducir riesgos importantes para la sociedad, como violaciones de privacidad y pérdida de confianza en el sistema. Las estrategias de mitigación que proponen para los ataques descritos no requieren cambios arquitectónicos en las especificaciones y son fáciles de adoptar.

En el artículo [MKHR+20], se detallan las principales arquitecturas usadas para el desarrollo de rastreo de contactos(centralizadas y descentralizadas), que tecnologías usan y cómo funcionan, como se muestra en la Tabla 5.1. El artículo también analiza diferentes tipos de posibles ataques y propone técnicas de mitigación.

Por último, el artículo habla sobre varias aplicaciones móviles de rastreo del COVID-19 europeas, nos informa sobre como se llaman, que tipo de arquitectura usan, que tecnología usan (BLE, *Global Positioning System* (GPS), etc.), como gestionan los datos y como funcionan éstas aplicaciones móviles. En el artículo [HWMF21] analizan la privacidad y seguridad de 28 aplicaciones Android de rastreo de contactos, que se muestran en la Tabla 5.2. Analizan los privilegios de su código, las políticas de privacidad y el desempeño estático y dinámico. También recopilan solicitudes de permisos, textos de políticas de privacidad, accesos a recursos en tiempo de ejecución y vulnerabilidades de seguridad existentes y con éstos datos evalúan el impacto sobre la privacidad de los usuarios. La investigación concluye que muchas de las aplicaciones se diseñaron de manera rápida, no tienen en cuenta la regulación de la privacidad y su documentación y políticas parecen incompletas.

Los principales hallazgos son:

- Las políticas de privacidad para muchas aplicaciones se encontraron incompletas o inexistentes.
- Muchas aplicaciones mostraron un comportamiento de acceso a los permisos que invadía la privacidad, especialmente los relacionados con la ubicación.
- Varias aplicaciones comenzaron a acceder a los permisos de ubicación antes de personalizarlos y registrarlos.
- Las aplicaciones de la UE mostraron, en general, menos riesgos de privacidad que las aplicaciones fuera de la UE. Aún así, la mayoría de las aplicaciones no cumplen con uno o más principios de privacidad.
- El análisis de vulnerabilidades de código mostró varias vulnerabilidades en los códigos de las aplicaciones.

Estos hallazgos indican un estado muy inmaduro de las aplicaciones de rastreo.

Tabla 5.1: Principales características de los frameworks de rastreo de contactos.

Framework	**Enfoque**	**Código fuente**	**Autoridad sanitaria**	**Ubicación recopilados**	**Autoreporte**
DP-3T	Descentralizado	Abierto	SI	NO	NO
Google/Apple	Descentralizado	Propiedad	SI	NO	NO
PEPP-PT-NTK	Centralizado	Abierto	SI	NO	NO
ROBERT	Centralizado	Abierto	SI	NO	NO
BlueTrace	Centralizado	Abierto	SI	NO	NO
TraceSecure	Centralizado	No disponible	SI	NO	NO
DESIRE	Hibrido	No disponible	SI	NO	NO
PACT(UW)	Descentralizado	Abierto	SI	NO	NO
PACT(MIT)	Descentralizado	No disponible	SI	Opcional	NO
TCN	Descentralizado	Abierto	Opcional	NO	SI
Open Covid Trace	Descentralizado	Abierto	SI	SI	NO
Whisper Tracing	Descentralizado	No disponible	NO	Opcional	SI

Tabla 5.2: Tabla de las 28 aplicaciones analizadas.

País	**App**	**Tecnología GPS**	**Tecnología BLE**	**Tecnología Sensors**
Australia	COVIDSafe		X	
Austria	Stopp Corona		X	
Brasil	Coronavirus-SUS	X		
Colombia	CoronApp	X		
Rep. Checa	eRouska		X	
Alemania	Ito		X	
Hungría	VirusRadar		X	
Georgia	Stop Covid	X	X	X
Islandia	Ranking C-19	X		
India	Mahakavach	X		
India	COVA Punjab	X		
India	Aarogya Setu	X	X	
Israel	Hamagen	X		
Italia	SM-Covid19	X	X	
Italia	diAry	X	X	X
Jordania	Aman	X		
Malasia	Gerak	X		
Malasia	MyTrace		X	
Macedonia del Norte	StopKorona!		X	
Noruega	Smittestopp	X	X	
Rusia	Contact Tracer	X	X	
Singapur	Trace Together		X	
Sudáfrica	Covi-ID	X		
UK	COVID Symptom Study	X		
US	COVID Safe		X	
US	PrivateKit	X		
US	NOVID		X	X
Global	Coalition		X	

Capítulo 6

Análisis de Vulnerabilidades en Aplicaciones Europeas de Rastreo de Covid-19 basadas en el Protocolo DP3T

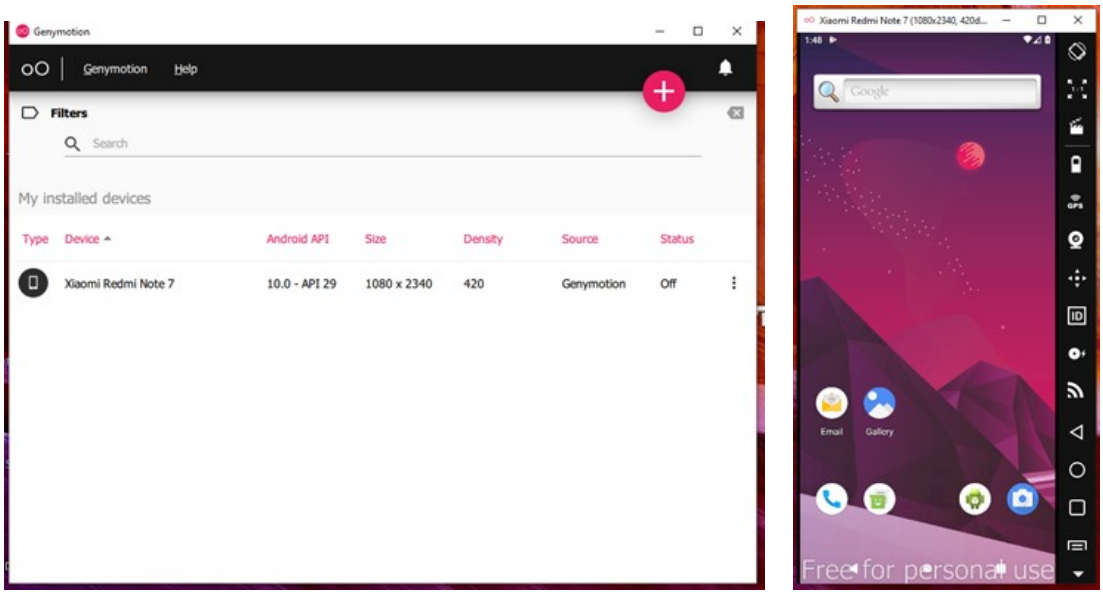
En este capítulo describimos detalladamente los resultados obtenidos de nuestro análisis sobre algunas de las aplicaciones móviles europeas de rastreo de proximidad para el COVID-19. Para realizar el análisis estático y dinámico de las aplicaciones se ha utilizado una herramienta automática como es *Mobile Security Framework* (MobSF) y un emulador de Android para simular un smartphone Android como es Genymotion. Se empezará explicando como se preparó el laboratorio de trabajo en la Sección 6.1, a continuación se detallará la información sobre las aplicaciones analizadas en la Sección 6.2 y se terminará con el análisis de las vulnerabilidades de dichas aplicaciones en la Sección 6.3.

6.1. Preparación del laboratorio de trabajo

Para el desarrollo del análisis de las vulnerabilidades en las aplicaciones de rastreo de proximidad de la COVID-19 se ha utilizado un ordenador portátil con conexión a internet en el que se ha instalado una herramienta automática de análisis de aplicaciones móviles y un emulador Android para realizar el análisis dinámico de las aplicaciones. Dado que las aplicaciones tienen protección contra las máquinas virtuales obligando al uso de bluetooth y políticas de rastreo de Google ha resultado más difícil emularlo. Por suerte con el entorno de pruebas de MobSF no se ha tenido mayor complicación para realizar el análisis tanto estático como dinámico.

Antes de decidir por utilizar el entorno de MobSF se barajaron más opciones, como una máquina virtual especializada para el análisis de vulnerabilidades en dispositivos móviles llamado Santoku. Se exploró como primera opción esta distribución de Linux por

su integración con el emulador de dispositivos móviles que se consideró mejor (Genymotion presentado en la Figura 6.1) además de por la gran cantidad de herramientas que tenía preinstaladas, como Wireshark, Mercury, Nmap, OWASP ZAP, Drozer o TCPDUMP entre muchas otras. Al final se acabó descartando este entorno por la imposibilidad de emular las aplicaciones y usar las herramientas eficientemente.



(a) Interfaz del emulador de Android Genymotion

(b) Emulación de smartphone Xiaomi Redmi Note 7

Figura 6.1: Emulador Genymotion

La siguiente distribución de Linux que se probó fue Androl4b, que cuenta también con una gran cantidad de herramientas para el análisis de penetración, lo más relevante es que tiene instalada la herramienta que tiene mejor fama, MobSF (Figura 6.2).

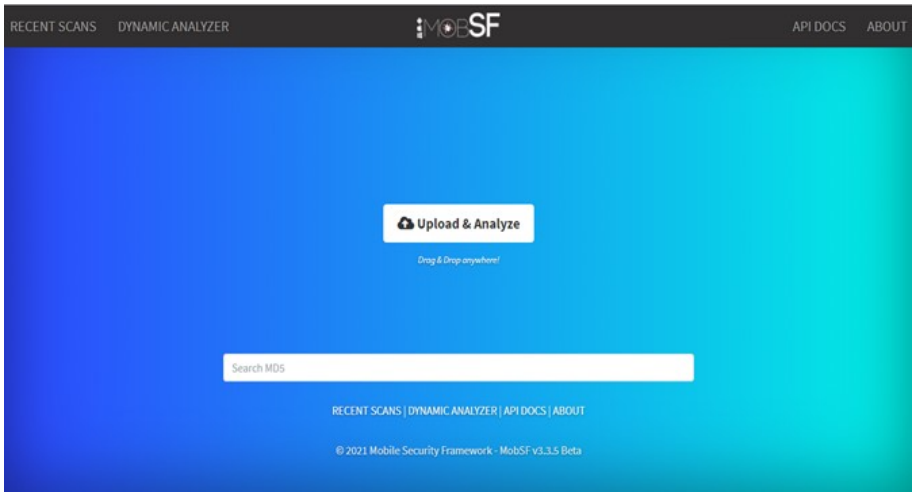


Figura 6.2: Interfaz de la herramienta MobSF.

Con este entorno se consiguió lanzar análisis estáticos sobre las *Android Application Package* (APK) de las aplicaciones, pero en la máquina virtual no se podían lanzar análisis dinámicos, por lo que también se acabó descartando este entorno. El entorno final se montó con MobSF instalado de forma nativa en Windows que, además de permitir realizar análisis dinámicos, daba información extra en los análisis estáticos como, por ejemplo, la localización de los servidores que se usan en las aplicaciones.

6.2. Aplicaciones

Para realizar los análisis se ha escogido 18 aplicaciones Europeas que usan la Tecnología desarrollada por Google y Apple, que se basa en el protocolo DP-3T. La Tabla 6.1 se muestran las aplicaciones analizadas.

Tabla 6.1: Aplicaciones analizadas

Pais	Aplicación	Versión
Alemania	Corona-Warn	1.13.2
Chipre	CovTracer	2.0.2
Dinamarca	Smite — stop	2.3.3
Eslovaquia	ZostanZdravy	1.0.1
Eslovenia	OstaniZdrav	1.10.1
España	Radar Covid	1.3.0
Estonia	Hoia	1.0.9
Finlandia	Koronavilkku	2.2.0
Francia	Stop Covid-19	2.2.0
Holanda	CoronaMelder	1.2.3
Hungría	VirusRadar	1.0.0
Italia	Immuni	2.2.1
Letonia	Apturi Covid	1.0.52
Macedonia	StopKorona!	1.1.0
Polonia	STOP COVID-ProteGO Safe	4.9.1
Portugal	STAYAWAY	1.1.3
Rep. Checa	eRouska	2.2.687
Suiza	SwissCovid	1.4.0

Es importante notar la versión que se ha utilizado para realizar los análisis. Los errores descritos más adelante pueden haber sido corregidos en versiones posteriores. Para obtener las APK de estas aplicaciones se consiguieron en la web APKPure pues no hay una manera oficial de conseguirlas.

6.3. Análisis de Resultados

Una vez conocidos el entorno de trabajo y las aplicaciones analizadas, se va a explorar cada vez más en profundidad los datos que ha proporcionado el analizador MobSF con una visión general en la Sección 6.3.1, un análisis de los permisos en la Sección 6.3.2 y un

análisis de los errores en el código en la Sección 6.3.3.

6.3.1. Análisis General

Un 38 % de las aplicaciones mostradas en la Tabla 6.1 no alcanzan un aprobado según el analizador MobSF y de las que sí superan los 50 de puntuación, solo una supera los 80 puntos. Sin embargo en las puntuaciones CVSS promedias todas son bastante parejas estando un 88 % en una severidad media y el 12 % restante con severidad alta.

Llama la atención la diferencia entre la puntuación de MobSF y la del estándar CVSS y esto se debe a la diferencia de criterios para puntuar, el funcionamiento de CVSS se explicó en la Sección 4.3.1 y la puntuación estática de MobSF funciona de la siguiente manera:

1.- Cada aplicación empieza con una puntuación ideal de 100 puntos

2.- Se ajusta la puntuación inicial en función de la severidad de los hallazgos de la siguiente manera:

- Si la severidad es alta se reduce en 15 la puntuación.
- Si la severidad es media (aviso) se reduce en 10 la puntuación.
- Si la severidad es buena se añade 5 a la puntuación.

3.- Se redondea la puntuación resultante de manera que si la puntuación final es mayor que 100, se considera como 100, y si la puntuación es menor que 0 se considera la puntuación final como 10.

Por lo que la puntuación se define en función del número de hallazgos que encuentra durante el análisis. Lo que, a primera vista pudiera parecer que las aplicaciones tienen una trato desigual porque tienen distinto número de hallazgos, lo que provoca en realidad es que se perjudique a aquellos cuyo balance entre buenas y malas practicas esté decantado por aquellas que tengan una severidad alta y termine beneficiando a aquellas aplicaciones que cuenten por ejemplo con: una buena encriptación o utilicen unas buenas *Application Programming Interface* (API)s de seguridad, que es lo que se busca en última instancia.

6.3.2. Análisis de Permisos

Antes de hablar sobre los resultados que ha arrojado MobSF se van a reducir la cantidad de permisos que se consideran relevantes. La gran mayoría de las aplicaciones tienen permisos relacionados con el acceso a internet (*android.permission.INTERNET*), ver el estado de las redes (*android.permission.ACCESS_NETWORK_STATE*), crear conexiones Bluetooth (*android.permission.BLUETOOTH*) Y para evitar que se apague la pantalla del dispositivo mientras se utiliza la aplicación (*android.permission.WAKE_LOCK*) Que

no son consideradas potencialmente peligrosos por el analizador MobSF que las marca con un estado normal. En lo referente a los permisos considerados peligrosos mostrados en la Tabla 6.3 se pueden encontrar los siguientes:

- **ACTIVITY_RECOGNITION**: Permite a la aplicación conocer datos sobre la actividad física del usuario a partir de los sensores del dispositivo móvil, es decir, si el usuario del dispositivo se encuentra corriendo, caminando, montando en bicicleta, moviéndose en algún vehículo o incluso si esta parado en algún lugar [Goo15].
- **ACCESS_BACKGROUND_LOCATION**: Permite a la aplicación acceder a la ubicación del dispositivo móvil cuando esta se encuentra en segundo plano.
- **ACCESS_COARSE_LOCATION**: Se accede a las fuentes de ubicación aproximada como la red de datos móviles o la red Wi-Fi para determinar la ubicación del dispositivo móvil. El nivel de exactitud equivale a una manzana [Goo21].
- **ACCESS_FINE_LOCATION**: Se accede a las fuentes de ubicación precisa, como el GPS del dispositivo móvil cuando sea posible además de usar también la red de datos móviles o la red Wi-Fi, el uso de este tipo de ubiación aumenta el consumo de la batería del dispositivo [Goo21].
- **AUTHENTICATE_ACCOUNTS**: Permite a la aplicación utilizar los autentificadores de cuentas del Account Manager, inclutyendo la creación de cuentas, además de obtener y establecer su contraseña.
- **CAMERA**: Permite a la aplicación capturar imágenes y videos con la cámara del dispositivo móvil en cualquier momento.
- **READ_EXTERNAL_STORAGE**: Permite a la aplicación leer datos de una almacenamiento externo, lo que puede dar lugar a una infección de malware por esta vía.
- **WRITE_EXTERNAL_STORAGE**: Permite a la aplicación escribir datos en un almacenamiento externo.

Según la API de Google sobre el acceso a la ubicación del dispositivo [Goo21] se debe decidir el nivel de precisión que necesita la aplicación, siendo más precisa la *FINE_LOCATION* que la *COARSE_LOCATION* debido al uso del GPS como instrumento adicional para determinar la posición del teléfono. Cuando se haya decidido la precisión necesaria, se debe solicitar solo **uno** de los permisos mencionados.

En la Tabla 6.3 se puede observar que en aquellas aplicaciones mostradas en la Tabla 6.2 que hacen uso de los permisos de ubicación (un 22 % de ellas) Ninguna utiliza únicamente **una** de ellas.

Tabla 6.2: Puntuaciones de MobSF

Aplicación	ID	Puntuación estática	CVSS Promedio
Corona-Warn	CW	10	6.4
CovTracer	CT	15	6.7
Smite — stop	SS	65	5.8
ZostanZdravy	ZZ	35	6.8
OstaniZdrav	OZ	5	6.4
Radar Covid	RC	55	6.5
Hoia	H	50	6.5
Koronavilkku	KV	45	6.8
Stop Covid-19	SpC	75	6.8
CoronaMelder	CM	25	6.2
VirusRadar	VR	50	6.4
Immuni	I	40	6.2
Apturi Covid	AC	65	7
StopKorona!	SK	50	6.4
STOP COVID-ProteGO Safe	PS	85	7.9
STAYAWAY	SA	65	6.3
eRouska	eR	75	6.6
SwissCovid	SwC	50	6.5

Siendo un número tan bajo las aplicaciones que usan estos permisos, da que pensar hasta que punto se necesita para el funcionamiento utilizar el permiso de la ubicación aproximada, pues solo se necesita el acceso a la ubicación precisa para hacer uso de BLE en la aplicación. La API de BLE describe que si el dispositivo tiene Android 9 o inferior **se puede usar** el permiso a la ubicación aproximada en lugar de la precisa [Dev19]. Lo que no se contempla es que se usen las dos a la vez, si no sabes la versión de Android del dispositivo que aloja la aplicación siempre es mejor pedir la ubicación precisa, pues no se debería poder tener las dos simultáneamente. *CovTracer*, por otra parte, pide acceso a la ubicación en segundo plano, cosa que no hacen ninguna de las otras aplicaciones que piden acceso a la ubicación.

Estos datos parecen indicar que realmente no ha habido un estudio de requisitos previo a la implementación de dichas aplicaciones, provocado, probablemente por el escaso tiempo de reacción que hayan tenido los desarrolladores al inicio de la pandemia, lo que ha acabado causando que algunas aplicaciones tengan un numero muy amplio de permisos que la gran mayoría no necesitan como se puede ver en la Figura 6.3. Como se observa en la tabla, las tablas en rojo muestran los permisos peligrosos y en gris los desconocidos.

Tabla 6.3: Permisos de las aplicaciones de la Tabla 6.1

Permiso	CW	CT	SS	ZZ	OZ	RC	H	KV	SpC	CM	VR	I	AC	SK	PS	SA	eR	Swc
ACTIVITY_RECOGNITION																		
ACCESS_BACKGROUND_LOCATION																		
ACCESS_COARSE_LOCATION																		
ACCESS_FINE_LOCATION																		
AUTHENTICATE_ACCOUNTS																		
CAMERA																		
READ_EXTERNAL_STORAGE																		
WRITE_EXTERNAL_STORAGE																		
READ_APP_BADGE																		
UPDATE_COUNT																		
CHECK_LICENSE																		
BIND_GET_INSTALL_REFER_SERVICE																		
hardware.location																		
htc.launcher.permission.READ_SETTINGS																		
UPDATE_SHORTCUT																		
CHANGE_BADGE																		
READ_SETTING																		
WRITE_SETTINGS																		
USE_COMPONENT																		
UPDATE_BADGE																		
oppo.launcher.permission.READ_SETTINGS																		
oppo.launcher.permission.WRITE_SETTINGS																		
badge.permision.READ																		
badge.permision.WRITE																		
BROADCAST_BADGE																		
PROVIDER_INSERT_BADGE																		
C2D_MESSAGE																		
BADGE_COUNT_READ																		
BADGE_COUNT_WRITE																		
OPPO_COMPONENT_SAFE																		

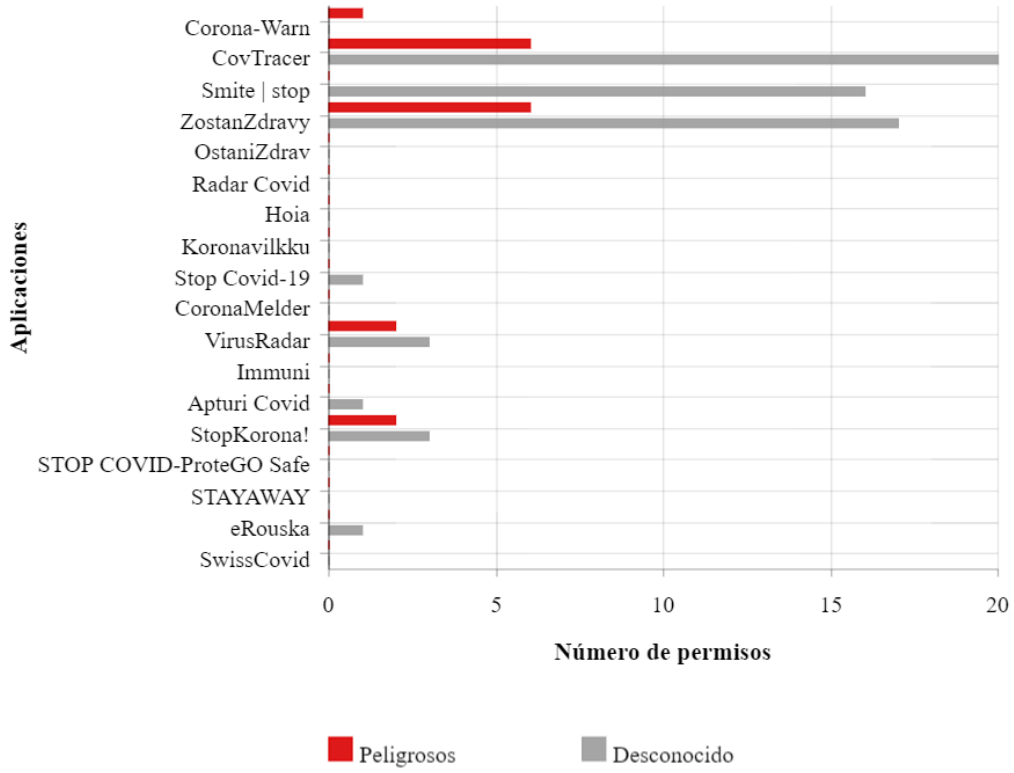


Figura 6.3: Tipos de permisos por aplicación

Pero este dato no solo es aplicable a los permisos relacionados con la ubicación, se extiende a otros como reconocimiento de la actividad física o el uso de la cámara. Donde este último sí que es necesario para la funcionalidad de la aplicación en concreto, pero cuando solo un 5.5 % de las aplicaciones lo usa, denota que su uso, aunque justificado puede ser innecesario si la implementación se piensa para aumentar la privacidad de los usuarios.

Con respecto a los últimos permisos marcados como peligrosos, *AUTHENTICATE_ACCOUNTS*, *READ_EXTERNAL_STORAGE* y *WRITE_EXTERNAL_STORAGE* no son peligrosos por sí mismo, pero pueden dar lugar a que sea más probable un ataque a aplicaciones con estas características. Con acceso a almacenamiento externo, un usuario malicioso podría infectar con más facilidad la aplicación con algún tipo de malware, así mismo, si ese usuario se hiciese con el control del dispositivo móvil, podría suplantar fácilmente la identidad del usuario del teléfono pues tendría acceso a la cuenta mediante el *AccountManager*.

MobSF a parte de las categorías de severidad normal y peligrosa que se han analizado previamente, tiene otra categoría donde coloca todos aquellos permisos que no sabe como catalogar, la gran mayoría son inocuos pero se pueden dividir en estos grupos:

- **Relacionados con el icono**: Estos permisos son los relacionados con el *Badge* o en español la insignia de notificaciones de la aplicación. En este grupo se incluyen todos

los permisos que permiten la manipulación del icono, como mostrar una insignia de notificación en el, leer que número tiene o actualizar el número o icono que aparece.

- **Ubicación**: En este grupo se incluye el permiso de *android.hardware.location*, que permite a la aplicación acceder a una o más funciones del dispositivo como el GPS o la red para determinar la ubicación del teléfono [Dev20a]. Tener un permiso para la ubicación (precisa o aproximada) Implica que si la aplicación esta destinada a un dispositivo con Android 5.0 o una versión posterior necesita pedir explícitamente este permiso [Dev20b].
- **Mensajería entre la nube y el dispositivo**: Los mensajes *Cloud to Device Messaging* (C2D) en español dispositivo-nube son un tipo de mensajería obsoleta desde mediados de 2012 y quitada del soporte 3 años mas tarde [Dev15]. Este tipo de mensajería pretendía dar soporte a los servicios de mensajería entre el backend y la aplicación brindando una conexión a la nube.

Como se puede observar estos permisos no son más peligrosos que que alguna aplicación esté usando permisos obsoletos.

6.3.3. Análisis de Código

A continuación se analizará el impacto de los problemas de código enumerados en la Tabla 6.4 encontrados en las aplicaciones de la Tabla 6.1. En ella se muestran todos los problemas agrupados por severidad junto con las clasificaciones que tienen en los distintos estándares que utiliza MobSF explicados en la Sección 4.3.

Como se puede ver en la Figura 6.4 la cantidad de problemas detectados es alta, estando la media en 5.5 problemas por aplicación con un mínimo de 3 en STOP COVID-ProteGO Safe y un máximo de 9 en Corona-Warn.

La gran mayoría de los problemas no suponen un peligro directo en el código de la aplicación, pero estos podrían ser de ayuda para un posible atacante a la hora de obtener información o tomar el control de la aplicación. Muchos de estos problemas vienen a raíz del uso de un mal uso del almacenamiento de información sensible (2 y 12) o el uso de algoritmos criptográficos con problemas como son SHA-1 (4) y MD5 (8), que ambos presentan colisiones hash. Estas se producen cuando dos entradas distintas devuelven la misma salida a través de la función de hash. Ambos algoritmos presentan debilidades probadas en este aspecto, por lo que siempre se recomienda el uso de algoritmos alternativos como SHA-256.

Tabla 6.4: Problemas de código detectados en las aplicaciones por MobSF

Problema	**Severidad**	**CVSS V2**	**CWE**	**OWASP Top 10**	**OWASP MASVS**
1. Utiliza un generador de números aleatorios inseguro	alto	7.5	CWE-330	M5: Criptogafía insuficiente	MSTG-CRYPTO-6
2. Los ficheros pueden contener información codificada	alto	7.4	CWE-312	M9: Ingeniería inversa	MSTG-STORAGE-14
3. Implementación insegura de SSL	alto	7.4	CWE-295	M3: Comunicación insegura	MSTG-NETWORK-3
4. SHA-1 es una hash con colisiones hash	alto	5.9	CWE-327	M5: Criptogafía insuficiente	MSTG-CRYPTO-4
5. Utiliza una BD SQLite y ejecuta consultas SQL sin formato	alto	5.9	CWE-89	M7: Calidad del código del cliente	NA
6. Crea archivos temporales	alto	5.5	CWE-276	M2: Almacenamiento	MSTG-STORAGE-2
7. Puede leer/escribir desde almacenamiento externo	alto	5.5	CWE-276	M2: Almacenamiento de datos inseguro	MSTG-STORAGE-2
8. MD5 es un hash con colisiones hash	alto	4.4	CWE-327	M5: Criptogafía insuficiente	MSTG-CRYPTO-4
9. Implementación insegura de WebView	aviso	8.8	CWE-749	M1: Uso inadecuado de la plataforma	MSTG-PLATAFORM-7
10. Divulgación de direcciones IP	aviso	4.3	CWE-200	NA	MSTG-CODE-2
11. Escucha cambios en el portapapeles	aviso	0	NA	NA	MSTG-STORAGE-2
12. Se escribe información sensible en los registros de la aplicación	información	7.5	CWE-532	NA	MSTG-STORAGE-3
13. Puede escribir en el directorio de la aplicación	información	3.9	CWE-276	NA	MSTG-STORAGE-14
14. Copia datos al portapapeles	información	0	NA	NA	MSTG-STORAGE-10
15. Utiliza cifrado SQL	información	0	NA	NA	MSTG-CRYPTO-1

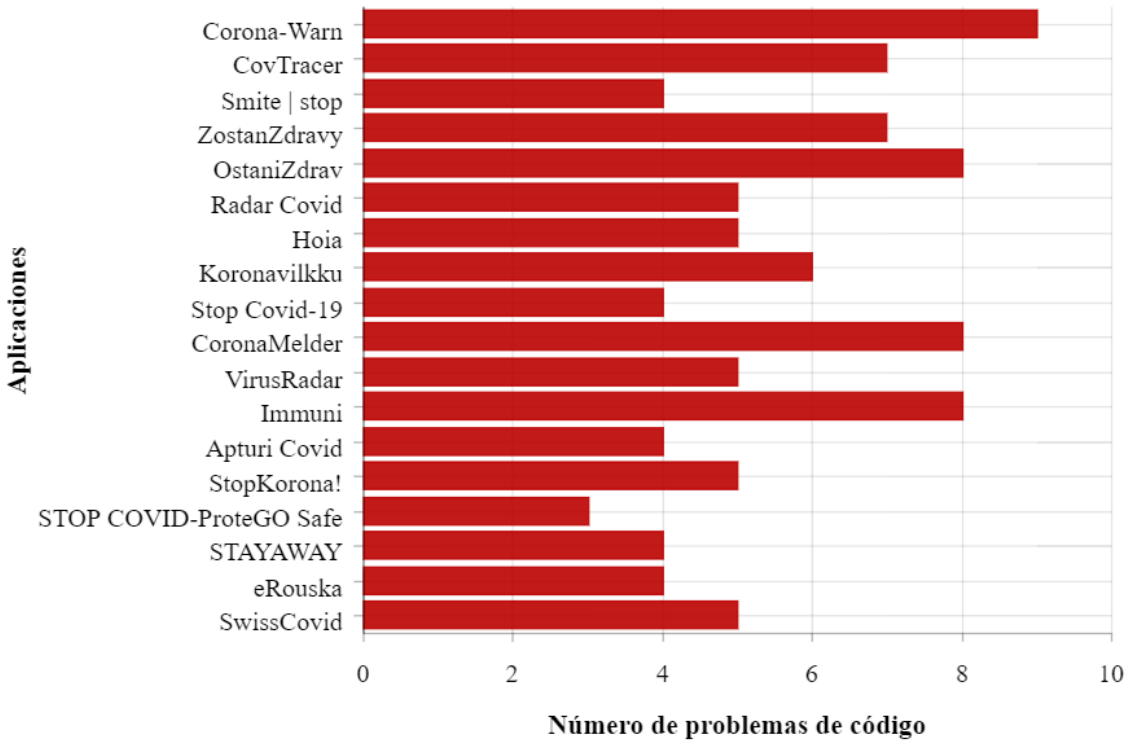


Figura 6.4: Número de problemas de código por aplicación

Los errores más graves que tienen algunas aplicaciones son referentes al tratamiento de información sensible. Como se indica en la Tabla 6.5 el 100 % de las aplicaciones escriben este tipo de información en los registros de la aplicación (logs) Lo que vulnera gravemente la privacidad del usuario de la misma y, además, un 72 % de las aplicaciones contienen ficheros con información codificada. Un 61 % de las aplicaciones tienen problemas con la divulgación de direcciones IP, lo que conllevaría una exposición de información que no debería llegar a personas no autorizadas [PLO21].

Aunque el problema más preocupante se encuentra en el generador de números aleatorios inseguro (1), es el más grave porque toda la infraestructura de generación de EphID se basa en números aleatorios, si su generación es insegura y se pueden predecir los EphID quiere decir que alguien con ese conocimiento podría suplantar la identidad de otro usuario con el fin de atacar el sistema.

Tabla 6.5: Aplicaciones con los problemas de código de la Tabla 6.4

Aplicación	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Corona-Warn	■		■	■	■	■		■		■		■			■
CovTracer	■	■			■	■	■	■				■			
Smite — stop							■			■	■	■			
ZostanZdravy		■			■	■	■		■			■		■	
OstaniZdrav	■		■	■		■		■		■		■			■
Radar Covid	■	■			■					■		■			
Hoia	■	■			■					■		■			
Koronavilkku	■	■			■	■						■			■
Stop Covid-19	■					■						■		■	
CoronaMelder	■	■			■	■	■			■		■		■	
VirusRadar	■	■				■				■		■			
Immuni	■	■	■			■				■		■	■	■	
Apturi Covid	■	■				■						■			
StopKorona!	■	■				■				■		■			
STOP COVID-ProteGO Safe	■								■			■			
STAYAWAY		■			■					■		■			
eRouska	■	■										■	■		
SwissCovid	■	■			■					■		■			

Capítulo 7

Contribuciones Individuales

7.1. Pablo Izquierdo Garbajosa

El primer paso al comenzar este trabajo fue recopilar toda la información posible de trabajos y artículos relacionados principalmente con el protocolo DP-3T, pero también información sobre los diferentes tipos de modelos y protocolos que existen acerca del rastreo de contactos. Para la recopilación de la información se utilizó el buscador de *Google Scholar*. En este paso aprendí a hacer uso del buscador *Google Scholar* para buscar los artículos relacionados con el desarrollo del trabajo. Al ser éste protocolo bastante reciente, la recopilación de la información relevante para el trabajo ha sido un poco complicada, con lo que también se ha aprendido a filtrar la información más relevante para nuestro trabajo. Con el paso de los meses se fueron publicando también artículos relacionados con el análisis de vulnerabilidades de las aplicaciones de rastreo del COVID-19 que usan diferentes países del mundo que nos ayudaron mucho en nuestro proceso de recopilación de información. También destacar que prácticamente todos los artículos relacionados con nuestro trabajo están escritos en inglés, esto me ha ayudado a mejorar mi lectura y mi vocabulario de dicho idioma.

El segundo paso fue buscar las aplicaciones móviles de rastreo de contactos y la preparación del laboratorio de trabajo para realizar el análisis de dichas aplicaciones. De la información recopilada anteriormente se eligieron las aplicaciones Europeas que se basaban en el protocolo DP-3T y descargamos sus APK de la web *APKPure*.

Para la creación del laboratorio buscamos información sobre las herramientas más utilizadas a la hora de realizar test de penetración en aplicaciones móviles Android, con lo que aprendí a montar mi propio laboratorio de pruebas para analizar vulnerabilidades en aplicaciones móviles Android. Instalamos y probamos varios entornos virtuales creados para el tipo de análisis que necesitábamos hacer pero ninguno permitía realizar tanto análisis estático como dinámico, hasta que encontramos la herramienta MobSF que se podía instalar de forma nativa en *Windows* y permitía hacer el análisis estático y permitía hacer el análisis dinámico a través de *Genymotion*, que fue la herramienta elegida para

emular un dispositivo Android.

En éste paso, a parte de aprender el uso de la herramienta MobSF, durante la búsqueda de las herramientas que nos sirvieran para el análisis de aplicaciones aprendí el uso de otras herramientas de análisis de vulnerabilidades, como *Drozer*, que es un *framework* para testear vulnerabilidades de aplicaciones haciendo uso también de la herramienta *Genymotion*, pero no nos proporcionaba la información que buscábamos. También aprendí acerca de entornos virtuales como *Androl4b*, que es una distribución de Linux que reúne herramientas de análisis de vulnerabilidades y de hacking ético, una de esas herramientas es MobSF, que es la herramienta que elegimos finalmente para nuestros análisis, pero ésta distribución no permitía hacer el análisis dinámico de la aplicación, por que la descartamos. Otra distribución que aprendí fue *Santoku*, que incluye herramientas como *Wireshark* y la ya mencionada *Drozer*, pero también la descartamos ya que no nos proporcionaba las herramientas de análisis que necesitabamos.

El siguiente paso fue analizar las aplicaciones seleccionadas a través del laboratorio creado. Pasamos todas las aplicaciones por el analizador y recabamos toda la información que nos daba la herramienta MobSF. La herramienta nos dio información de todo tipo como el país y la versión de Android de la aplicación, localización de los servidores que utilizan, las API de Google y los permisos que solicitan cada aplicación para funcionar. Para el apartado de los permisos, la herramienta hace uso de estándares como CWE y OWASP MASVS, con los que hace una puntuación de severidad sobre cada permiso, para entender el significado de dichas puntuaciones tuvimos que consultar sus respectivas páginas web y estudiar su significado. En este proceso, descartamos una gran cantidad de datos que la herramienta no identificaba como peligrosos y nos centramos en aquellos datos en los que la herramienta daba una puntuación de peligro potencial. Toda esta información la recopilamos en tablas que luego fueron plasmadas en la memoria.

En este paso aprendí a realizar un análisis estático y dinámico de una aplicación móvil y cosas como ver y entender que tipos de permisos solicitan las aplicaciones y si suponen un peligro para el usuario, y el funcionamiento de los diferentes estándares usados en el análisis de vulnerabilidades.

Por último pasamos a realizar la memoria y plasmar en ella toda la información estudiada y los experimentos realizados sobre las aplicaciones de rastreo de contactos. Para la realización de la memoria nuestros tutores nos recomendaron el uso de la herramienta *LaTeX*, herramienta que usábamos por primera vez. Durante la elaboración de la memoria he aprendido a hacer uso de la herramienta *LaTeX* y he comprobado que es una herramienta muy útil para la realización de este tipo de trabajos, ayudando a mantener un formato adecuado y facilitando la inserción de imágenes y tablas.

7.2. Alonso Martín Zurdo

Durante los primeros meses el primero objetivo fue la de recopilar información en distintos trabajos sobre el protocolo DP-3T, las vulnerabilidades que se pueden realizar y como se podrían mitigar además de distintos modelos de rastreo de contactos, así como las ventajas de cada uno de ellos. Como este protocolo es relativamente nuevo, ha sido más complicado encontrar información relevante para el trabajo, aun con todo aprendiendo a usar el motor de busqueda de *Google Scholar* para encontrar distintos *papers* también se ha aprendido a filtrar la información relevante y resumirla de forma que sea más entendible a la hora de que alguien externo al tema lo lea más adelante.

No fue hasta mas adelante que se publicaron varios *papers* sobre el análisis de vulnerabilidades en el protocolo DP-3T en los que analizaban aplicaciones de todos los países, pero centrándose en el reglamento general de protección de datos y comparando así las aplicaciones que estaban dentro de la UE con las que se desarrollaron fuera de ella. Este trabajo fue clave para enfocar el proyecto y, en concreto como guía para poder interpretar los datos obtenidos por el analizador.

El planteamiento inicial era ejecutar distintos ataques sobre las aplicaciones basados en BLE que, si no esta bien implementado puede dar lugar a serios huecos en la seguridad del sistema. Todo este planteamiento requirió un estudio del protocolo de Bluetooth en profundidad que mas adelante se acabó descartando por un enfoque más general con un analizador automático de vulnerabilidades.

Más adelante empezó la búsqueda de herramientas para realizar un entorno de trabajo válido, en total se barajaron 3 entornos, empezando por la herramienta MobSF instalado en *Windows* de manera nativa. Este primer acercamiento al analizador dio bastantes problemas al ejecutarlo por, como se sabría mas adelante una instalación corrupta de *Python* en el dispositivo.

A continuación se optó por el sistema Linux *Santoku*, que esta especializado para hacer análisis sobre dispositivos móviles, en especial se aprendió a utilizar *Wireshark* que sirve para hacer un análisis de los paquetes enviados a una red, conectándolo al emulador de Android se pretendía analizar el tráfico de las aplicaciones de forma manual. La otra aplicación que aprendió en profundidad fue Drozer, que se trata de un *framework* que utiliza un APK instalado en el móvil para hacer una conexión entre el móvil (emulador) y la máquina anfitriona de *Santoku* para ejecutar ataques como inyecciones SQL. Todo este aprendizaje se hizo de forma teórica pues cuando se fue a ejecutar la herramienta se acabó descubriendo la protección contra la emulación de las aplicaciones y se tuvo que descartar este entorno.

Lo siguiente que se probó fue la distribución de Linux *Androl4b* que, aunque tenía menos herramientas para la simulación de ataques, contenía la herramienta que estaba mejor valorada, que era MobSF. Con esta herramienta se pudieron realizar los análisis estáticos de las aplicaciones sin necesidad de utilizar un dispositivo físico conectado. Aun

así este entorno tenía la limitación de solo poder realizar análisis estático. Al ver esto se optó por instalar la herramienta de forma nativa en Windows, lo que resultó, junto al emulador de Android *Genymotion* en el entorno de trabajo final.

Hasta encontrar una combinación de emulador Android y analizador que permitiera buscar errores en las aplicaciones el tiempo invertido en aprender a manejar los entornos anteriores sirvió más adelante para entender mejor la herramienta final MobSF.

Las aplicaciones que se utilizaron para el análisis se obtuvieron a través de la página web *APKPure* debido a que muchos proyectos oficiales no daban facilidades para compilar el código.

Para agilizar la búsqueda de información se dividió el trabajo de documentación y la especialización fue la búsqueda y entendimiento del funcionamiento de las herramientas de análisis, así como el funcionamiento interno de las mismas. Esto incluye los estándares utilizados por la herramienta MobSF así como las distintas fases y tipos de análisis utilizados por las empresas de software. Este conocimiento está plasmado en el Capítulo 4. Este trabajo de documentación sirvió para entender de manera más crítica las puntuaciones que daba la herramienta MobSF.

Tras analizar las aplicaciones se configuraron distintas tablas recabando los resultados obtenidos al ejecutar el analizador MobSF. Dichas tablas agruparon los resultados según el lugar donde se obtuvieron los errores, como en el código o en los permisos.

Debido al gran volumen de datos que se estaban manejando, se acabó por descartar los datos que el analizador detectaba con ninguna severidad o datos de la estructura. Esta criba redujo los datos en un 40 % del total, lo que hizo el estudio más manejable y, por tanto, más sencillo.

Una vez clasificados los errores se buscó la severidad real de los mismos para comprobar como de grave eran los resultados que estaban marcados como severos o desconocidos. Para este proceso de comprobación se consultaron las páginas web de distintos estándares como CWE o OWASP MASVS, así como también la API de Google para ver el uso de los permisos así como las precondiciones y usos de los mismos. Esto último acabó dando mucha perspectiva con lo que respecta a los datos obtenidos y, sobre todo una visión de como se estaban utilizando ciertos permisos en contra de lo que indicaba la API.

Con toda la información contrastada se escribió la memoria haciendo uso por primera vez del lenguaje de edición *LaTeX* haciendo hincapié en la realización de tablas y figuras además de plasmar los resultados de los análisis y las conclusiones de los mismos. El trabajo más duro fue el de la organización de la información en el documento, pues debido al gran volumen de información empezaba a ser caótica. Las tablas mencionadas anteriormente sirvieron mucho a la hora de compactar y organizar los datos.

Para la realización de las figuras mencionadas anteriormente se utilizó la web *draw.io* que al ser la primera vez que se montaba una figura que compactaba información ayudó la simplicidad de la herramienta para acostumbrarse al entorno. Se hizo una excepción con el diagrama de *Gantt* que se hizo uso de la web *Lucidchart* que había sido utilizada

anteriormente para asignaturas donde se habían hecho proyectos que requerían de este tipo de diagramas.

Todo el uso de la herramienta *LaTeX* supuso un esfuerzo extra ya que, como se ha comentado anteriormente, ha sido la primera toma de contacto con el entorno y ha sido necesario un aprendizaje de la realización de tablas, el uso de glosarios y la adecuada utilización de referencias para dar una estructura adecuada al documento final.

Capítulo 8

Conclusiones y Trabajo Futuro

8.1. Conclusiones

El objetivo de este trabajo era la comprobación de seguridad en distintas aplicaciones europeas de rastreo de contactos basadas en el protocolo DP-3T mediante herramientas automáticas de análisis de vulnerabilidades. Esto se ha visto ralentizado por varios factores.

El primero de esos factores es la novedad del protocolo. Esto provoca que la información sobre la misma sea reducida. El siguiente factor es la inexperiencia en el ámbito del análisis de vulnerabilidades. Esto aumentó la curva de dificultad de entrada a la hora de realizar los experimentos como de interpretar los resultados devueltos por la misma. El último de los factores es la protección contra emulaciones de las aplicaciones. Como se ha mencionado en capítulos anteriores, estas aplicaciones tienen código que impide emularlo en una plataforma como Genymotion. Al principio fue una gran barrera para ejecutar los análisis, pero gracias a la herramienta MobSF acabó por no ser más problema que la situación inicial.

Aun así, el resultado fue satisfactorio y se pudo encontrar la manera de ejecutar los análisis sobre las aplicaciones. Estos análisis se ejecutaron con la ayuda de un analizador automático llamado MobSF que ejecutaba sobre un emulador de Android, Genymotion, las aplicaciones que usan el protocolo DP-3T con la intención de encontrar brechas en la seguridad y con ellas hacer un informe, puntuando según los estándares de la herramienta, estableciendo así un grado de severidad de los errores encontrados. Los estándares de esta herramienta se detalla su funcionamiento íntegramente en el Capítulo 4, estos son CVSS que ayuda a puntuar de forma numérica la gravedad de los resultados, OWASP MASVS que detalla los requisitos mínimos de ciberseguridad requeridos en una aplicación con características similares, y CWE que detalla los ataques que se pueden realizar con determinados errores en la seguridad.

El resultado final de los análisis han sido negativos en cuanto a los permisos otorgados, errores de código entre otros. Como se detalla en el Capítulo 6 las puntuaciones por parte del analizador MobSF han sido bajas en general, con una media de 47.7 de puntuación

y de 6.56 puntos según el estándar CVSS, lo que indica que se encuentran en severidad media, tan solo a 0.44 puntos de empezar a ser considerado con severidad alta. Esto se debe al mal tratamiento de información sensible que se ha detectado y al uso de algoritmos de cifrado conocidos por tener problemas de seguridad, lo que provoca que se desaconseje su uso.

Los errores más críticos son el uso de los algoritmos criptográficos SHA-1 y MD5 que son conocidos por tener colisiones de hash, esto significa que varias entradas a la función de hash pueden dar lograr a la misma salida, creando una brecha de seguridad. Otro error crítico es el uso de varios tipos de localización (precisa y aproximada) Cuando la API de Google pone explícitamente que se debe usar si es necesario, una u otra, pero no ambas.

Todo esto se puede llegar a mejorar con nuevas versiones de las aplicaciones donde se traten todos estos aspectos. Se entiende también el tiempo que puedan haber tenido los equipos de desarrollo de estas aplicaciones puede haber sido muy reducido teniendo en cuenta la espontaneidad de la pandemia del COVID-19 lo que puede haber conllevado a la toma de decisiones poco óptimas durante el desarrollo o más enfocadas en la funcionalidad antes que en la protección de la privacidad y la seguridad de los usuarios finales. Todo esto no quiere decir que la falta de los elementos citados de seguridad se deba a que no les haya parecido relevante reforzar la seguridad si no que pueden haber reforzado otros que no se han sido detectados como errores por ese motivo.

8.2. Trabajo Futuro

Como posibles trabajos futuros podría ser interesante realizar las siguientes acciones:

- Analizar las versiones más actuales de las aplicaciones móviles para comprobar si han mejorado la seguridad de las aplicaciones.
- Analizar, a parte de las aplicaciones europeas, las aplicaciones del resto del mundo.
- Analizar aplicaciones móviles de rastreo de contactos que usen modelos centralizados.
- Utilizar otros tipos de analizadores de aplicaciones móviles que usen estándares mas actualizados y modernos.
- Analizar con más profundidad el generador de número aleatorios para comprobar hasta que punto afecta al funcionamiento de la creación de EphID.
- Simular un ataque real a un dispositivo móvil que tenga instalada una aplicación de rastreo de contactos, cómo por ejemplo un ataque DoS Bluetooth.

Resumen del Trabajo en Inglés

Capítulo 9

Introduction

9.1. Motivation

Due to the global pandemic caused at the beginning of 2020 by the COVID-19, we have seen the raise of the number of people infected by the virus growing at a very high speed, provoking multiple saturations in hospitals, many deaths and leading many countries to impose home confinement for months.

To control the infections and act quickly in case of infections by the virus and prevent its spread, a key job arised, the tracker, which is a person who is dedicated to investigating the close contacts a person who has been tested positive for COVID-19 so they can be warned to do a home quarantine and carry out the corresponding tests.

Due to the rapid spread of the virus, manual tracking becomes very arduous and impossible to achieve, so it is no longer effective, since, for example, if an infected person has been using public transport it is impossible to contact all those people with whom he has shared a wagon. For this reason, the idea of creating a mobile application to make a more effective digital contact tracing arose.

But with tracking through a mobile application, there arises, in people, the concern and fear that with this system their privacy and intimacy will be violated and their data will be exposed to possible hackers, causing users not to use the system and with this, the effectiveness of contact tracing is lost.

9.2. Objectives

The objective of this work is to analyze what kind of vulnerabilities mobile applications for contact tracing that are based on the DP-3T protocol may have. For this, an automatic vulnerability analysis tool for mobile applications is used, with which a static and dynamic analysis of the applications is made that will help us to verify the different types of vulnerabilities that these applications may have and if it is put into risk the security and privacy of the users who use them.

9.3. Workplan

Table 9.1 summarizes the tasks carried out throughout the development of the project. Additionally, a Gantt chart illustrating the tasks and their timing is attached in Figure 9.1.

Identifier	Task
1	Investigation
1.1	Reading and Research of academic articles
2	Documentation
2.1	Search for information about the protocol DP-3T
2.2	Search for information on penetration analysis and different tools
3	Search for workbench
3.1	Testing of different tools
3.2	Final work environment setup
4	Experimentation
4.1	Obtaining APKs of the applications
4.2	Static analysis
4.3	Analysis of the results
5	Report
5.1	Annotations, citations and article compilation
5.2	Report writing

Tabla 9.1: Tasks Performed throughout the Project

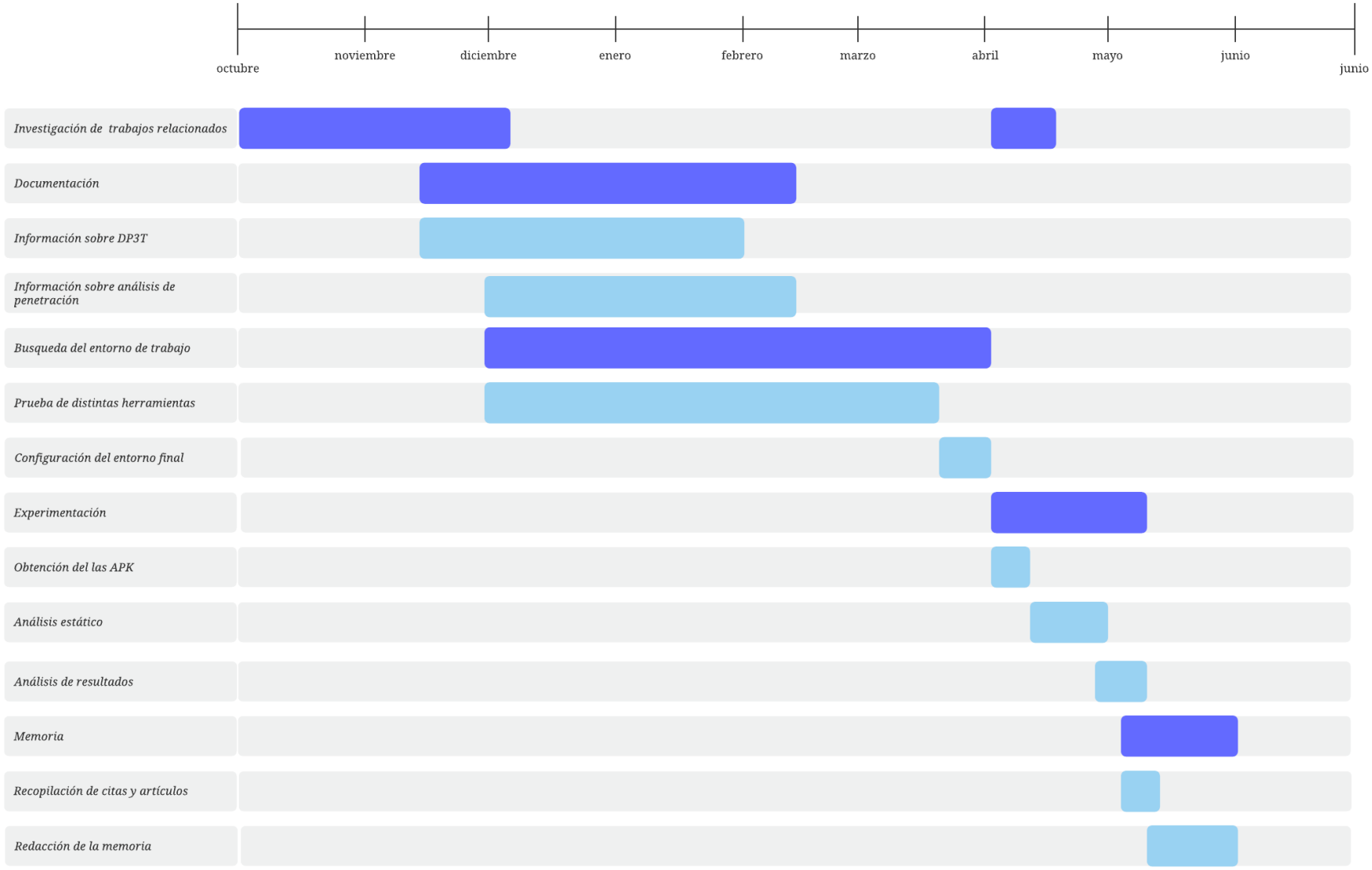


Figura 9.1: Gantt Diagram

9.4. Structure Of This Paper

The rest of the work is organized in seven chapters with the structure that is commented below:

Chapter 2 contextualizes and briefly introduces the origins, concepts and theoretical bases of contact tracing applications in a general way.

In Chapter 3 the decentralized proximity tracking protocol is discussed in more detail. Their types and their operation are described, as well as analyzing and comparing their privacy and security between the different models that exist.

Chapter 4 contextualizes the operation of vulnerability analysis tools. The different types are reviewed and the standards used by the tool used to analyze the applications are explained in detail.

Chapter 5 presents different works related to this, explaining the results obtained.

Chapter 6 details the results of the analyzes carried out on the different European applications and how the analysis laboratory is configured.

Chapter 7 summarizes the individual contributions of each member of the work team.

Chapter 8 concludes this work: briefly summarizing the project experience and analyzing the results of the analyzes. Future work and possible lines of research are presented.

Chapters 9 and 10 are English translations of Chapters 1 and 8, respectively.

Capítulo 10

Conclusions and Future Works

10.1. Conclusions

The objective of this work was to verify security in different European contact tracing applications based on the DP-3T protocol using automatic vulnerability analysis tools. This has been slowed down by several factors.

The first of these factors is the novelty of the protocol. This causes the information on it to be reduced. The next factor is inexperience in the field of vulnerability analysis. This increased the entry difficulty curve when performing the experiments and interpreting the results returned by it. The last factor is protection against application emulations. As mentioned in previous chapters, these applications have code that prevents emulation on a platform like Genymotion. At first it was a big barrier to run the analyzes, but thanks to the MobSF tool it ended up being no more of a problem than the initial situation.

Even so, the result was satisfactory and a way could be found to run the scans on the applications. These analyzes were executed with the help of an automatic analyzer called MobSF that ran on an Android emulator, Genymotion, the applications that use the DP-3T protocol with the intention of finding security gaps and with them make a report, scoring according to the standards of the tool, thus establishing a degree of severity of the errors found. The standards of this tool are fully detailed in Chapter ref Chapter4, these are CVSS which helps to numerically score the severity of the results, OWASP MASVS which details the minimum cybersecurity requirements required in an application with similar characteristics, and CWE which details the attacks that can be carried out with certain security errors.

The final result of the analysis has been negative in terms of the permissions granted, code errors among others. As detailed in Chapter 6, the scores from the MobSF analyzer have been low in general, with an average score of 47.7 and 6.56 points according to the CVSS standard, which indicates that they are in medium severity, only 0.44 points from starting to be considered high severity. This is due to the mishandling of sensitive information that has been detected and the use of encryption algorithms known to have

security problems, which causes their use to be discouraged.

The most critical errors are the use of the SHA-1 and MD5 cryptographic algorithms that are known to have hash collisions, this means that several inputs to the hash function can achieve the same output, creating a security breach. Another critical error is the use of several types of localization (precise and approximate) When the Google API explicitly states that one or the other should be used if necessary, but not both.

All this can be improved with new versions of the applications where all these aspects are dealt with. It is also understood that the time that the development teams of these applications may have had may have been very reduced taking into account the spontaneity of the COVID-19 pandemic, which may have led to suboptimal decisions during development or more focused on functionality rather than protecting the privacy and security of end users. All this does not mean that the lack of the aforementioned security elements is due to the fact that they did not find it relevant to reinforce security, but rather that they may have reinforced others that have not been detected as errors for that reason.

10.2. Future Works

As possible future works, it could be interesting to carry out the following actions:

- Analyze the latest versions of mobile applications to see if they have improved application security.
- Analyze, apart from European applications, applications from the rest of the world.
- Analyze mobile contact tracing applications that use centralized models.
- Use other types of mobile application analyzers that use more updated and modern standards.
- Take a closer look at the random number generator to see to what extent it affects the operation of EphID creation.
- Simulate a real attack on a mobile device that has a contact tracing application installed, such as a DoS Bluetooth attack.

Bibliografía

[CBB+20] Claude Castelluccia, Nataliia Bielova, Antoine Boutet, Mathieu Cunche, Cédric Lauradoux, Daniel Le Métayer, and Vincent Roca. Desire: A third way for a european exposure notification system. https://github.com/3rd-ways-for-EU-exposure-notification/project-DESIRE/blob/master/DESIRE-specification-EN-v1_0.pdf, May 2020.

[CIY20] Hyunghoon Cho, Daphne Ippolito, and Yun William Yu. Contact tracing mobile apps for covid-19: Privacy considerations and related trade-offs. *arXiv preprint arXiv:2003.11511*, 2020.

[Dev15] Android Developers. C2d deprecated. https://developers.google.com/android/c2dm, Jul 2015.

[Dev19] Android Developers. Permisos ble. https://developer.android.com/guide/topics/connectivity/bluetooth-le#permissions, Dec 2019.

[Dev20a] Android Developers. Funciones de hardware para la iu del dispositivo. https://developer.android.com/guide/topics/manifest/uses-feature-element?hl=es-419#location-hw-features, Apr 2020.

[Dev20b] Android Developers. Permisos que implican requisitos de funciones. https://developer.android.com/guide/topics/manifest/uses-feature-element?hl=es-419#permissions, Apr 2020.

[Goo15] Google. Activity recognition api. https://developers.google.com/location-context/activity-recognition, Jun 2015.

[Goo21] Google. Permisos de ubicación. https://developers.google.com/maps/documentation/android-sdk/location?hl=es-419#location_permissions, Apr 2021.

[Gvi20] Yaron Gvili. Security analysis of the covid-19 contact tracing specifications by apple inc. and google inc. *IACR Cryptol. ePrint Arch.*, 2020:428, 2020.

[HWMF21] Majid Hatamian, Samuel Wairimu, Nurul Momen, and Lothar Fritsch. A privacy and security analysis of early-deployed covid-19 contact tracing android apps. *Empirical Software Engineering*, 26(3):1–51, 2021.

[IMP] IMPERVA. Penetration testing. https://www.imperva.com/learn/application-security/penetration-testing/.

[JD20] Nele Quast Jenny Wanger Sameer Halai Andreas Gebhard Kate Gallagher Jason D'Orazio, Christoph Steinlehner. Tcn ux recommendations whitepaper. https://github.com/TCNCoalition/whitepapers/blob/master/tcn-ux-recommendations-whitepaper-v1.pdf, June 2020.

[LR] Puneet Mehta Linda Rosencrance. pen test (penetration testing). https://searchsecurity.techtarget.com/definition/penetration-testing.

[MKHR+20] Tania Martin, Georgios Karopoulos, José L Hernández-Ramos, Georgios Kambourakis, and Igor Nai Fovino. Demystifying covid-19 digital contact tracing: A survey on frameworks and mobile apps. *Wireless Communications and Mobile Computing*, 2020, 2020.

[OWA17] OWASP. Owasp top ten. https://owasp.org/www-project-top-ten/, 2017.

[PLO21] MITRE PLOVER, Eric Dalci. Cwe-200: Exposure of sensitive information to an unauthorized actor. https://cwe.mitre.org/data/definitions/200.html, Mar 2021.

[PM07] Sasha Romanosky Peter Mell, Karen Scarfone. A complete guide to the common vulnerability scoring system. https://www.first.org/cvss/v2/guide, Jun 2007.

[RWI+20] Ronald L Rivest, DJ Weitzner, LC Ivers, Israel Soibelman, and MA Zissman. Pact: Private automated contact tracing. *Retrieved December*, 2:2020, 2020.

[SS20] Carlos Holguera Sven Schleire, Jeroen Willemsen. Mobile Application Security Verification Standard. In *Mobile Application Security Verification Standard*, pages 4 – 17, March 2020.

[TPH+20] Carmela Troncoso, Mathias Payer, Jean-Pierre Hubaux, Marcel Salathé, James Larus, Edouard Bugnion, Wouter Lueks, Theresa Stadler, Apostolos Pyrgelis, Daniele Antonioli, et al. Decentralized privacy-preserving proximity tracing. *arXiv preprint arXiv:2005.12273*, 2020.

[Vau20] Serge Vaudenay. Centralized or decentralized? the contact tracing dilemma. Technical report, 2020.