

JOSE MANUEL ORTEGA CANDEL

# DESARROLLO SEGURO EN INGENIERÍA DEL SOFTWARE

APLICACIONES SEGURAS CON ANDROID, NODEJS, PYTHON Y C++



<https://dogramcode.com/bloglibros>



Dogram

# **Desarrollo seguro en ingeniería del software**

Aplicaciones seguras con Android, NodeJS, Python y C++

José Manuel Ortega Candel

Acceda a [www.marcombo.info](http://www.marcombo.info)  
para descargar gratis  
*contenido adicional*  
complemento imprescindible de este libro

Código: **SOFTHAR51**

<https://dogramcode.com/bloglibros>



Dogram

# **Desarrollo seguro en ingeniería del software**

Aplicaciones seguras con Android, NodeJS, Python y C++

José Manuel Ortega Candel



*Desarrollo seguro en ingeniería del software*

© 2020 José Manuel Ortega Candel

Primera edición, 2020

© MARCOMBO, S. L. 2020

[www.marcombo.com](http://www.marcombo.com)

«Cualquier forma de reproducción, distribución, comunicación pública o transformación de esta obra solo puede ser realizada con la autorización de sus titulares, salvo excepción prevista por la ley. Diríjase a CEDRO (Centro Español de Derechos Reprográficos, [www.cedro.org](http://www.cedro.org)) si necesita fotocopiar o escanear algún fragmento de esta obra».

ISBN: 978-84-267-2800-5

D.L.: B-2094-2020

Impreso en Servicepoint  
*Printed in Spain*

Este libro va dedicado a aquellos que me han seguido,  
me siguen y me seguirán, no importa cuál sea el camino escogido,  
en algún lugar nos encontraremos.

<https://dogramcode.com/bloglibros>



Dogram

## Índice

<b>Capítulo 1. Introducción al desarrollo seguro .....</b>	<b>15</b>
<b>1.1 Propiedades del software seguro .....</b>	<b>17</b>
<b>1.2 Principios de diseño seguro de aplicaciones .....</b>	<b>18</b>
<b>1.2.1 Minimizar el área de la superficie de ataque.....</b>	<b>19</b>
<b>1.2.2 Seguridad por defecto .....</b>	<b>19</b>
<b>1.2.3 Privilegios mínimos.....</b>	<b>19</b>
<b>1.2.4 Validación de datos de entrada .....</b>	<b>20</b>
<b>1.2.5 Defensa en profundidad .....</b>	<b>20</b>
<b>1.2.6 Control seguro de errores.....</b>	<b>20</b>
<b>1.2.7 Separación de funciones .....</b>	<b>21</b>
<b>1.2.8 Evitar la seguridad por oscuridad .....</b>	<b>21</b>
<b>1.3 Análisis de requisitos de seguridad.....</b>	<b>21</b>
<b>Capítulo 2. Aspectos fundamentales de desarrollo seguro.....</b>	<b>23</b>
<b>2.1 Controles proactivos.....</b>	<b>24</b>
<b>2.2 OWASP (Open Web Application Security Project) .....</b>	<b>25</b>
<b>2.3. OWASP Mobile Security Project .....</b>	<b>29</b>
<b>2.4 Controles proactivos OWASP .....</b>	<b>31</b>
<b>2.4.1 Verificación de la seguridad desde las primeras etapas de desarrollo .....</b>	<b>32</b>
<b>2.4.2 Validación de las entradas del cliente .....</b>	<b>32</b>
<b>2.4.3 Desbordamientos del búfer .....</b>	<b>35</b>
<b>2.4.4 Gestión de sesiones .....</b>	<b>35</b>
<b>2.4.5 Implementación de controles de acceso .....</b>	<b>36</b>
<b>2.4.6 Implementación de controles de identidad y autenticación.....</b>	<b>36</b>
<b>2.4.7 Autenticación por múltiples factores.....</b>	<b>37</b>
<b>2.4.8 Manejo de errores y excepciones.....</b>	<b>38</b>
<b>2.5 Ataques en aplicaciones web.....</b>	<b>39</b>
<b>2.5.1 Vectores de ataque.....</b>	<b>39</b>
<b>2.5.2 Cross-site scripting (XSS).....</b>	<b>40</b>
<b>2.5.3 Cross-site request forgery (CSRF).....</b>	<b>42</b>
<b>2.5.4 Seguridad en las redirecciones.....</b>	<b>44</b>
<b>2.6 SQL Injection: parametrización de las consultas en bases de datos .....</b>	<b>45</b>
<b>2.6.1 Introducción a SQL Injection .....</b>	<b>46</b>
<b>2.6.2 Problemas que pueden causar este tipo de ataques .....</b>	<b>47</b>
<b>2.6.3 Ejemplo de inyección de SQL .....</b>	<b>48</b>
<b>2.6.4 Escapar caracteres especiales utilizados en las consultas SQL.....</b>	<b>50</b>
<b>2.6.5 Delimitación de los valores de las consultas.....</b>	<b>51</b>
<b>2.6.6 Uso de sentencias preparadas parametrizadas .....</b>	<b>52</b>
<b>2.6.7 Uso de procedimientos almacenados .....</b>	<b>53</b>

2.7 Seguridad en AJAX .....	55
<b>Capítulo 3. Herramientas OWASP .....</b>	<b>63</b>
3.1 DefectDojo .....	63
3.2 SonarQube .....	69
3.2.1 El cuadro de mando de SonarQube .....	70
3.2.2 Issues por nivel de criticidad .....	72
3.2.3 Perfiles de calidad .....	73
3.2.4 Reglas SonarQube .....	76
3.2.5 Informes de seguridad en SonarQube .....	78
3.2.6 SonarQube Plugins .....	79
3.2.7 Vulnerabilidades más comunes y explotadas .....	83
3.3 Find Security Bugs .....	83
3.3.1 Inyección potencial de Android SQL .....	85
3.3.2 Abrir un socket sin cifrar .....	85
3.4 LGTM .....	86
3.5 OSS Index .....	89
3.6 Snyk .....	91
3.7 Otras herramientas de análisis estático .....	92
3.8 Checklist de seguridad .....	93
<b>Capítulo 4. Seguridad en aplicaciones Android .....</b>	<b>95</b>
4.1 Introducción al protocolo HTTPS .....	95
4.1.1 Conceptos básicos sobre certificados .....	95
4.1.2 Despliegues en producción .....	97
4.1.3 Certificado de servidor autofirmado .....	100
4.1.4 CA no encontrada dentro de la cadena de certificados .....	100
4.1.5 Configuración de seguridad .....	101
4.1.6 Actualización de proveedores criptográficos .....	102
4.1.7 Android Certificate Pinning .....	102
4.1.8 Cifrado extremo a extremo .....	104
4.1.9 Firmando una aplicación Android .....	104
4.2 Principios fundamentales de desarrollo en Android .....	106
4.2.1 Componentes en Android .....	107
4.2.2 Android Lint .....	109
4.3 Ingeniería inversa en Android .....	113
4.3.1 ADB (Android Debug Bridge) .....	114
4.3.2 Dex2jar .....	114
4.3.3 JD-GUI .....	116
4.3.4 jadx - Dex to Java decompiler .....	116
4.3.5 Apktool .....	118
4.3.6 Código smali y MobyLizer .....	121
4.3.7 Androwarn .....	123
4.3.8 Mobile Security Framework (MobSF) .....	125
4.3.9 ClassyShark .....	127
4.3.10 Drozer .....	128
4.3.11 QARK .....	134

4.3.12 SanDroid.....	137
4.3.13 Yaazhini .....	138
4.4 Buenas prácticas de desarrollo seguro en Android.....	139
4.4.1 Seguridad en AndroidManifest.xml.....	140
4.4.2 Modelo de permisos en Android .....	142
4.4.3 Asegurando la capa de aplicación .....	143
4.4.4 Evitar almacenar datos confidenciales en el dispositivo.....	144
4.4.5 Uso adecuado del componente WebView .....	145
4.4.6 Usar método POST para el envío de datos confidenciales.....	146
4.4.7 Validar los certificados SSL/TLS.....	147
4.4.8 Restricción de uso de la aplicación a determinados dispositivos.....	147
4.4.9 Gestión de logs .....	148
4.4.10 Comprobar la conexión de red.....	149
4.4.11 Realizar operaciones de red en un hilo separado .....	149
4.4.12 Permisos de localización.....	151
4.4.13 Optimizar el código en Android y memoria caché.....	151
4.4.14 Implementación segura de proveedores de contenido .....	153
4.4.15 Almacenamiento de preferencias compartidas (SharedPreferences) .....	154
4.4.16 Almacenamiento seguro de preferencias .....	157
4.4.17 Almacenamiento en ficheros .....	160
4.4.18 Almacenamiento externo .....	160
4.4.19 Implementación segura de Intents .....	162
4.4.20 Implementación segura de servicios.....	163
4.4.21 Implementación segura de broadcast receivers.....	163
4.4.22 Implementación segura de content providers .....	164
4.4.23 Invocar actividades de forma segura .....	164
4.4.24 Implementar almacenamiento de datos seguro.....	165
4.4.25 Algoritmos criptográficos .....	170
4.4.26 Uso de java.util.String para almacenar información sensible .....	172
4.4.27 Proteger la configuración de la aplicación .....	172
4.4.28 Cifrado en base de datos SQLite .....	173
4.4.29 Optimización y ofuscación del código con ProGuard .....	174
4.5 Metodología OASAM .....	177
<b>Capítulo 5. Seguridad en proyectos NodeJS .....</b>	<b>179</b>
5.1 Introducción a NodeJS .....	179
5.2 Modelo Event-Loop.....	181
5.3 Gestión de paquetes .....	182
5.4 Programación asíncrona .....	184
5.5 Problema del código piramidal .....	185
5.6 Módulo para administrar el sistema de archivos .....	186
5.7 Módulo http .....	188
5.8 Utilización del Middleware Express .....	190
5.8.1 Middleware de nivel de aplicación .....	190
5.8.2 Middleware de nivel de direccionamiento .....	191
5.8.3 Middleware de terceros .....	191
5.9 Autenticación en NodeJS .....	192
5.9.1 Auth0.....	192
5.9.2 PassportJS.....	192

<b>5.10 OWASP top 10 en NodeJS .....</b>	<b>193</b>
5.10.1 OWASP NodeGOAT.....	193
5.10.2 Inyección de código.....	196
5.10.3 Función eval.....	197
5.10.4 Ataque de denegación de servicio .....	198
5.10.5 Uso de patrones y expresiones regulares .....	198
5.10.6 Acceso al sistema de ficheros .....	200
5.10.7 Inyección de SQL .....	202
5.10.8 Inyección de NoSQL.....	203
5.10.9 Inyección de logs .....	205
5.10.10 Gestión de la sesión y autenticación .....	205
5.10.11 Protegiendo credenciales de usuario.....	206
5.10.12 Tiempo de espera de sesión y protección de cookies.....	208
5.10.13 Secuestro de sesión (Session hijacking).....	212
5.10.14 Módulo helmet.....	213
5.10.15 Cross-site scripting (XSS).....	216
5.10.16 Referencias de objetos directos inseguros .....	218
5.10.17 Mala configuración de seguridad.....	219
5.10.18 Deshabilitar fingerprinting.....	221
5.10.19 Exposición de datos sensibles.....	222
5.10.20 Configurando SSL/TLS.....	222
5.10.21 Forzar peticiones HTTPS.....	225
5.10.22 Falta de control de acceso.....	227
5.10.23 Redirecciones no validadas .....	228
5.10.24 Denegación de servicio mediante expresiones regulares.....	229
5.10.25 Validar datos de entrada con validator.....	230
5.10.26 Validar datos de entrada con express-validator.....	231
5.10.27 Configuración de cabeceras HTTP .....	233
5.10.28 Política de seguridad de contenido (CSP) .....	235
5.10.29 Cross-site request forgery (CSRF).....	236
5.10.30 Ejecutar código JavaScript de forma aislada .....	238
5.10.31 Uso de componentes con vulnerabilidades conocidas.....	239
5.10.32 NodeJsScan .....	242
<b>Capítulo 6. Seguridad en proyectos Python .....</b>	<b>245</b>
6.1 Componentes inseguros en Python.....	248
6.2 Validación incorrecta de entrada/salida.....	249
6.3 Función eval() .....	250
6.4 Serialización y deserialización de datos con pickle.....	253
6.5 Ataques de inyección de entrada.....	257
6.5.1 Inyección de comandos .....	258
6.5.2 Inyección de SQL .....	264
6.6 Acceso seguro al sistema de archivos y ficheros temporales .....	266
6.7 Inyección de XSS .....	268
6.8 Inyección de SSTI .....	270
6.9 Servicios para comprobar la seguridad de proyectos Python .....	273
6.9.1 Pyup .....	273
6.9.2 LGTM en proyectos Python.....	275

6.9.3 Sanitización de las URL .....	275
6.9.4 Uso de un algoritmo criptográfico roto o débil .....	276
6.9.5 Peticiones con requests sin validación de certificado .....	276
6.9.6 Uso de la versión insegura SSL/TLS .....	277
6.9.7 Deserialización de entrada no confiable .....	277
6.9.8 Vulnerabilidades de XSS .....	278
6.9.9 Exposición de información a través de una excepción .....	279
6.9.10 Conexión con hosts remotos mediante SSH utilizando Paramiko .....	280
6.10 Análisis estático de código Python .....	280
6.10.1 Python Taint .....	281
6.10.2 Bandit .....	282
6.10.3 Hawkeye .....	286
6.10.4 DLint .....	288
6.11 Gestión de dependencias .....	289
6.11.1 Instalación de dependencias .....	290
6.11.2 Requires.io .....	291
6.11.3 Safety .....	292
6.11.4 Paquetes maliciosos en PyPI .....	292
6.12 Python code checkers .....	293
6.12.1 Pyflakes .....	293
6.12.2 PyLint .....	294
6.13 Escáner de seguridad de aplicaciones web .....	295
6.13.1 WAScan .....	295
6.13.2 SQLmap .....	296
6.13.3 XSScrapy .....	297
6.14 Seguridad en Django .....	298
6.14.1 Protección ante ataques XSS .....	299
6.14.2 Protección ante ataques CSRF .....	299
6.14.3 Protección de inyección de SQL .....	300
6.14.4 Protección de clickjacking .....	301
6.14.5 SSL/HTTPS .....	301
6.15 Otras herramientas de seguridad .....	302
6.15.1 Yosai .....	302
6.15.2 Flask-Security .....	303
6.15.3 OWASP Python Security Project .....	304
<b>Capítulo 7. Análisis estático y dinámico en aplicaciones C/C++ .....</b>	<b>305</b>
7.1 Análisis estático .....	306
7.1.1 Code Analyzer .....	307
7.2 Análisis estático de código C/C++ .....	308
7.2.1 Flawfinder .....	308
7.2.2 Clang .....	312
7.2.3 Uso de variables sin inicializar .....	312
7.2.4 Uso inseguro de funciones .....	314
7.2.5 RATS .....	315
7.2.6 Vulnerabilidad cadena de formato (format string) .....	315
7.2.7 Pscan para detectar vulnerabilidades format string .....	316
7.2.8 Buffer overflow .....	317

7.2.9 Tipos de heap overflow .....	320
7.2.10 Vulnerabilidad use after free .....	321
7.2.11 Dereference after free .....	322
7.2.12 Vulnerabilidad double free .....	323
7.2.13 Vulnerabilidad off by one .....	324
7.2.14 Vulnerabilidades race condition .....	325
7.2.15 Vulnerabilidad integer overflow .....	327
7.2.16 Uso de StackOverflow .....	328
7.3. Análisis dinámico .....	329
7.3.1. Análisis dinámico en C/C++ con Valgrind .....	330
7.4 Herramientas de análisis .....	333
<b>Capítulo 8. Metodologías de desarrollo .....</b>	<b>338</b>
8.1. Metodologías de desarrollo de software seguro .....	338
8.1.1 Correctness by Construction (CbyC) .....	339
8.1.2 Security Development Lifecycle (SDLC) .....	341
8.1.3 Fases de la metodología SDLC .....	342
8.1.4 Vulnerabilidades en SDLC .....	346
8.1.5 Tipos de SDLC .....	349
8.1.5.1 Microsoft Trustworthy Computing SDL .....	349
8.1.5.2 CLASP .....	350
8.1.5.3 TSP-Secure .....	351
8.1.5.4 Oracle Software Security Assurance .....	352
8.1.5.5 Propuesta híbrida .....	352
8.1.6 Tipos de pruebas de seguridad SDLC .....	353
8.1.7 Conclusiones de ciclo de vida de desarrollo de software (SDLC) .....	355
8.2 Modelado de amenazas .....	356
8.2.1 Modelado de amenazas con STRIDE .....	358
8.3 Perspectiva del atacante .....	361
8.4 Patrones de ataque .....	362
8.5 OWASP Testing Framework y perfiles para pruebas de seguridad .....	363
8.6 OWASP Security Knowledge Framework (SKF) .....	365
8.7 Seguridad en ingeniería del software .....	374
8.8 Bibliografía y fuentes de información .....	376
8.9 Conclusiones .....	378
<b>Capítulo 9. Glosario de términos .....</b>	<b>380</b>

## **Capítulo 1. Introducción al desarrollo seguro**

La necesidad de desarrollar aplicaciones seguras y, por tanto, tener en cuenta la seguridad en las metodologías de desarrollo es tan importante como tenerla en cuenta para la construcción de edificios o barcos. Desarrollar aplicaciones sin considerar la seguridad es como fabricar un barco sin botes salvavidas o sin los botes necesarios para salvar a todas las personas a bordo en caso de una catástrofe como la del Titanic.

Actualmente, las nuevas tecnologías son un orquestador fundamental en la sociedad de la información, lo cual incluye a las personas y los nuevos tipos de relaciones que se establecen entre ellas. Esta dependencia se halla presente en múltiples escenarios y contextos, desde actividades comerciales y sociales hasta el ámbito militar o la gestión de las infraestructuras críticas nacionales. Ello ha provocado la necesidad creciente de que la tecnología se comporte como se espera, esto es, de acuerdo con sus especificaciones, y de que implemente los mecanismos de seguridad apropiados desde el punto de vista de la confidencialidad, privacidad y seguridad de los datos.

Es cierto que algunos escenarios demandan unos requisitos de seguridad más rigurosos que otros. Sin embargo, la interdependencia entre sistemas de información en un contexto global como el actual nos sugiere la necesidad de una aproximación global, lo que hace de la confiabilidad en la tecnología un concepto de aplicación universal y no particular a un campo específico.

A pesar de ello, se ha demostrado ampliamente que la tecnología actual no ha alcanzado el grado de madurez y seguridad requerido, pues es vulnerable a una gran cantidad de amenazas. Según estudios realizados, cerca del 90 % de los incidentes de seguridad del software están causados por atacantes que han explotado fallos conocidos en él. Además, en un análisis de 45 aplicaciones de negocio electrónico, se mostró que el 70 % de los fallos del software estaban relacionados con el diseño. De media, un ingeniero de software experimentado y capacitado introduce un fallo por cada nueve líneas de código. Aproximadamente, un millón de líneas de código podrían tener de mil a cinco mil fallos, los cuales podrían ser el origen de alrededor de cien vulnerabilidades en producción. Además, se ha demostrado que el software se degrada con el tiempo, y lo que vale en un momento en concreto puede no valer al cabo de seis meses.

En este sentido, la ingeniería de software de desarrollo seguro es una disciplina de la ingeniería del software en la que se busca proporcionar garantías objetivas respecto a la seguridad del software desarrollado. Para ello, se aplican

mecanismos y se producen evidencias durante el proceso que garantizan que el software contiene las propiedades de seguridad que se le requieran.

En particular, y partiendo del axioma "la seguridad absoluta no es alcanzable", el software seguro es capaz de minimizar el impacto de la mayoría de los ataques, tolerar aquellos que no pueda resistir, y recuperarse rápidamente y con el menor impacto de aquellos otros que no pueda tolerar.

El campo de la investigación en ingeniería de software seguro se centra en desarrollar métodos eficaces y herramientas que permitan al desarrollador, durante el ciclo de vida del desarrollo de software, la implementación segura desde las primeras fases de análisis hasta las etapas de más largo plazo de desarrollo.

Uno de los grandes retos actuales para la aplicación de cualquier técnica orientada a mejorar la calidad y la seguridad del software es el cambio de mentalidad en el desarrollador, así como, en muchos casos, la compleja integración de estas técnicas en los procesos existentes, con los costes que ello conlleva. Los hechos nos demuestran que resulta urgente mejorar los procesos de ingeniería del software en pro de unos sistemas de información más robustos y seguros que aporten la confianza necesaria a la sociedad.

El desarrollo seguro es una parte importante de la seguridad informática, englobado dentro del ámbito de la prevención. Podríamos definir que un programa seguro es aquel capaz de seguir realizando las funciones para las que ha sido creado en todo momento, y capaz de evitar que la existencia de errores en él pueda ser utilizada como puente para la realización de acciones que pongan en peligro la integridad, confidencialidad o disponibilidad del resto de elementos del sistema en el que se está ejecutando.

Por tanto, la programación segura engloba el conjunto de técnicas, normas y conocimientos que permiten crear programas que no puedan ser modificados de forma ilegítima con fines maliciosos y que estén carentes de fallos que puedan comprometer la seguridad del resto de elementos del sistema con el que interactúan.

Los objetivos de la utilización de prácticas de software seguras son las siguientes:

- Los fallos explotables y otros puntos débiles se eliminan en la mayor medida posible.
- La probabilidad de que los desarrolladores puedan producir errores y vulnerabilidades explotables o puertas traseras en el software se reduce o elimina en gran medida.

- El software es resistente y tolerante a posibles ataques para apoyar el cumplimiento de la misión de la organización.

Uno de los principales objetivos radica en ayudar a desarrolladores de software a implementar medidas preventivas que ayuden a mitigar o reducir lo máximo posible errores comunes de programación y que pueden ser aprovechados por un posible atacante para comprometer una aplicación.

El libro presenta gran interés para cualquier desarrollador responsable de una página web o un administrador de grandes portales corporativos, así como para analistas y programadores de una empresa de las denominadas *software factories*. El lector estará en disposición de definir procedimientos y políticas de desarrollo de código dentro de una misma organización, y también de establecer los controles necesarios para el desarrollo de software seguro.

## 1.1 Propiedades del software seguro

Si bien con las metodologías y los estándares relacionados con el desarrollo y la construcción de software se busca mantener altos niveles de confiabilidad y control de la solución informática, la seguridad informática y sus principios de diseño seguro no suelen ser parte formal u obligatoria de dichos estándares.

Cuando una aplicación transmite o almacena información confidencial, la aplicación es responsable de garantizar que los datos almacenados y transferidos estén cifrados y no puedan obtenerse, alterarse o divulgarse fácilmente de forma ilícita. En general, se suele decir que el objetivo fundamental de la seguridad informática se basa en preservar los siguientes puntos:

- **Confidencialidad.** El software debe asegurar que cualquiera de sus características, los activos que administra y su contenido son accesibles solo para las entidades autorizadas e inaccesibles para el resto. El acceso a la información está limitado a usuarios autorizados. Los datos deben protegerse de la observación o divulgación no autorizadas tanto en tránsito como almacenadas.
- **Integridad.** El software y los activos del sistema solo pueden ser modificados por usuarios autorizados. Esta propiedad se debe preservar durante el desarrollo del software y su ejecución. Los datos han de protegerse al ser creados, alterados o borrados maliciosamente por atacantes no autorizados.

- **Disponibilidad.** El software debe estar operativo y ser accesible a sus usuarios autorizados siempre que se requiera, así como ha de desempeñarse con una *performance* adecuada para que los usuarios puedan realizar sus tareas de forma correcta y dar cumplimiento a los objetivos de la organización que lo utiliza. El acceso a los activos en un tiempo razonable está garantizado para usuarios autorizados. Los datos deben encontrarse disponibles para los usuarios autorizados, según sea necesario.

Para las entidades que actúan como usuarios, se requieren dos propiedades adicionales:

- **Trazabilidad.** Todas las acciones relevantes relacionadas con la seguridad de una entidad que actúa como usuario se deben registrar y trazar con el objetivo de disponer de datos para la realización de auditorías; de esta forma, la trazabilidad debe ser posible tanto durante la ocurrencia de las acciones registradas como *a posteriori*.
- **No repudio.** Constituye la habilidad de prevenir que una entidad que actúa como usuario desmienta o niegue la responsabilidad de acciones que han sido ejecutadas.

Estas propiedades básicas son las más utilizadas normalmente para describir la seguridad de los sistemas y aplicaciones. Un ataque con éxito de **inyección de SQL** en una aplicación para extraer información de identificación personal de su base de datos sería una violación de la propiedad de **confidencialidad**. El éxito de un ataque **cross-site scripting (XSS)** en contra de una aplicación web podría dar lugar a una violación tanto en su **integridad** como en su **disponibilidad**. Un ataque con éxito de desbordamiento de búfer que inyecta código con el objetivo de obtener y modificar la información de usuarios sería una violación de las cinco propiedades básicas de seguridad.

## 1.2 Principios de diseño seguro de aplicaciones

En consecuencia, los efectos de vulnerar la seguridad del software se pueden describir en términos de los efectos sobre estas propiedades fundamentales. A continuación se sugieren una serie de principios orientados al diseño seguro de aplicaciones:

### **1.2.1 Minimizar el área de la superficie de ataque**

Cada característica que se añade a una aplicación incrementa su complejidad y aumenta también el riesgo de aplicación en conjunto. Una nueva característica implica un nuevo punto de ataque. Uno de los factores clave para reducir el riesgo de una aplicación reside en la reducción de la superficie de ataque. Pueden eliminarse posibles puntos de ataque si se deshabilitan módulos o componentes innecesarios para la aplicación; por ejemplo, si la aplicación no utiliza el almacenamiento en caché de resultados, sería recomendable deshabilitar dicho módulo. De esta manera, si se detecta una vulnerabilidad de seguridad en ese módulo, la aplicación no se verá amenazada.

### **1.2.2 Seguridad por defecto**

Existen muchas maneras de entregar una experiencia *out of the box* (“lista para usar”) a los usuarios. Sin embargo, por defecto, la experiencia debe ser segura, mas facilitar, no obstante, la capacidad de reducción del nivel de seguridad si el usuario lo cree necesario.

Del lado de los desarrolladores, constituye una práctica habitual utilizar opciones de configuración de seguridad reducidas para evitar que dichas configuraciones compliquen el desarrollo. Si, para su implementación, la aplicación requiere de características que obligan a reducir o cambiar la configuración de seguridad predeterminada, se recomienda estudiar sus efectos y consecuencias, para lo que hay que realizar pruebas específicas de auditoría de seguridad.

### **1.2.3 Privilegios mínimos**

Según el principio del mínimo privilegio, se recomienda que las cuentas de usuario tengan la mínima cantidad de privilegios necesarios. Asimismo, se aconseja que los procesos se ejecuten únicamente con los privilegios necesarios para completar sus tareas. De esta manera, se limitan los posibles daños que podrían producirse si se ve comprometido el proceso.

Si un atacante llegase a tomar el control de un proceso en un servidor o comprometiese una cuenta de usuario, los privilegios concedidos determinarán, en gran medida, los tipos de operaciones que podrá llegar a realizar dicho atacante; por ejemplo, si cierto servidor solo requiere acceso de lectura a la tabla de una base de datos, bajo ninguna condición deberían darse privilegios administrativos a los usuarios si no resultan necesarios.

#### **1.2.4 Validación de datos de entrada**

Se ha de garantizar que la aplicación sea robusta ante todas las posibles formas de entrada de datos, ya sean proporcionados por el usuario, por la infraestructura, por entidades externas o por bases de datos.

Una premisa fundamental reside en no confiar en los datos que el usuario pueda introducir, ya que este tiene todas las posibilidades de manipularlos. La debilidad de seguridad más común en aplicaciones es la falta de validación apropiada de las entradas del usuario o del entorno. Esta debilidad origina casi todas las principales vulnerabilidades en las aplicaciones, tales como inyecciones de código, inserción de secuencias de comandos, ataques al sistema de archivos o desbordamientos de memoria.

Las aplicaciones deben validar todos los datos introducidos por el usuario antes de realizar cualquier operación con ellos. La validación podría incluir el filtrado de caracteres especiales o el control de la longitud de los datos introducidos. Esta medida preventiva protege a la aplicación de usos incorrectos accidentales o de ataques deliberados por parte de usuarios.

#### **1.2.5 Defensa en profundidad**

Con “defensa en profundidad”, nos referimos a definir una estrategia de seguridad estándar en la que se establezcan varios controles de defensa en cada una de las capas y los subsistemas de la aplicación. Estos puntos de control ayudan a garantizar que solo los usuarios autenticados y autorizados puedan obtener el acceso a la siguiente capa y a sus datos.

#### **1.2.6 Control seguro de errores**

Es necesario controlar las respuestas cuando se produce algún error y no mostrar en ellas información que pudiera ayudar al atacante a descubrir datos acerca de cómo ha sido desarrollada la aplicación o cómo funcionan sus procedimientos. La información detallada de los errores producidos no debería ser mostrada al usuario, sino que tendría que ser enviada al fichero de log correspondiente. Una simple página de error 403 indicando un acceso denegado puede decir a un escáner de vulnerabilidades que un directorio existe y puede proporcionar a un atacante información que le permita realizar un mapa de la estructura de directorios de la aplicación.

### **1.2.7 Separación de funciones**

Un punto importante consiste en la separación de funciones entre los distintos perfiles de la aplicación; por ejemplo, en una tienda de subastas online, un usuario vendedor pone en subasta un producto. Si la separación de funciones se encuentra correctamente implementada, este mismo usuario no debe poder participar en la puja por el artículo. Por otra parte, determinados roles deben contar con un nivel de confianza más elevado que los usuarios normales, como en el caso del administrador, que posee privilegios avanzados, como administrar al resto de usuarios del sistema o configurar políticas de contraseñas, así como definir los diferentes parámetros de la aplicación.

### **1.2.8 Evitar la seguridad por oscuridad**

La seguridad de una aplicación no debería depender del secreto o confidencialidad de su diseño o implementación. Si se intentan ocultar secretos mediante el uso de nombres de variables engañosos o de ubicaciones de archivos no habituales, no se estará mejorando la seguridad. La seguridad basada en la oscuridad es un control de seguridad débil, especialmente si se trata del único control. Esto no significa que mantener secretos constituya una mala idea; significa que la seguridad de los sistemas clave no debería basarse, exclusivamente, en mantener detalles ocultos.

Por ejemplo, la seguridad de una aplicación no debería basarse en mantener en secreto el conocimiento del código fuente. La seguridad ha de fundamentarse en muchos otros factores, incluyendo políticas de contraseñas, diseño de una arquitectura sólida tanto a nivel de aplicación como a nivel de comunicaciones de red y realización periódica de controles de auditoría. Un ejemplo práctico es Linux, cuyo código fuente es open source y está disponible para todo el mundo y, aun así, se trata de un sistema operativo seguro y robusto.

## **1.3 Análisis de requisitos de seguridad**

La seguridad debería ser un aspecto de alta prioridad cuando estamos desarrollando y probando aplicaciones que manejan datos confidenciales a nivel de autenticación y autorización. El análisis de los requisitos de seguridad debe ser parte de la primera fase de análisis de los requisitos de cada proyecto de aplicaciones móviles. La siguiente lista lo puede ayudar con el análisis de los requisitos de seguridad:

- Identificar los posibles roles de usuario, así como sus limitaciones y permisos dentro de la arquitectura (app y backend).
- Identificar qué tipo de enfoques y herramientas de pruebas de seguridad se requieren para lograr un buen nivel de seguridad.
- Identificar el impacto que tienen las funcionalidades a nivel de usuario en la seguridad a nivel frontend y backend.

La mayoría de los fallos de seguridad de aplicaciones se pueden prevenir mediante la integración de los procesos y buenas prácticas de seguridad desde las primeras etapas de desarrollo. La planificación de una estrategia inicial de diseño de aplicaciones y mantenimiento de la seguridad en mente todo el tiempo permite reducir las posibilidades de los riesgos de seguridad que surgen durante las últimas etapas de desarrollo de aplicaciones.

## **Capítulo 2. Aspectos fundamentales de desarrollo seguro**

Resulta importante hacer entender que la seguridad debe formar parte del ciclo de desarrollo como algo integrado dentro de las herramientas de integración continua, no como una fase separada, sino como parte integral del desarrollo. El objetivo, al final, es tratar de minimizar los riesgos que pueda suponer la seguridad de las aplicaciones.

Una buena práctica sería poner toda la lógica de negocio posible en la parte de servidor/backend, para evitar que alguien pueda hacer ingeniería inversa en la aplicación y averiguar determinados datos. En el caso de que haya que introducir datos sensibles, se deberían encriptar para evitar ataques.

Cuando se habla de seguridad, los atacantes tienen una ventaja sobre los desarrolladores. Esta se expone en cuatro principios que la ilustran muy bien:

- Quien defiende debe preocuparse por proteger todos los puntos; el atacante seleccionará solo uno: el más débil.
- Los encargados de la defensa van a procurar defenderse de ataques conocidos; el atacante probará nuevas formas de llevar adelante sus ataques.
- Quien se encarga de defender debe estar en constante estado de vigilancia.
- El defensor ha de respetar reglas; el atacante, normalmente, juega con sus propias reglas.

Cabe resaltar que un atacante necesita solamente un pequeño error, una vulnerabilidad, para lograr su cometido. Si dentro de la empresa se descuidan los temas de seguridad por acelerar la operatividad del negocio, podemos estar dejando la puerta abierta a que se comprometa la seguridad de la información. Ser responsables con los procesos constituye la mejor defensa, y no está de más preguntarse si es mejor invertir unas semanas más en desarrollo que perder reputación y dinero en un instante por un incidente de seguridad.

Controlar un sistema resulta parte fundamental para garantizar un mejoramiento continuo. Por esta razón, vamos a ver algunos controles que deberían tenerse en cuenta para que una aplicación garantice la integridad y disponibilidad de la información y, además, se pueda mejorar progresivamente.

En particular, para aquellos que están directamente relacionados con el desarrollo, la responsabilidad es mayor, pues, finalmente, son quienes atienden los requerimientos e implementan las soluciones.

Por esta razón, se recomienda tener en cuenta algunos controles en sus desarrollos, para prevenir escenarios como el **cross-site scripting (XSS)**, las inyecciones de código, la ejecución de códigos maliciosos y el **cross-site request forgery**. Posteriormente, revisamos el proyecto de OWASP, que publica un top 10 de vulnerabilidades, y donde se presentan los riesgos más críticos de aplicaciones web, junto con la forma de prevenirlas.

Los controles implementados dentro de los desarrollos deben validar la entrada, el procesamiento y la salida de los datos, de forma que se cumpla con los niveles de confidencialidad, integridad y disponibilidad.

De esta forma, deben implementarse **controles de detección**, que ayudan a identificar los accesos no autorizados o entradas no válidas; **controles preventivos**, diseñados para evitar la entrada de datos no autorizados o no válidos, y **controles proactivos**, que permiten probar los componentes de forma continua, a medida que se desarrollan, mediante pruebas unitarias y de integración.

Los **controles de detección** aseguran que las operaciones se hagan con la exactitud requerida y, además, que las consultas que se realicen sobre bases de datos u otros sistemas sean consistentes. Operaciones como la combinación de archivos, la modificación de datos, la actualización de bases de datos o los mantenimientos de sistemas son operaciones sensibles que deberían tener este tipo de controles.

Los **controles preventivos** deben enfocarse en que la información ingresada sea precisa y, por lo tanto, en que aquello que se registre en el sistema sea adecuado. Dentro de estos controles, se pueden tener validaciones mediante el uso de expresiones regulares, comprobar la integridad de los campos, verificar que no exista duplicidad en los datos, establecer rangos de validación e incluso comprobar que los valores de los campos sean razonables de acuerdo con unos criterios predefinidos.

## 2.1 Controles proactivos

Independientemente de los controles implementados, resulta muy importante, además, definir un adecuado plan de pruebas que incluya, asimismo, las pruebas de aceptación por parte de los usuarios finales, donde se evalúa la funcionalidad de la aplicación.

Aparte de los usuarios finales, deberían considerarse recursos del equipo de desarrollo para realizar pruebas de funcionalidad del código y de integración.

Incluso si la aplicación es muy crítica para el negocio, se podría considerar la alternativa de que un equipo externo al equipo de desarrollo también completamente con otras pruebas a las ya realizadas por el equipo de desarrollo.

Para lograr un software seguro, los desarrolladores deben contar con el respaldo y la ayuda de la organización para la que escriben el código, cosa que, a veces, no sucede. A medida que los desarrolladores de software crean el código que compone una aplicación, se necesita adoptar y practicar una amplia variedad de técnicas de codificación segura. Todas las capas que componen una aplicación, tales como la interfaz de usuario, la lógica de negocio, el controlador o la base de datos, deben desarrollarse teniendo la seguridad en mente. La mayoría de los desarrolladores no han aprendido sobre la codificación segura o la criptografía en su etapa de formación y, en ocasiones, tampoco se lo han exigido.

Los lenguajes y frameworks que los desarrolladores usan para construir aplicaciones a menudo carecen de controles centrales críticos o son inseguros por defecto de alguna manera. También, en honor a la verdad, es muy raro que las organizaciones brinden a los desarrolladores requisitos prescriptivos que los guíen por el camino del software seguro e, incluso, cuando lo hacen, puede haber errores de seguridad inherentes a los requisitos y diseños. Por ello, se ha de intentar ser proactivo desde los inicios del diseño del proyecto que se realice, teniendo en cuenta, en principio, la seguridad.

## 2.2 OWASP (Open Web Application Security Project)

OWASP (Open Web Application Security Project) nos provee de una serie de recursos basados, sobre todo, en guías y herramientas, para que nuestros proyectos web sean lo más seguros posible, tanto desde el punto de vista de desarrollo seguro como de la evaluación de la seguridad de estos.

Para los desarrolladores y auditores de seguridad, ofrece una gran cantidad de recursos para ayudar a llevar a cabo un ciclo de vida de desarrollo de software seguro (S-SDLC, Secure Software Development Life Cycle), así como una variedad de recursos, como guías y herramientas, para evaluar la seguridad de las aplicaciones web.

Entre los principales objetivos de la OWASP, podemos destacar:

- Proporcionar a los desarrolladores un conjunto de buenas prácticas, para que las aplicaciones sean lo más seguras posible.

- Proporcionar a desarrolladores y profesionales de seguridad recursos para asegurar aplicaciones; en concreto, para aplicaciones móviles, surgió el proyecto OWASP Mobile Security Project.

El objetivo del proyecto es intentar ayudar a los desarrolladores a comprender qué, por qué, cuándo, dónde y cómo probar aplicaciones web. Los desarrolladores pueden usar este framework como plantilla para construir sus propios programas de prueba o para evaluar los procesos de otros desarrolladores. En la guía se describen en detalle tanto el framework de pruebas en general como las técnicas requeridas para llevarlo a la práctica.

El proyecto de seguridad en aplicaciones web (OWASP) es una comunidad abierta dedicada a facultar a las organizaciones a desarrollar, adquirir y mantener aplicaciones que pueden resultar confiables. Esta organización nos presenta los 10 riesgos más comunes, entre los que podemos destacar:

- **Inyección.** Los errores de inyección, tales como SQL y LDAP, ocurren cuando datos no confiables son enviados a un intérprete como parte de un comando o consulta. Los datos hostiles del atacante pueden engañar al intérprete a la hora de ejecutar comandos no intencionados o acceder a datos no autorizados.
- **Pérdida de autenticación y gestión de sesiones.** Las funciones de la aplicación relacionadas con la autenticación y gestión de sesiones son, frecuentemente, implementadas de manera incorrecta, lo que permite a los atacantes comprometer contraseñas, claves, *token* de sesiones o explotar otros errores de implementación para asumir la identidad de otros usuarios.
- Secuencia de comandos en sitios cruzados (XSS). **Los errores XSS ocurren** cada vez que una aplicación toma datos no confiables y los envía al navegador web sin una validación y codificación apropiada. XSS permite a los atacantes ejecutar una secuencia de comandos en el navegador de la víctima, los cuales pueden secuestrar las sesiones de usuario, destruir sitios web o dirigir al usuario hacia un sitio malicioso.
- **Referencia directa insegura a objetos.** Una referencia directa a objetos ocurre cuando un desarrollador expone una referencia a un objeto de implementación interno, tal como un fichero, directorio o base de datos. Sin un chequeo de control de acceso u otra protección, los atacantes pueden manipular estas referencias para acceder a datos no autorizados.

- **Configuración de seguridad incorrecta.** Una buena seguridad requiere tener definida e implementada una configuración segura para la aplicación, marcos de trabajo, servidor de aplicación, servidor web, base de datos y plataforma. Todas estas configuraciones deben ser definidas, implementadas y mantenidas ya que, por lo general, no son seguras por defecto. Esto incluye mantener todo el software actualizado, incluidas las librerías de código utilizadas por la aplicación.
- **Exposición de datos sensibles.** Muchas aplicaciones web no protegen adecuadamente datos sensibles, tales como números de tarjetas de crédito o credenciales de autenticación. Los atacantes pueden robar o modificar tales datos para llevar a cabo fraudes, robos de identidad u otros delitos. Los datos sensibles requieren de métodos de protección adicionales, tales como el cifrado de datos, así como también de precauciones especiales en un intercambio de datos con el navegador.
- **Ausencia de control de acceso a funciones.** La mayoría de las aplicaciones web verifican los derechos de acceso a nivel de función antes de hacer visible una funcionalidad en la interfaz de usuario. A pesar de esto, las aplicaciones necesitan verificar el control de acceso en el servidor cuando se accede a cada función. Si las solicitudes de acceso no se verifican, los atacantes podrán realizar peticiones sin la autorización apropiada.
- **Falsificación de peticiones en sitios cruzados (CSRF).** Un ataque CSRF obliga al navegador de una víctima autenticada a enviar una petición HTTP falsificada, incluyendo la sesión del usuario y cualquier otra información de autenticación, incluida automáticamente, a una aplicación web vulnerable. Esto permite al atacante forzar al navegador de la víctima para generar peticiones que la aplicación vulnerable piensa que son peticiones legítimas.
- **Utilización de componentes con vulnerabilidades conocidas.** Algunos componentes tales como las librerías, los frameworks y otros módulos de software casi siempre funcionan con todos los permisos y privilegios. Si se ataca un componente vulnerable, esto podría facilitar la intrusión en el servidor o una pérdida de datos. Las aplicaciones en las que se utilicen componentes con vulnerabilidades conocidas permiten ampliar el rango de posibles ataques.
- **Redirecciones y reenvíos no validados.** Las aplicaciones web, frecuentemente, redirigen y reenvían a los usuarios hacia otras páginas o

sitios web y utilizan datos no confiables para determinar la página de destino. Sin una validación apropiada, los atacantes pueden redirigir a las víctimas hacia sitios de phishing o malware, o utilizar reenvíos para acceder a páginas no autorizadas.

Un proyecto interesante que ofrece OWASP es la aplicación open source **Zed Attack Proxy Project (ZAP)** <https://www.zaproxy.org>, que nos permite realizar un análisis de todos los datos que se envían y que recibimos a la hora de realizar una navegación de un sitio web. La herramienta se puede descargar desde el repositorio de github: <https://github.com/zaproxy/zaproxy/wiki/Downloads>

La herramienta soporta análisis automático y manual. El primero es más sencillo y menos eficiente, ya que, si se requiere introducir login en la web, solo va a escanear hasta ese punto. Para utilizar el modo automático, bastaría con introducir la url que deseamos analizar en el apartado “url a atacar” y presionar el botón “Atacar”.



Figura 2.1 Modos de funcionamiento de OWASP ZAP.

En el modo manual, ZAP actúa como un proxy de intercepción de las peticiones que se realizan en el navegador, de forma que es capaz de analizar cada una de las url para extraer valores de formularios y parámetros de navegación del usuario. Utilizando esta opción, OWASP ZAP utiliza un proxy propio para capturar las peticiones que realizamos durante una sesión de navegación. Toda la navegación efectuada será monitorizada en la aplicación ZAP y se irá mostrando en el apartado “Sitios”. En el panel inferior se ven los resultados obtenidos en forma de alertas.



Figura 2.2 Alertas obtenidas al realizar un análisis sobre un sitio web.

### 2.3. OWASP Mobile Security Project

El objetivo del proyecto reside en crear conciencia acerca de la seguridad en aplicaciones, identificando algunos de los riesgos más críticos a los que se enfrentan las organizaciones. Pues bien, tanto los desarrolladores de aplicaciones móviles como los auditores pueden aprovecharse también de recursos de este tipo para alcanzar el objetivo de la máxima seguridad posible. Para ello, OWASP nos propone el proyecto Mobile Security Project.

La checklist de OWASP para dispositivos móviles está centrada en la seguridad referente a aplicaciones móviles. Trata de dar a los desarrolladores de aplicaciones un recurso centralizado para mantener las aplicaciones seguras. El objetivo es clasificar los riesgos de seguridad y proveer de controles para reducir el impacto o el riesgo de explotación. OWASP se focaliza en la capa de aplicación, en la aplicación entre la aplicación y los recursos remotos de servicios de autenticación, y en las características específicas de plataformas *cloud*.

El proyecto de seguridad móvil OWASP ofrece a los desarrolladores y equipos de seguridad los recursos para construir y mantener aplicaciones móviles seguras. También permite clasificar los riesgos de seguridad móvil y proporciona controles de desarrollo para reducir su impacto o probabilidad de explotación.

Tal y como podemos encontrar en el top 10 de riesgos en aplicaciones móviles [https://www.owasp.org/index.php/Mobile\\_Top\\_10\\_2016-Top\\_10](https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10), la siguiente lista nos indica el top 10 de los controles de seguridad para evaluar en aplicaciones móviles:

- **M1-Improper Platform Usage (uso indebido de plataformas).** Esta categoría cubre el mal uso de las características de las plataformas móviles o de los fallos de uso en los controles de seguridad. Puede incluir permisos en las plataformas, así como la mala utilización de TouchID,

del llavero u otros controles de seguridad que forman parte del sistema operativo.

- **M2-Insecure Data Storage (almacenamiento de datos inseguro).** Esta categoría cubre las antiguas categorías M2 y M4 del año 2014 (M2: Weak Server Side Controls; M4: Client Side Injection). Cubre el almacenamiento de datos inseguro y la fuga de datos no deseados.
- **M3-Insecure Communication (comunicaciones inseguras).** Esta categoría cubre un mal establecimiento de conexión o handshake, versiones SSL inseguras o comunicaciones en texto en claro sensibles. Toda comunicación donde se transmitan datos de usuarios debe realizarse mediante un canal seguro. Si la aplicación implementa sesiones de usuario mediante el uso de cookies, se deben proteger todas las conexiones contra el servidor.
- **M4-Insecure Authentication (autenticación insegura).** Esta categoría incluye nociones de autenticación del usuario final o una mala gestión de la sesión del usuario. Algunos ejemplos son no poder identificar al usuario, con lo que se pierde trazabilidad de las acciones, o no mantener la identidad del usuario cuando se requiere.
- **M5-Insufficient Cryptography (criptografía insuficiente).** Se hace uso de la criptografía en el código para proteger activos sensibles. Sin embargo, a veces, la criptografía no es suficiente en las aplicaciones, ya que se emplean algoritmos criptográficamente rotos, como MD5 o SHA1. En este punto resulta recomendable usar, al menos, un hash del tipo SHA256, que devuelve un bloque de 256 ante cualquier entrada. Al aumentar los bits de salida del algoritmo, en comparación con el algoritmo MD5, disminuye la posibilidad de encontrar colisiones; por lo tanto, es más seguro.
- **M6-Insecure Authorization (autorización insegura).** Se refiere a fallos en las autorizaciones en las aplicaciones. Es diferente a problemas de autenticación e identificación de usuarios. Cuando las aplicaciones autorizan, dan acceso a los usuarios a visualizar en la aplicación solo aquello que debe visualizar. Se ha de dotar de permisos al usuario, ya sean de escritura, lectura, ejecución u otros permisos más afines al negocio.
- **M7-Client Code Quality (calidad del código de cliente).** Aquí podríamos incluir problemas relacionados, como desbordamiento de búfer, vulnerabilidades de cadena de formato y otros varios errores a nivel de código, donde la solución estriba en volver a escribir algo de código, que se ejecuta en el dispositivo móvil.

- **M8-Code Tampering (manipulación de código).** Esta categoría incluye parchear binarios o modificar recursos locales y memoria dinámica. Un atacante puede directamente modificar el código, cambiar contenidos en la memoria dinámica o incluso reemplazar el sistema de API, que usa la aplicación, o bien cambiar los recursos y datos de la aplicación.
- **M9-Reverse Engineering (ingeniería inversa).** Esta categoría incluye el análisis de binarios para obtener el código fuente y las librerías. Herramientas como IDA Pro, Hooper y otras de inspección de binarios permitirán al atacante visionar acerca del funcionamiento interno de la aplicación.
- **M10-Extraneous Functionality (funcionalidad extraña).** En ocasiones, los desarrolladores incluyen funcionalidades ocultas de acceso mediante backdoors u otros controles de seguridad internos, no destinados al entorno de producción; por ejemplo, un desarrollador podría incluir, accidentalmente, una contraseña en un comentario de la aplicación o la posibilidad de desactivar el doble factor de autenticación durante la etapa de *testing*.

## 2.4 Controles proactivos OWASP

El proyecto OWASP Application Security Verification Standard (ASVS) proporciona una base para probar los controles técnicos de seguridad de aplicaciones web y también proporciona a los desarrolladores una lista de requisitos para un desarrollo seguro.

El estándar proporciona una base para probar los controles técnicos de seguridad de las aplicaciones, así como los controles técnicos de seguridad en el entorno, utilizados para proteger contra vulnerabilidades como **Cross-site scripting (XSS)** e **inyección de SQL**. Este estándar se puede utilizar para establecer un nivel de confianza en la seguridad de las aplicaciones web.

A continuación comentaremos los principales controles proactivos que OWASP recomienda y que deberían incluirse en cada proyecto de desarrollo de software. Entre la lista de los controles que se deben realizar, por orden de importancia, podemos destacar:

- **Verificación de la seguridad desde las primeras etapas de desarrollo**
- **Validación de las entradas del cliente**
- **Desbordamientos del búfer**
- **Gestión de sesiones**

- **Implementación de controles de acceso**
- **Implementación de controles de identidad y autenticación**
- **Autenticación por múltiples factores**
- **Manejo de errores y excepciones**

#### **2.4.1 Verificación de la seguridad desde las primeras etapas de desarrollo**

Las pruebas de seguridad deberían ser parte integral de la práctica de ingeniería de software de un desarrollador. En muchas organizaciones, esto no suele hacerse, ya que las pruebas de seguridad se realizan fuera de los ciclos de pruebas de desarrollo.

Por ello, es importante realizar comprobaciones de seguridad de manera temprana y lo más frecuentemente posible, ya sea mediante pruebas manuales o pruebas automatizadas. Obviamente, esto aumenta los plazos de entrega y lógicamente, por ello, ha de ser tenido en cuenta desde el inicio del diseño de la aplicación.

Como parte del desarrollo, debe plantearse la necesidad de implementar medidas de testing durante la fase de desarrollo. Para ello, OWASP tiene un proyecto de testing del código. Puede considerarse **OWASP ASVS (Application Security Verification Standard Project)** <https://github.com/OWASP/ASVS>, como una guía para definir los requisitos de seguridad y las pruebas.

#### **2.4.2 Validación de las entradas del cliente**

La debilidad más común en la seguridad de las aplicaciones web es la falta de validación adecuada de los datos que provienen del cliente o del entorno antes de utilizarlo. Se debe validar y filtrar cada input de la aplicación, tanto a nivel frontend como backend. Incluso si los datos provienen de su aplicación, son posibles su intercepción y manipulación. Sin la validación correcta de todas las entradas, podríamos exponer nuestra aplicación a ataques, como desbordamientos de búfer e inyección de SQL..

Esta debilidad conduce a casi todas las principales vulnerabilidades en las aplicaciones web, tales como scripting cross site, inyección de SQL, inyección de intérprete, ataques de sistemas de archivos y desbordamientos de búfer. En este punto cabe asegurar que la aplicación satisface los siguientes requisitos de gestión de sesiones de alto nivel:

- Todas las entradas están validadas para ser correctas y utilizables para el propósito previsto.
- Los datos de una entidad externa o cliente nunca deben ser confiables y han de tratarse en todos los casos.

La validación de las entradas se tiene que realizar tanto en el lado del cliente como en el lado del servidor; por ejemplo, la validación JavaScript puede alertar al usuario de que un campo, en particular, debe consistir en números, pero el servidor ha de validar que el campo realmente consiste en números. Por otro lado, no podemos olvidar que la validación de entrada no debe utilizarse como el método principal para prevenir vulnerabilidades del tipo XSS e inyección de SQL, pero contribuye significativamente a reducir su impacto, si se implementa correctamente.

Todos los parámetros que gestiona la aplicación han de ser validados, especialmente cuando son recibidos como datos de entrada por el usuario. De esta forma, **dicha validación se realizará tanto en la parte del cliente como a nivel de servidor**. Dentro de esta validación, se comprobará:

- Si los caracteres incluidos son correctos, no incluyen caracteres HTML o códigos SQL o JavaScript.
- Si el tipo del valor introducido es válido.
- Si su longitud se encuentra entre el valor mínimo y el máximo permitido.
- Si el rango está entre los límites permitidos.

Una gran mayoría de las vulnerabilidades de la aplicación web surgen por no validar correctamente la entrada del usuario. Los usuarios que utilizan una interfaz de usuario no necesariamente introducen esta "entrada" directamente. En el contexto de aplicaciones y servicios web, esto podría incluir los siguientes casos:

- Cabeceras HTTP
- Cookies
- Parámetros GET y POST (incluyendo campos ocultos)
- Subidas de archivos (incluyendo información, como el nombre del archivo)

Existen dos enfoques generales para realizar la validación de la sintaxis de entrada, comúnmente conocida como "lista negra" y "lista blanca":

- **La lista negra (o blacklist)** es un listado de valores no aceptables con el que se intenta comprobar que una entrada de usuario determinada no posee contenido “conocido como malicioso”. Esto es similar a cómo funcionará un programa antivirus, que comprueba si un archivo coincide exactamente con el contenido malicioso conocido y, si lo hace, lo rechaza.
- **La lista blanca (o whitelist)** es un listado de valores válidos con la que se intenta comprobar que una entrada de usuario determinada coincide con un conjunto de entradas “bien conocido”; por ejemplo, una aplicación web puede permitirle seleccionar una de entre una serie de opciones: la aplicación comprobará que una de ellas ha sido seleccionada y rechazará todas las demás entradas posibles. Debido a que la validación de los datos de entrada y salida son aspectos básicos de seguridad informática, las listas blancas representan una herramienta indispensable para los desarrolladores.

Desde el punto de vista de la seguridad, siempre es mejor utilizar listas blancas. Si utilizáramos una lista negra, deberíamos introducir en esta lista todos los valores no aceptables, que podrían ser cantidades innumerables en constante crecimiento. Sin embargo, siempre sabemos qué valores son aceptables y podemos enumerarlos rápidamente, sin tener que complementar el listado en el futuro; por ejemplo, si sabemos que nuestra variable ha de tener un identificador numérico, una lista blanca incluiría tan solo valores enteros, mientras que una lista negra tendrá que incluir todo lo que no sea un valor entero (por ejemplo, letras, caracteres especiales, espacios, etc.).

Los ataques que se evitarán utilizando una correcta validación de parámetros de entrada serán:

- Cross-site scripting
- Inyección de código SQL
- Denegación de servicio en la aplicación: para evitarla, se deberán validar los tipos, las longitudes y los rangos de los parámetros de entrada, de modo que no se produzcan excepciones en la aplicación.
- Desbordamientos de búfer: mediante la validación de las longitudes de los parámetros de entrada, se evitará la ejecución de código arbitrario en el sistema.

#### **2.4.3 Desbordamientos del búfer**

Los desbordamientos del búfer son métodos que se usan para explotar el software de forma remota mediante la inyección de código malicioso en la aplicación destino. La causa raíz de los problemas de desbordamiento de búfer radica en que los lenguajes de programación más utilizados, como C y C++, tienen funciones que son inseguras. El principal problema se debe a que no se comproban los límites de manejo de búfer, al intentar acceder a posiciones que se encuentran por encima de la zona asignada.

El comportamiento indeterminado de programas que han invadido un búfer hace que sea particularmente difícil de depurar. En el peor de los casos, un programa puede desbordar un búfer y no mostrar ningún efecto secundario adverso. Como resultado, los problemas de desbordamiento de búfer a menudo permanecen invisibles durante las pruebas. Lo más importante a la hora de analizar los desbordamientos del búfer reside en que cualquier dato que pasa a ser asignado cerca del búfer potencialmente puede ser modificado cuando se produzca el desbordamiento.

#### **2.4.4 Gestión de sesiones**

La sesión es uno de los componentes principales de cualquier aplicación basada en web, por el cual se controla y mantiene el estado de un usuario que interactúa con ella. En este punto, se ha de asegurar que la aplicación satisface los siguientes requisitos de gestión de sesiones de alto nivel:

- Las sesiones son únicas para cada individuo y no pueden ser adivinadas o compartidas.
- Las sesiones se invalidan cuando ya no son necesarias y se anulan durante los períodos de inactividad.

Las sesiones para los usuarios se mantienen, en la mayoría de las aplicaciones, a través de una cookie, que puede ser vulnerable. La mayoría de frameworks web ofrece administración de sesiones de una forma segura. También se debe asegurar que el tamaño de la cookie de sesión es suficientemente largo. Las cookies de sesión cortas o predecibles hacen posible que un atacante pueda predecir o realizar otros ataques contra la sesión.

#### **2.4.5 Implementación de controles de acceso**

El control de acceso supone el concepto de permitir el acceso a los recursos solo a quienes tienen permiso para utilizarlos. Debemos asegurarnos de que una aplicación verificada cumple los siguientes requisitos de alto nivel:

- Los usuarios que acceden a los recursos deben aportar credenciales válidas para hacerlo.
- Los usuarios están asociados con un conjunto bien definido de roles y privilegios.
- Los metadatos de roles y permisos están protegidos contra su manipulación.

#### **2.4.6 Implementación de controles de identidad y autenticación**

La autenticación se corresponde con el proceso de verificar que un individuo o una entidad es quien dice ser. La autenticación se realiza, comúnmente, enviando un nombre de usuario o ID y uno o más elementos de información privada, que solo un usuario determinado debe conocer.

La gestión de sesiones es un proceso mediante el cual un servidor mantiene el estado de una entidad que interactúa con ella. Esto resulta necesario para que un servidor recuerde cómo reaccionar a las solicitudes posteriores durante una transacción. Las sesiones se mantienen en el servidor mediante un identificador de sesión, que se puede pasar de un lado a otro entre el cliente y el servidor al transmitir y recibir solicitudes. Las sesiones deben ser únicas por usuario y, computacionalmente, imposibles de predecir.

Es importante que la aplicación cubra los siguientes requisitos de alto nivel:

- Capacidad de comprobar la identidad de los usuarios que intentan entrar en la aplicación.
- Posibilidad de garantizar que solo los usuarios autorizados sean capaces de autenticarse y que las credenciales del usuario se transporten por la red de forma segura.

En cuanto a los **protocolos de autenticación**, mientras que la autenticación se realice a través de una combinación usuario/contraseña y el uso de la autenticación de factores múltiples, esta opción se considera de las más seguras hoy día. Entre los principales protocolos de autenticación, los más extendidos actualmente son:

- **OAuth.** Open Authorization (OAuth) es un protocolo que permite a una aplicación autenticar a un usuario contra un servidor, sin requerir contraseñas o algún servidor externo que actúe como proveedor de identidad. Utiliza un token generado por el servidor, y ofrece un flujo de autorización para que un cliente, tal como una aplicación móvil, pueda llamar al servidor sobre el cual el usuario está utilizando el servicio. OAuth 2.0 se basa en HTTPS para la seguridad y, actualmente, es usado e implementado por las API de empresas como Facebook, Google, Twitter y Microsoft.
- **OpenId.** OpenId es un protocolo basado en HTTP que utiliza proveedores de identidad para validar que un usuario es quien dice ser. Constituye un protocolo que permite a un proveedor de servicios de identidad un camino para el inicio de sesión único (SSO, por las siglas en inglés de *single sign-on*). Esto permite a los usuarios reutilizar la misma identidad dada a un proveedor de identidad OpenId de confianza y ser el mismo usuario en múltiples sitios web, sin necesidad de proveer de la contraseña a ningún sitio de servicio web externo.

En algunos casos, un nombre de usuario y una contraseña no proporcionan suficiente seguridad para una aplicación móvil. Cuando se trata de datos o transacciones confidenciales, resulta importante implementar la autenticación de dos factores. Las opciones para la autenticación basada en dos factores incluyen:

- Contraseña adicional
- Código adicional proporcionado por SMS o correo electrónico
- Contraseña más un valor adicional conocido solo por el usuario
- Pregunta/respuesta de seguridad, establecida de antemano durante el registro del usuario

#### 2.4.7 Autenticación por múltiples factores

La autenticación por múltiples factores es el uso de más de un factor de autenticación para iniciar sesión o procesar una transacción, mediante:

- Algo que se conoce (detalles de la cuenta o contraseñas).
- Algo que se tiene (tokens o teléfonos móviles).
- Algo que se es (factores biométricos).

Los esquemas de autenticación como las contraseñas de un solo uso (OTP, por las siglas en inglés de *one time passwords*) implementadas utilizando un token físico (hardware) también pueden conformar un factor clave en la lucha contra ataques, tales como los ataques CSRF del lado del cliente.

#### 2.4.8 Manejo de errores y excepciones

El objetivo principal del manejo y registro de errores reside en proporcionar una reacción útil por parte del usuario, los administradores y los equipos de respuesta a incidentes. Debemos asegurarnos de que una aplicación verificada cumple los siguientes requisitos de alto nivel:

- No recopilar o registrar información confidencial si no se requiere específicamente.
- Asegurarnos de que toda la información registrada sea manejada de forma segura y protegida, según las leyes de protección y privacidad de los datos.
- Si los registros contienen datos privados o confidenciales, cuya definición varía de un país a otro, los registros se convierten en la información más sensible que posee la aplicación y, por lo tanto, resultan muy atractivos para los atacantes.
- Deshabilitar errores detallados. Los errores detallados pueden ser útiles en un entorno de desarrollo, pero, en un entorno de producción, pueden filtrar información sensible de la aplicación.

El control de excepciones permite gestionar los diferentes tipos de errores que puedan producirse, tanto en el sistema como en la aplicación. Una incorrecta gestión de las excepciones, involuntariamente, puede llegar a mostrar información propia de la aplicación, la cual no debería ser accesible. Un ejemplo de una incorrecta gestión de las excepciones se produce cuando no se consigue validar correctamente algún dato de entrada enviado por el usuario y la aplicación no sabe tratar dicha excepción. Esto provoca un error, el cual muestra demasiada información, como trazas de la pila, sentencias SQL erróneas u otra información de depuración.

Se recomienda manejar, de forma adecuada, todos los posibles errores de la aplicación y que, cuando suceda un error, se responda con un resultado específicamente diseñado que sea de ayuda al usuario, pero que no revele detalles internos innecesarios.

El control de excepciones deberá tener las siguientes características:

- Se deberán capturar y controlar todas las excepciones que se puedan producir en la aplicación, para evitar posibles fallos de disponibilidad, control de acceso o confidencialidad.
- Se deberá realizar un tratamiento de todos los códigos de retorno, para comprobar la causa de la excepción.
- Como norma general, las operaciones dentro de una aplicación que deberán ser capturadas son las siguientes: operaciones de entrada/salida, acceso a datos, operaciones con hilos y llamadas a terceros productos.

## 2.5 Ataques en aplicaciones web

Los problemas de seguridad han crecido rápidamente en los últimos años. En las aplicaciones desarrolladas en C/C++, el principal vector de ataque se corresponde con el de desbordamiento de búfer, debido a que los desarrolladores no tienen en cuenta los tamaños de los búferes de datos. Este problema sigue siendo bastante común hoy día en navegadores que utilizan C/C++ como lenguaje de programación.

En la actualidad, la mayoría de las aplicaciones que se desarrollan incluyen tecnologías web, donde tenemos otros vectores de ataque, como los de inyección de SQL o cross-site scripting (XSS). Estos ataques se aprovechan de una incorrecta validación de los datos de entrada. Otros tipos de ataques, como el de CSRF de falsificación de solicitudes entre sitios, atacan tanto el contexto de entrada como el de salida de una aplicación.

El problema del contexto de salida se crea por el hecho de que la salida es interpretada y procesada de manera diferente por distintos motores de renderizado por los navegadores, además de los múltiples motores de plantillas que podemos encontrar hoy día para diferentes framework webs desarrollados en Java, Python o Ruby. Conocer el contexto de salida y saber cómo se muestran los datos proporcionados por el usuario es, actualmente, un requisito esencial para la seguridad de las aplicaciones web, de móvil o de escritorio.

### 2.5.1 Vectores de ataque

Los vectores de ataque son las vías específicas que podrían hacer que un ataque se realice con éxito; por ejemplo, incluir un código de consulta MySQL dentro

de una solicitud HTTP con la posibilidad de que una inyección de SQL se realice con éxito.

Una aplicación segura debe conocer estas posibilidades y tener un código que las maneje adecuadamente. Existen muchas partes específicas de las aplicaciones web que pueden ser atacadas. Entre los **vectores de ataque de entrada**, podemos destacar:

- Encabezados HTTP
- Entradas de formulario a través del método POST
- Entradas de formulario ocultas a través del método POST
- Valores de parámetros de url a través de GET
- Solicitud de datos via AJAX con método GET/POST

Entre los **vectores de ataque de salida**, podemos destacar:

- Texto HTML proporcionado por el usuario
- Hipervínculos y enlaces proporcionados por el usuario
- Hipervínculos proporcionados por el usuario que se ejecutan a través de JavaScript

Vulnerabilidades clásicas como inyección de SQL o XSS siguen siendo hoy día vulnerabilidades top de OWASP y las principales amenazas al trabajar con nuevas tecnologías tanto en frontend como en backend. También han surgido nuevos ataques del tipo XSS que permiten obtener cookies y credenciales del usuario.

### 2.5.2 Cross-site scripting (XSS)

Una aplicación web es vulnerable a XSS cuando aquello que nosotros enviamos al servidor (un comentario, un cambio en un perfil, una búsqueda, etc.) se ve posteriormente mostrado en la página de respuesta. Básicamente, un ataque XSS funciona engañando al motor de plantillas HTML que esté usando la aplicación, para ejecutar código arbitrario cuando lo que debería hacer, simplemente, sería mostrar los datos.

Por ejemplo, cuando realizamos una búsqueda y se nos muestra un mensaje ("No se han encontrado resultados"), se está incluyendo dentro de la página el mismo texto que nosotros hemos introducido. Ahí es donde podemos empezar a sospechar que el sitio web se muestra vulnerable a este tipo de ataque.

Cuando se habla de ejecución remota de código, se debe considerar cómo se realiza la interacción con la página. Evidentemente, toda petición enviada al servidor va a ser procesada por este y, en función de cómo la interprete, será o no factible un ataque mediante XSS.

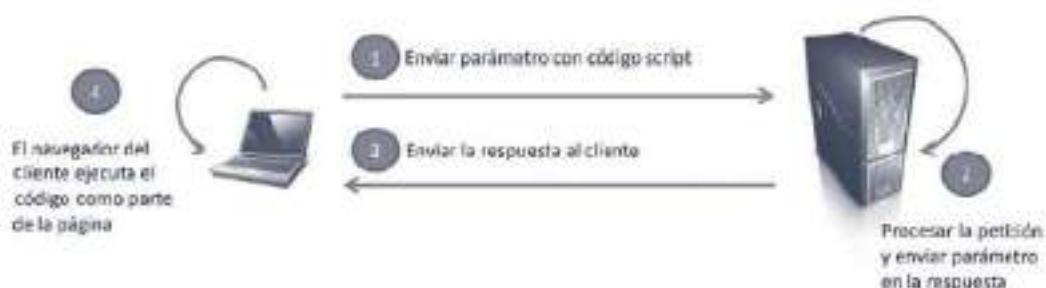


Figura 2.3 Ejemplo de ataque XSS.

Dentro de los posibles fallos de XSS, podemos distinguir dos grandes categorías:

- **No permanentes:** Esta categoría queda ilustrada con el caso que ahora se menciona. Nos encontramos con una página web que dispone de buscador, el cual, al introducir una palabra inventada o una cadena aleatoria de caracteres, muestra un mensaje del tipo: “No se han encontrado resultados para la búsqueda <texto>”, donde <texto> es la cadena introducida en el campo de búsqueda. Si en la búsqueda, se introduce como <texto> el código JavaScript antes indicado, y de nuevo aparece la ventana de alerta, significa que la aplicación es vulnerable a XSS. La diferencia radica en que, en esta ocasión, los efectos de la acción no resultan permanentes.
- **Permanentes:** Su denominación se debe al hecho de que, como mostraba el ejemplo, la ventana de alerta en JavaScript queda almacenada en algún lugar, habitualmente una base de datos SQL, y se muestra a cualquier usuario que visite nuestro perfil. Evidentemente, este tipo de fallos de XSS son mucho más peligrosos que los no permanentes.

Para inyectar el código, se localiza una zona de la página que, por su funcionalidad, incorpora dentro de su propio código HTML el código JavaScript, que anteriormente se ha introducido en algún lugar de la aplicación. Si este código se ha escrito en algún campo donde queda almacenado en una base de datos de forma permanente, se mostrará cada vez que un usuario acceda a la página. Sin embargo, las vulnerabilidades más habituales son las no permanentes. En estos casos, se debe

recurrir a la ingeniería social para efectuar el ataque. Para ello, resulta necesario fijarse, en primer lugar, en la url de la página, que deberá ser algo similar a:

```
http://www.dominio_victima.com/search?query=<script>alert("alert");</script>
```

Vamos a comentar a continuación las posibles implicaciones de seguridad que puede presentar un fallo de este tipo. Ha de considerarse que se trata únicamente de ideas generales y que el límite lo pone la imaginación del atacante y las funcionalidades de la aplicación objetivo del ataque:

- Un ataque de XSS puede tomar el control sobre el navegador del usuario afectado y, como tal, realizar acciones en la aplicación web. Si se ha logrado que un usuario administrador ejecute nuestro código JavaScript, las posibilidades de actuación maliciosa son muy superiores. Podrá, por citar algún ejemplo, desde borrar todas las noticias de una página a generar una cuenta de administrador con los datos que nosotros especifiquemos. Si el código JavaScript se ejecuta por un usuario sin derechos de administración, se podrá realizar cualquier modificación asociada al perfil específico del usuario en el sitio web.
- Igualmente, se pueden ejecutar ataques de defacement apoyándonos en técnicas que explotan vulnerabilidades XSS.
- Por otra parte, aunque menos habitual, es posible realizar un ataque de denegación de servicios distribuidos. Para ello, se puede forzar mediante código JavaScript a que los navegadores hagan un uso intensivo de recursos muy costosos en ancho de banda o capacidad de procesamiento de un servidor de forma asíncrona.

Como se puede comprobar, las posibilidades que ofrece el XSS son realmente amplias. Las únicas limitaciones se encuentran en la imposibilidad de ejecutar código fuera del navegador, dado que la sandbox sobre la que se ejecuta no permite el acceso a ficheros del sistema y a las propias funcionalidades que ofrezca el sitio web objeto del posible ataque.

### 2.5.3 Cross-site request forgery (CSRF)

CSRF (falsificación de solicitudes en sitios cruzados) se basa en valores de formulario conocidos o predecibles y en una sesión de navegador iniciada. Cada

envío de formulario debe contener un token, que se cargó con el formulario o al comienzo de una sesión de usuario. Verifique este token en el servidor cuando reciba POST para asegurarse de que el usuario lo originó. Esta capacidad se proporciona con las principales plataformas web y se puede implementar en formularios con un desarrollo personalizado mínimo. El diagrama de la figura 2.4 muestra en detalle el flujo de un ataque de este tipo.



Figura 2.4 Ejemplo de ataque CSRF.

Para mejorar la seguridad frente a un posible ataque, podemos hacer uso de las cabeceras Referer. Estas indican la página desde la que se ha llegado a otra. Se utilizan, por ejemplo, para conocer cuáles son las búsquedas que permiten a un usuario llegar a un sitio web desde un buscador. Una medida de protección, aunque se puede saltar, sería comprobar que las peticiones realizadas a nuestras páginas, o al menos a aquellas que impliquen acciones sensibles, se realicen desde nuestro propio dominio.

La única protección real frente a ataques de CSRF reside en establecer una serie de valores numéricos que se generen de manera única en cada petición. Estos pueden ser establecidos como un CAPTCHA, que el usuario debe introducir al realizar acciones críticas, o como un valor oculto, en un campo hidden, dentro del código de la página que comprobamos cuando el usuario nos devuelve la petición. Estas medidas, aunque tediosas de programar, nos permiten asegurar los datos de nuestros usuarios.

Como usuarios de aplicaciones posiblemente vulnerables a CSRF, podemos tomar una serie de precauciones que intenten evitar un ataque mediante esta técnica. Las medidas son:

- Cerrar la sesión inmediatamente tras el uso de una aplicación.
- No permitir que el navegador almacene las credenciales de ninguna página ni que ningún servidor mantenga nuestra sesión recordada más que durante el tiempo de uso.
- Utilizar navegadores distintos para las aplicaciones de ocio y aquellas que utilizamos para nuestro trabajo. Con esto nos aseguramos la independencia de las cookies de sesión entre navegadores.

#### 2.5.4 Seguridad en las redirecciones

Las redirecciones y los reenvíos no validados son posibles cuando una aplicación web acepta entradas no confiables que podrían hacer que la aplicación web redireccionase la petición a una url contenida en una entrada no confiable. Al modificar las entradas de url no confiables a un sitio malicioso, un atacante puede iniciar un ataque de phishing y robar las credenciales de los usuarios.

Los ataques de redirección y reenvío no validados también se pueden usar para crear maliciosamente una url que pasaría la verificación de control de acceso de la aplicación y, posteriormente, haría el reenvío a un atacante a funciones privilegiadas a las que normalmente no podrían acceder.

Los siguientes ejemplos demuestran códigos de redireccionamiento y reenvío inseguros. Si estamos programando con Java, podríamos realizar una redirección de la siguiente forma, donde recibimos la url a partir de un parámetro por GET o POST:

```
response.sendRedirect(request.getParameter("url"));
```

El siguiente código PHP obtiene una url a través del parámetro llamado url y luego redirige al usuario a esa url:

```
$ redirect_url = $_GET['url'];
header ("Location:". $ redirect_url);
```

Estos ejemplos se muestran vulnerables a un ataque si no se aplican controles de validación o métodos adicionales para verificar que el parámetro se trata de una url válida. Esta vulnerabilidad podría usarse como parte de una estafa de suplantación de identidad, al redirigir a los usuarios a un sitio malicioso. Si no se aplica una validación, un usuario malintencionado podría crear un hiper vínculo para redirigir a sus usuarios a un sitio web malicioso no validado; por ejemplo:

```
http://example.com/example.php?url=http://malicious.example.com
```

Para realizar estas redirecciones de forma segura, podríamos eliminar el hecho de que la url se pase por parámetro y añadirla directamente en el código:

```
response.sendRedirect ("http://www.mysite.com");
```

El uso seguro de redireccionamientos y reenvios se puede realizar de varias maneras:

- No permitir la url como entrada del usuario.
- Si permitimos la url como entrada del usuario, asegurarnos de que el valor proporcionado sea válido, apropiado para la aplicación y autorizado para el usuario.
- Crear una lista de url confiables a través de listas de hosts o una expresión regular.
- Forzar que todas las redirecciones pasen primero por una página para notificar a los usuarios que están saliendo de su sitio y hacer que hagan clic en un enlace para confirmar.

## 2.6 SQL Injection: parametrización de las consultas en bases de datos

SQL Injection es una de las vulnerabilidades web más peligrosas y representa una amenaza seria ya que, entre otras acciones, permite la ejecución de código malicioso del atacante al cambiar la estructura de la declaración SQL de una aplicación web de una manera tal que puede robar datos, modificarlos o, poten-

cialmente, facilitar la inyección de comandos al sistema operativo subyacente. Asimismo, debemos tener en cuenta además que las sentencias SQL se ejecutan con rol de administrador, con lo cual las consecuencias pueden llegar a ser desastrosas para las aplicaciones.

### 2.6.1 Introducción a SQL Injection

La inyección de SQL es un ataque dirigido a la base de datos de una aplicación, ya sea web, de móvil o de escritorio. Un ataque de inyección de SQL funciona al injectar código SQL en una sentencia que, en realidad, está cambiando su lógica original.

Mediante un ataque por inyección de SQL exitoso, se puede leer información sensible desde la base de datos, modificar la información (Insert/Update/Delete), ejecutar operaciones de administración sobre la base de datos (por ejemplo, parar la base de datos), recuperar el contenido de un determinado archivo presente sobre el sistema de archivos del DBMS y, en algunos casos, emitir comandos al sistema operativo. Los ataques por inyección de SQL son un tipo de ataque de inyección, en el cual los comandos SQL se insertan en la entrada de datos, con la finalidad de efectuar la ejecución de comandos SQL predefinidos.

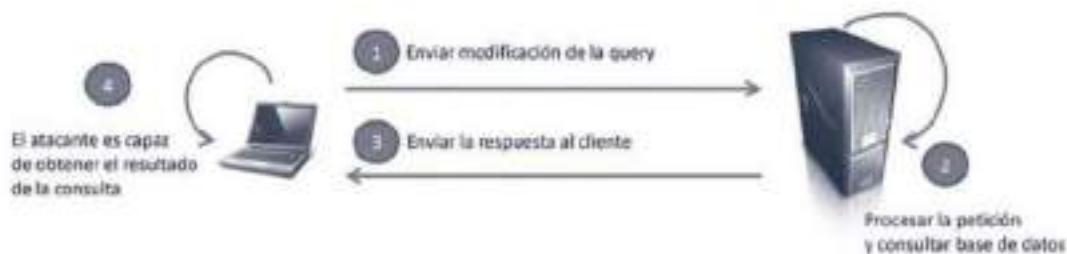


Figura 2.5 Ejemplo de ataque de inyección de SQL.

Además, esta vulnerabilidad no solamente pone en riesgo la integridad de la aplicación, sino de todos los datos almacenados de los usuarios que la utilicen, y se produce cuando no se filtra de forma correcta la información enviada por los usuarios.

Se realiza mediante la inserción de código SQL por medio de los datos de entrada: desde la parte del cliente hacia la aplicación, es decir, por medio de la inserción de este código, el atacante puede modificar las consultas originales

que debe realizar la aplicación y hacer ejecutar otras totalmente distintas, con la intención de acceder a las bases de datos, obtener información de alguna de las tablas o borrar los datos almacenados, entre otras muchas cosas. Como consecuencia directa de estos ataques y dependiendo de los privilegios que tenga el usuario de la base de datos bajo la que se ejecutan las consultas, se puede acceder no solo a las tablas relacionadas con la aplicación, sino también a otras tablas, pertenecientes a otras bases de datos alojadas en ese mismo servidor, con lo que el posible impacto se puede magnificar.

Todo lo comentado anteriormente es posible gracias al uso de determinados caracteres en los campos de entrada de información por parte del usuario, ya sea mediante el uso de los campos de los formularios enviados al servidor mediante POST o por medio de los datos enviados mediante GET en las url de las páginas web que posibilitan coordinar varias consultas SQL o ignorar el resto de la consulta, lo que permite al usuario malicioso ejecutar la consulta que elija; de ahí que sea de obligado cumplimiento realizar un filtrado de esos datos enviados, para evitar problemas en el futuro.

El atacante intentará provocar errores en la aplicación con el fin de extraer información que luego usará para insertar sus sentencias. Cuando una aplicación controla los mensajes de error y no ofrece ningún tipo de mensaje en la salida, se pueden realizar otros tipos de ataque más avanzados, como el de Blind SQL Injection, traducido como “ataque a ciegas por inyección de SQL”, que se produce cuando, en una página web, no aparece ningún mensaje de error al ejecutar una sentencia SQL errónea, por lo que el atacante va realizando pruebas hasta dar con el nombre de los campos o tablas sobre los que poder actuar.

### 2.6.2 Problemas que pueden causar este tipo de ataques

- **Integridad:** Al igual que un ataque por inyección de SQL permite leer información relevante almacenada en la base de datos, también es posible realizar cambios o incluso borrar toda la información mediante este tipo de vulnerabilidad. Se han dado casos de webs en las que se han dumpeado las bases de datos y, posteriormente, se han eliminado las que había para, a continuación, pedir rescates.
- **Confidencialidad:** Las bases de datos almacenan información sensible, por lo que la pérdida de confiabilidad representa un problema muy frecuente en aquellos sitios vulnerables a este tipo de ataques. Nadie vuelve a confiar en una plataforma a la que le han robado sus bases de datos.

- **Identificación:** Si el sistema de logueo que se utiliza para acceder a una zona restringida de una web es débil, por medio de este tipo de ataques se podría acceder sin la necesidad de conocer ni el nombre de usuario ni la contraseña. En nuestro entorno de trabajo, esto resultaría catastrófico en según qué circunstancias: se podrían crear usuarios que no existan anteriormente, con derechos de administrador y no conocidos, o podría entrar quien quisiera, desde donde quisiera, y hacer lo que deseara.

### 2.6.3 Ejemplo de inyección de SQL

A continuación, vemos un ejemplo de inyección de SQL donde tenemos un formulario en el que se pide un nombre de usuario y una contraseña para acceder a una zona privada. El código del lado del servidor que forma la consulta sería algo como lo siguiente:

```
$username = $_POST['username'];
$password = $_POST['password'];
$sql = " SELECT * FROM usuarios WHERE username = '$username' AND password =
'$password' ";
```

Como vemos, primero se almacenan el nombre de usuario y contraseña ingresados en las variables \$username y \$password, respectivamente; a continuación, se incluyen estos valores en la consulta que será enviada a la base de datos para comprobar si existe dicho usuario. Supongamos, en este caso, que el usuario ingresa como nombre de usuario “admin” y como contraseña “password123”. Al enviarse el formulario, la consulta formada será la siguiente:

```
SELECT * FROM usuarios WHERE username = 'admin' AND password = 'password123'
```

Como se ve, al colocarse los datos de entrada, estos quedan entre las comillas simples, para representar que se trata de una cadena de texto. Esta consulta será enviada a la base de datos y nos devolverá un resultado con los datos de dicho usuario, si existe, y se permitirá el acceso a la zona privada; de lo contrario, nos devolverá un resultado vacío y se impedirá negando el acceso.

Como se comentó anteriormente, SQL Injection consiste en anexar código SQL a una consulta, y este formulario lo permite mediante los campos de entrada, de forma que tenemos una aplicación vulnerable a SQL Injection.

El objeto de explotar esta vulnerabilidad reside en conseguir acceso a la zona privada sin conocer ni el usuario ni la contraseña correcta y aprovechar la vulnerabilidad, de forma que lo que tenemos que lograr es inyectar código SQL para formar una consulta que devuelva un resultado válido.

Veamos cómo queda formada la consulta si inyectamos el siguiente código SQL en el campo de contraseña:

Usuario:

Password:

Figura 2.6 Ejemplo de formulario de login con inyección de SQL en el campo de password.

Al formarse la consulta, quedará de la siguiente manera:

```
SELECT * FROM usuarios WHERE username = 'hacker' AND password = '' or 1=1#'
```

Se debe prestar atención a que el código insertado se encuentra después de las comillas simples en el campo de password. La comilla simple al inicio del código insertado se encarga de completar la comilla abierta en la parte de la consulta password = ''; de esta forma, obtenemos temporalmente la siguiente consulta:

```
SELECT * FROM usuarios WHERE username = 'hacker' AND password = ''
```

Esta consulta, de momento, no devolverá resultados, pues no existe tal usuario con esas credenciales. Sin embargo, vamos a analizar el resto del código insertado: `or 1=1#`.

La sentencia `or 1=1` cambia radicalmente la lógica de la consulta, ya que, como sabemos, en una consulta formada por el condicional OR, devolverá verdadero al cumplirse, al menos, una de las dos expresiones; en nuestro caso, la primera expresión es `username = 'hacker' AND password = ''`, y la segunda `or 1=1`. Esta última se cumple siempre, es decir, 1 es siempre igual a 1, porque la consulta nos devolverá un resultado válido.

Por último, tan solo queda deshacernos de la comilla que cierra la sentencia. Para esto podemos usar los comentarios utilizados en SQL: `#`, `--` (doble guion) o bien `/* */`. De esta forma, la consulta completa es:

```
SELECT * FROM usuarios WHERE username = 'hacker' AND password = '' or 1=1#'
```

A la hora de desarrollar una aplicación, resulta muy complicado crear una herramienta totalmente segura a las primeras de cambio y, si además se "hereda" de otros desarrolladores anteriores, aún más: a esto pueden añadirse versiones antiguas de software o framework que no dispongan de soporte. La falta de tiempo, con las consiguientes presiones y la intervención de varios programadores para su desarrollo, son factores que también juegan en contra de la seguridad. A pesar de tales inconvenientes, siempre se pueden tomar medidas de seguridad que nos ayuden a desarrollar aplicaciones más robustas; ajenas, en la medida de lo posible, a este tipo de problemas.

A continuación vamos a repasar algunos consejos para evitar sufrir el ataque por inyección de código SQL en nuestros desarrollos o aplicaciones en producción.

#### 2.6.4 Escapar caracteres especiales utilizados en las consultas SQL

Al hablar de "escapar caracteres", estamos haciendo referencia a añadir la barra invertida "`\`" delante de las cadenas utilizadas en las consultas SQL, con el fin de evitar que estas corrompan la consulta. Algunos de los caracteres especiales que es aconsejable escapar son las comillas dobles (`"`), las comillas simples (`'`) o los caracteres `\x00` o `\x1a`, ya que se consideran peligrosos, pues pueden ser utilizados durante los ataques.

Los distintos lenguajes de programación ofrecen mecanismos para lograr escapar estos caracteres. En el caso de PHP, podemos optar por la función que toma como parámetro una cadena y la modifica, para evitar así todos los caracteres especiales y devolver dicha cadena de forma segura para ser ejecutada

dentro de la instrucción SQL. En Java, la clase **StringEscapeUtils**, del proyecto Apache Commons, puede ayudar bastante, en según qué circunstancias, a escapar de según qué caracteres, lo que evita ataques de inyección de SQL o XSS cross-site scripting, por poner varios ejemplos.

#### 2.6.5 Delimitación de los valores de las consultas

Aunque el valor de la consulta sea un entero, se aconseja delimitarlo siempre entre comillas simples, como una instrucción SQL del tipo:

```
SELECT nombre FROM usuarios WHERE id_user = $id
```

Desde el punto de vista de la seguridad, resulta mejor delimitar el parámetro entre comillas simples:

```
SELECT nombre FROM usuarios WHERE id_user = '$id'
```

Si, en una consulta, estamos a la espera de recibir un entero, no confiemos en que sea así, sino que se recomienda tomar medidas de seguridad y realizar la comprobación de que realmente se trata del tipo de dato que estamos esperando. Para realizarlo, los lenguajes de programación ofrecen funciones que realizan esta acción, como pueden ser **ctype\_digit()**, para saber si es un número, o **ctype\_alpha()**, para saber si se trata de una cadena de texto, en el caso del lenguaje PHP.

También se debe comprobar la longitud de los datos, con el fin de descartar posibles técnicas de inyección de SQL. Si, por ejemplo, estamos esperando un nombre o una cadena larga, podría suponer que estén intentando atacarnos por este método. En el caso de PHP, podemos utilizar la función **strlen()**, para ver el tamaño de la cadena.

#### 2.6.6 Uso de sentencias preparadas parametrizadas

Las consultas SQL parametrizadas se construyen utilizando secuencias de SQL regulares, pero, en el momento de incluir datos suministrados por el usuario, incluyen parámetros de enlace, que son marcadores de posición para los datos, insertados posteriormente. En otras palabras, los parámetros bind permiten al

programador especificar explícitamente a la base de datos lo que debe ser tratado como un comando y lo que debe ser tratado como datos.

En la mayoría de los frameworks de desarrollo (Rails, Django, etc.), se emplea un modelo objeto-relacional (ORM) para la comunicación abstracta con una base de datos. Muchos ORM proporcionan parametrización automática de consultas cuando se utilizan métodos programáticos para recuperar y modificar datos, pero esto también implica que los desarrolladores deben tener cuidado al permitir la entrada de usuarios en consultas de objetos u otras consultas avanzadas soportadas por el framework en cuestión.

Con esta técnica, la consulta SQL se envía por separado de cualquier parámetro malicioso que pudiera suponer un peligro para la aplicación. Por último, si es posible, los motores de base de datos deben configurarse con el fin de que solamente tengan soporte para consultas parametrizadas; por eso, son tan importantes el diseño y el planteamiento inicial de las aplicaciones teniendo en cuenta la seguridad.

En el caso de Java y la utilización de un driver JDBC para conectarnos con la base de datos, podríamos parametrizar las consultas con el método `PreparedStatement` <https://docs.oracle.com/javase/8/docs/api/java/sql/PreparedStatement.html>

```
// Consulta Preparada.

import java.sql.*;

String custname = request.getParameter("customerName");

String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";

PreparedStatement pstmt = connection.prepareStatement(query);

pstmt.setString(1,custname);

ResultSet results = pstmt.executeQuery();
```

En el caso de C#, podríamos parametrizar las consultas con la clase `OleDbCommand` <https://docs.microsoft.com/es-es/dotnet/api/system.data.oledb.oledbcommand>

```
String query = "SELECT account_balance FROM user_data WHERE user_name = ?";  
  
try {  
  
    OleDbCommand command = new OleDbCommand(query, connection);  
  
    command.Parameters.Add(new OleDbParameter("customerName", CustomerName  
Name.Text));  
  
    OleDbDataReader reader = command.ExecuteReader();  
  
} catch (OleDbException se) {  
  
    // error handling  
  
}
```

En el caso de PHP, podríamos utilizar **PHP Data Objects** [https://es.wikipedia.org/wiki/PHP\\_Data\\_Objects](https://es.wikipedia.org/wiki/PHP_Data_Objects) para parametrizar las consultas inyectando los parámetros con el método `bindParam`.

```
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (:name, :value)");  
  
$stmt->bindParam(':name', $name);  
  
$stmt->bindParam(':value', $value);
```

## 2.6.7 Uso de procedimientos almacenados

El SQL que escribe en su aplicación web no es el único lugar donde se pueden introducir vulnerabilidades de inyección de SQL. Si está utilizando procedimientos almacenados y está construyendo dinámicamente SQL dentro de ellos, también puede introducir vulnerabilidades de inyección de SQL. Para garantizar que este SQL dinámico sea seguro, también puede parametrizar este SQL dinámico utilizando variables de enlace.

Los procedimientos almacenados no incluyen ninguna generación dinámica de SQL, ya que se encuentran almacenados en la base de datos. Estos son algunos ejemplos del uso de variables de enlace en procedimientos almacenados en diferentes bases de datos.

En el caso de **Oracle – PL/SQL**, no se está creando ningún SQL dinámico. Los parámetros que se pasan a los procedimientos almacenados están vinculados a su ubicación dentro de la consulta.

```
PROCEDURE SafeGetBalanceQuery (UserID varchar, Dept varchar) AS BEGIN  
SELECT balance FROM accounts_table WHERE user_ID = UserID AND department = Dept;  
END;
```

También tenemos la posibilidad de utilizar procedimientos almacenados empleando variables de enlace o temporales en la sentencia SQL para ejecutar con la sentencia “EXECUTE”. Las variables de enlace `stmt` y `result` se utilizan para indicar a la base de datos que las entradas a este SQL dinámico son “datos”.

```
PROCEDURE SafeGetBalanceQuery( UserID varchar, Dept varchar) AS  
stmt VARCHAR(400); result NUMBER;  
BEGIN  
stmt := 'SELECT balance FROM accounts_table WHERE user_ID = :1  
AND department = :2';  
EXECUTE IMMEDIATE stmt INTO result USING UserID, Dept;  
RETURN result;  
END;
```

Si estamos utilizando **SQL Server – Transact-SQL**, los parámetros que se pasan a los procedimientos almacenados están naturalmente vinculados a su ubicación dentro de la consulta.

```
PROCEDURE SafeGetBalanceQuery(@UserID varchar(20), @Dept varchar(10)) AS BEGIN  
SELECT balance FROM accounts_table WHERE user_ID = @UserID AND department = @  
Dept  
END
```

También podríamos utilizar **variables temporales**, donde las variables de vinculación se utilizan para indicar a la base de datos que las entradas a este SQL dinámico son “datos” y no código que se ejecute.

```
PROCEDURE SafeGetBalanceQuery(@UserID varchar(20), @Dept varchar(10)) AS BEGIN  
DECLARE @sql VARCHAR(200)  
  
SELECT @sql = 'SELECT balance FROM accounts_table WHERE ' + 'user_ID = @UID AND de-  
partment = @DPT' EXEC sp_executesql @sql, '@UID VARCHAR(20), @DPT VARCHAR(10)',  
@UID=@UserID, @DPT=@Dept  
  
END
```

## 2.7 Seguridad en AJAX

AJAX, Asynchronous JavaScript and XML, es una técnica que permite crear aplicaciones interactivas ricas, las cuales se ejecutan en el lado del cliente en el navegador. El funcionamiento de AJAX es sencillo: mientras la aplicación se ejecuta en el navegador del cliente, la comunicación se lleva a cabo de manera asíncrona en segundo plano con el servidor. Esto permite crear el efecto de que el sitio web va cambiando en función de las necesidades, por la actualización de la información en el servidor, y podría ser debido a otras acciones del cliente.

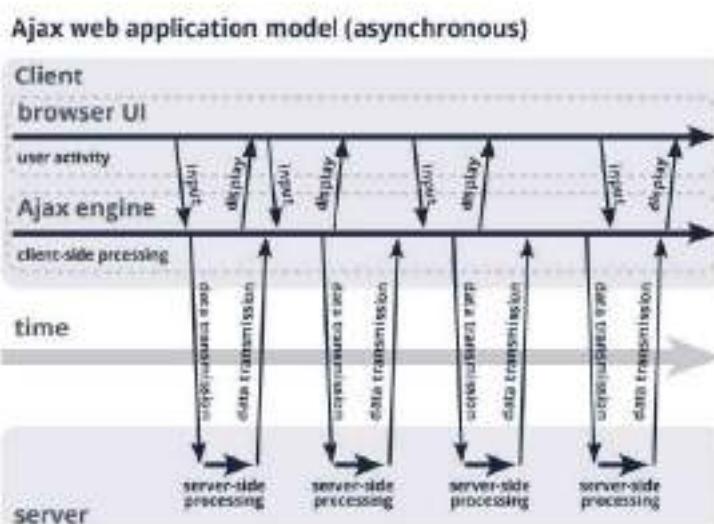


Figura 2.7 Modelo asíncrono de peticiones con AJAX.

AJAX es una tecnología cliente (se ejecuta en el navegador del usuario) que nos permite, mediante JavaScript, tener una comunicación asíncrona (mediante el protocolo HTTP) en segundo plano con el servidor. Las **ventajas** que encontramos al utilizar la tecnología AJAX en nuestras páginas web son las siguientes:

- Minimización del tráfico: Ya no es necesario cargar toda la información cada vez que necesitemos datos nuevos; podemos realizar peticiones en segundo plano con solo las partes que nos interesen.
- Mejora en las interfaces de usuario: Gracias a la evolución del DHTML y CSS, unido a la tecnología AJAX, podemos realizar webs que tengan un comportamiento más amigable e intuitivo, no solo limitarnos a la indexación de contenidos.

Existen algunos **inconvenientes** y limitaciones a la hora de usar AJAX; por ejemplo:

- Se requiere un navegador que tenga implementado el objeto XMLHttpRequest.
- La mayoría de los navegadores limitan a dos el número de conexiones simultáneas.
- Por seguridad, las peticiones que hagamos deben realizarse sobre un servidor local.
- Al no haber recarga de página, se pierde la funcionalidad de los botones “siguiente” y “atrás” de los navegadores.
- Al no cambiar la url tras una petición AJAX, no podremos acceder a toda la información de una página desde la barra de direcciones. Por esta misma razón, los buscadores no serán capaces de indexar todo el contenido de la web.

En la figura 2.8 vemos el esquema de una llamada AJAX.

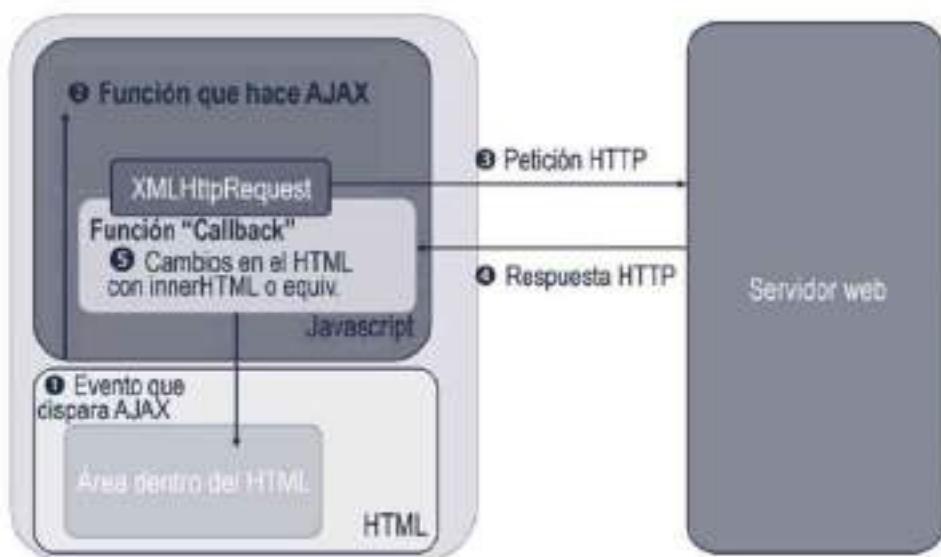


Figura 2.8. Esquema de una petición AJAX

1. **Nuestro componente HTML genera un evento** que disparará, posteriormente, una llamada. La gestión de los eventos se hace con el estándar W3C de event listeners, sobre el objeto XMLHttpRequest.
2. Se llama a la función que realizará el tratamiento de la petición de forma asíncrona.
3. **Se crea el objeto XMLHttpRequest** para realizar la petición.
4. **Se obtiene la respuesta.**
5. **Se procesa la respuesta.** La propiedad estatus del objeto XMLHttpRequest contiene el código de estado del servidor (en HTTP, OK es el 200). Utilizamos la función de callback para procesar la respuesta del servidor que tenemos en la propiedad responseText.

El objeto **XMLHttpRequest** se declara de forma diferente dependiendo del navegador, pero sus funciones son las mismas y permite que la interacción del usuario con la aplicación suceda asincrónicamente (con independencia de la comunicación con el servidor).



Figura 2.9 Objeto XMLHttpRequest.

Supongamos que, en el lado de servidor, tenemos un script en php que espera un parámetro “id” con un valor numérico que tenemos en la variable JavaScript “identificador”. De esta forma, el servidor nos avisará cuando haya terminado el script, y llamará a la función que le designemos como función callback.

```

var req = new XMLHttpRequest();
//preparar la petición. El tercer parámetro indica que si es asíncrona
req.open('GET','http://www.server.com/script.php?id=' + identificador, true);
//decir qué función hace de "callback"
req.onreadystatechange = callbackFunction;
req.send(null);
// función de callback
function callbackFunction() {
if (req.readyState == 4) { //también valdría this.readyState
if(req.status == 200)
alert(req.responseText);
}
}
    
```

En el ejemplo anterior, utilizamos la propiedad **onreadystatechange** del objeto XMLHttpRequest para indicarle cuál es la función de callback. El servidor

nos informa en la propiedad **readyState** de XMLHttpRequest de si la respuesta ha empezado a llegar (**readyState==2**), está cargándose (**readyState==3**) o ha sido completada (**readyState==4**). La propiedad **responseText** contiene la información que nos envía el servidor en una cadena de texto.

El siguiente ejemplo permite la subida de ficheros al servidor al mismo tiempo que visualmente el usuario puede ver el avance:

```
<script>

function verProgreso(e) {
    var progreso= document.getElementById("progreso");
    progreso.innerHTML = Math.round((e.loaded / e.total)*100)+"%";
}

function uploadAJAX() {
    var fdata = new FormData(document.getElementById("form"))
    var xhr = new XMLHttpRequest();
    xhr.upload.addEventListener("progress", verProgreso, false)
    xhr.open("POST","http://server.com/upload", true)
    xhr.onreadystatechange = function() {
        if (this.readyState==4)
            alert(this.responseText)}
        xhr.send(fdata)
    }
}

</script>

<form enctype="multipart/form-data" id="form">
    Elegir archivo: <input type="file" name="archivo"/> <br/>
    <input type="button" value="enviar" onclick="uploadAJAX()"/>
</form>

<div id="progreso"></div>
```

Con respecto a la seguridad, hay que tener en cuenta que la seguridad de la aplicación dependerá de la seguridad que tienen los componentes que la forman.

Se debe considerar también el aumento de la superficie de ataque, es decir, la parte que se ejecuta o desarrolla en la parte del cliente, y esto hace que la exposición o superficie de ataque sea mayor. La revelación de la lógica de la aplicación hace que los posibles atacantes conozcan parte del código, ya que este reside en la parte del cliente. Ello lleva a que un posible atacante pueda estudiar cierta parte de la lógica y utilizarla para llevar a cabo acciones maliciosas sobre la lógica de la aplicación.

Un problema que tenemos utilizando AJAX entre nombres de dominios es lo que se conoce como **Same Origin Policy**. Con este problema de seguridad, un script obtenido en un origen puede cargar o modificar propiedades del documento desde otro origen distinto al primero.

Con la política de seguridad del “mismo origen”, un objeto XMLHttpRequest solo puede realizar una petición AJAX al mismo host del que vino la página en la que está definido. El código JavaScript puede provenir de otros dominios, como [www.myapi.com](http://www.myapi.com), pero, como la página se carga desde [www.myapp.com](http://www.myapp.com), el código se ejecuta en ese contexto y se restringe a ese dominio.

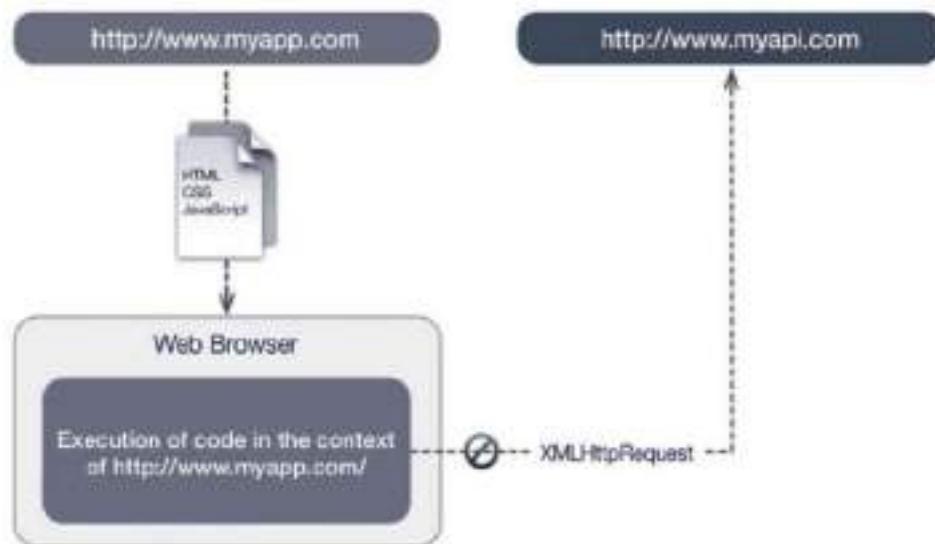


Figura 2.10 Política del mismo origen en XMLHttpRequest.

El “cross-domain AJAX” (CORS) permite romper esta política bajo determinadas circunstancias. Si el servidor al que se realiza la petición permite la cabecera **Access-Control-Allow-Origin**, el navegador también dejará que se realice:

```
HTTP/1.1 200 OK
Server: Apache/2.0.61
Access-Control-Allow-Origin: *
```

El tratamiento de ficheros XML conforma otro de los problemas de seguridad a los que nos podemos enfrentar. El servidor debe validar todos los datos que recibe, ya que un posible XML con un formato incorrecto puede causar un error en el servidor, lo que provocaría una denegación de servicio.

La ejecución de código malicioso también se encuentra en este ámbito, y es importante tenerlo en cuenta. Las llamadas que se realizan con AJAX se ejecutan en background, sin ninguna interacción del usuario, por lo que este no es consciente de lo que se está realizando en un sitio concreto, de modo que la web puede aprovechar este hecho para proceder a un robo de cookies. Constituye un problema de seguridad que se debe tener muy en cuenta, ya que se puede llevar a cabo de manera silenciosa y poco sospechosa.

Algo que se debe considerar igualmente son las validaciones de cliente y no realizadas en la parte del servidor. Este hecho es quizás una de las mayores amenazas a las que se enfrenta AJAX y cualquier tecnología que se encuentre en el lado del cliente. Toda petición ha de ser validada en el lado del servidor, independientemente de si se hace una validación en el cliente.

Para solucionar tales problemas, se proponen las siguientes **prácticas con JavaScript**:

- El código de JavaScript se debe ejecutar mediante el uso de una *sandbox*, con lo que el código malicioso que se intente ejecutar queda aislado y sin acceso a los recursos de interés de la máquina atacada.
- No se deben establecer conexiones con sitios diferentes del nombre de dominio del que se ha obtenido el script de JavaScript. Esta medida se denomina **CORS, Cross-Origin Resource Sharing**. El intercambio de recursos de origen cruzado es un estándar del W3C para especificar, de manera flexible, qué peticiones están permitidas entre dominios.
- Todos los campos de entrada y comprobación siempre se validarán en el lado del servidor. De otra manera, cualquier acción que se valide en el cliente puede ser manipulada; por ejemplo, mediante el uso de un proxy.

- Se han de implantar controles de seguridad adicional en el servidor, como pueden ser autenticación, autorización e incluso registro y log de operaciones.
- Se utilizará el método POST en lugar del método GET.
- Se deben tener en cuenta los ataques clásicos, como inyección de SQL, XSS, CSRF, los cuales podrán ser solventados mediante un filtrado correcto o la utilización de tokens correctamente, en el caso del CSRF.
- No se pueden almacenar datos sensibles o confidenciales en el lado del cliente. Este hecho haría que obtenerlos por un atacante fuera potencialmente sencillo.
- Se han de utilizar métodos criptográficos para transmitir datos sensibles o confidenciales entre el cliente y el servidor.
- La lógica de negocio debe seguir el principio de mínima exposición y encontrarse siempre en el lado del servidor.
- Toda petición realizada con AJAX y que acceda a recursos protegidos ha de encontrarse autenticada.

## Capítulo 3. Herramientas OWASP

OWASP nos propone un conjunto de herramientas que nos ayudarán tanto para el desarrollo seguro como para testear los distintos controles de seguridad. La seguridad es un factor que hay que tener en cuenta en cada fase del desarrollo, ya que el desarrollo de software constituye un proceso en el que es necesario ir identificando y corrigiendo vulnerabilidades de forma continua:

- **Checkmarx** <https://www.checkmarx.com>, una herramienta de análisis de seguridad del código fuente, integrable con los entornos de desarrollo más comunes, como Eclipse, MS-Visual Studio, Jira, Jenkins..., que ofrece alertas a los desarrolladores sobre las vulnerabilidades que se produzcan a nivel de código.
- **BlackDuck** <https://www.blackducksoftware.com>, una herramienta de análisis de la seguridad e integrable con los entornos de desarrollo y despliegues más usados del mercado.
- **DefectDojo** <https://www.defectdojo.org>, una herramienta open source de gestión de vulnerabilidades construida para análisis DevOps, integración y entrega continua y seguridad.
- **SonarQube** <https://www.sonarqube.org>, plataforma open source para análisis estático.

### 3.1 DefectDojo

DefectDojo <https://www.defectdojo.org/> es una herramienta de gestión de vulnerabilidades que permite administrar la seguridad de su aplicación o realizar análisis, así como detectar y evaluar vulnerabilidades.

Entre las principales características, podemos destacar:

- Gestión y detección de vulnerabilidades en varios formatos
- Integración con jira
- Automatización y seguimiento de pruebas de seguridad mediante integración continua
- Visualización y seguimiento de métricas

Al ser una herramienta open source, disponemos del código fuente en el repositorio de github <https://github.com/DefectDojo/django-DefectDojo>. Se encuentra desarrollada en python con **Django web framework**. También tenemos disponible una imagen de Docker en el repositorio DockerHub <https://hub.docker.com/r/aweaver/django-defectdojo>

Para su instalación con Docker, bastaría tener Docker instalado y ejecutar el comando para descargarnos la imagen y, posteriormente, ejecutar el contenedor con ella:

```
docker pull aweaver/django-defectdojo
```

Para el caso de que vayamos a instalarlo en local, se recomienda leer la guía y la documentación en <https://defectdojo.readthedocs.io>. La aplicación también la podemos probar directamente en la url <https://defectdojo.herokuapp.com>, donde podemos entrar con las credenciales de usuario y contraseña que aparecen en el repositorio de github.

Podemos visualizar las aplicaciones que estamos analizando junto con un resumen a nivel de vulnerabilidades encontradas y nivel de criticidad.

Product List						
Product #	Criticality	Mitigate	Findings	Environment	Context	Product Type #
1 Applic Accounting Software	★★★★	● ● ● ●	0	2	User 2; Technical	Billing
1 Budgeter	★★★★	● ● ● ●	229	4	Tester, Jester, Manager Bob, Bruce, Technical (User 1), Technical	Commerce
1 internal CRM App	★★★★	● ● ● ●	0	0	Bob, Bruce, Manager Tester, Jester, Technical (product_manager), Technical	Commerce
1 ownCloud	★★★★	● ● ● ●	0	2	T. Manager	Billing
1 TechProcket	★★★★	● ● ● ●	0	0	Anthony, Manager Gurkha, Technical	Research and Development
1 todo			0	0		Commerce

Figura 3.1 Visualizando aplicaciones [Product List].

Si vemos en detalle una determinada aplicación, de forma visual ofrece una serie de **métricas**, con las cuales podemos ver el estado general de la aplicación.

Figura 3.2 Detalle de una aplicación.



Figura 3.3 Visualizando métricas.



Figura 3.4 Visualizando detalles de las métricas.

Los **EndPoints** representan aquellas direcciones IP o nombres de dominio para comprobar la seguridad.

All Endpoints		
Endpoint #	Product #	Open Findings
192.168.1.1	Appx Accounting Software	No Open, Active Findings
http://localhost/	Budget	0
http://localhost:8080	Budget	0
http://localhost:8080/budget/index.jsp	Budget	0
http://localhost:8080/budget/about.jsp	Budget	0
http://localhost:8080/budget/contact.jsp	Budget	0
http://localhost:8080/budget/advanced.jsp	Budget	0
http://localhost:8080/budget/logout.jsp	Budget	0
http://localhost:8080/budget/login.jsp	Budget	0

Figura 3.5 Visualizando EndPoints.

Podemos visualizar aquellos **servicios vulnerables** para un determinadoEndPoint.

All Endpoints		
Endpoint #	Product #	Open Findings
192.168.1.1	Appx Accounting Software	No Open, Active Findings
http://localhost/	Budget	0
http://localhost:8080	Budget	0
http://localhost:8080/budget/index.jsp	Budget	0
http://localhost:8080/budget/about.jsp	Budget	0
http://localhost:8080/budget/contact.jsp	Budget	0
http://localhost:8080/budget/advanced.jsp	Budget	0
http://localhost:8080/budget/logout.jsp	Budget	0
http://localhost:8080/budget/login.jsp	Budget	0

Figura 3.6 Visualizando servicios vulnerables para un determinadoEndPoint.

Los **findings** (hallazgos) representan una vulnerabilidad en la aplicación que se está probando.

Severity	Name	CWE	Date Found	Age	SLA	Reporter	Found By	Status	Product
High	High Impact Buffer	699	Jan 1, 2018	464	AAA	admin	API Test	Active, Verified	Internal CHM App
Medium	Cross Site Scripting	675	April 11, 2018	9	BB	admin	API Test	Active, Verified	TestProduct

Figura 3.7 Visualizando hallazgos para una determinada aplicación.

Si entramos en el detalle de una vulnerabilidad, podemos ver el nivel de criticidad, una descripción, así como el impacto y referencias de las vulnerabilidades en otras bases de datos como **CVE**, **NVD**, **SecurityFocus** y **BugTraq**.

Severity	SLA	Status	Type	Date Discovered	Age	Reporter	CWE	Found By
High	Inactive	Active	Dynamic	Aug 16, 2018	237 days	admin	CWE-119	Dependency Check Scan

**Description**

CVE-2018-2060: Injurer Restriction of Operations within the Bounds of a Memory Buffer  
Stack-based buffer overflow in LaxRuby.cgi (StalwartCGI) in Samba 4.7.3, as used by notepad++ 6.1.1 and earlier, allows man-in-the-middle attackers to execute arbitrary code via certain Ruby (.rb) files with long lines. NOTE: this was originally reported as a vulnerability in notepad++.

Figura 3.8 Visualizando el detalle de una vulnerabilidad.

Reference
NIST: 23962
source: 820
url: <a href="http://www.securityfocus.com/bid/23962">http://www.securityfocus.com/bid/23962</a>
name: 23962 in notepad++(4.1); (void) ruby file processing buffer overflow exploit.
source: 8207040
url: <a href="http://www.securityfocus.com/bid/23962">http://www.securityfocus.com/bid/23962</a>
name: 23962 in notepad++(4.1); (void) ruby file processing buffer overflow exploit.
source: 8207040
url: <a href="https://www.virusbyte.com/article/1/antivirus/1340348/188/388/">https://www.virusbyte.com/article/1/antivirus/1340348/188/388/</a>

Figura 3.9 Visualizando las referencias de una vulnerabilidad.

DefectDojo utiliza los **benchmarks** de OWASP ASVS para analizar las aplicaciones y verificar cada uno de los casos que ofrece la guía para detectar fallos de seguridad en la aplicación.

#	Description	L1	L2	L3	Applicable	Pass
10.1	Verify that a path can be used from a trusted CA to each Transport Layer Security (TLS) server certificate, and that each server certificate is valid.	/	/	/	✓	✓
10.10	Verify that proper verification resolution, such as Online Certificate Status Protocol (OCSP) Signing, is enabled and configured.	/	/	/	✓	✓
10.12	Verify that perfect forward secrecy is configured to mitigate passive attackers recording traffic.	/	/	/	✓	✓
10.13	Verify that only strong algorithms, ciphers, and protocols are used, through all the certificate hierarchy, including root and intermediary certificates of your selected certifying authority.	/	/	/	✓	✓
10.14	Verify that the TLS settings are in line with current leading practice, particularly as concern configurations, ciphers, and algorithms become measure.	/	/	/	✓	✓
10.2	Verify that TLS is used for all connections (including both external and backend connections) that are authenticated or that handle sensitive data or functions, and does not fall back to insecure or unencrypted protocols. Ensure the strongest alternative is the preferred.	/	/	/	✓	✓

Figura 3.10 Visualizando la checklist de OWASP.

9.20	Verify that all untrusted HTML input from WYSIWYG editors or similar is properly sanitized with an HTML sanitizer library or framework feature.	/	/	/	✓	✓
9.24	Verify that when data is transferred from one DOM context to another, the transfer uses safe JavaScript methods, such as using innerText or .val() to ensure the application is not susceptible to DOM Cross-Site Scripting (XSS) attacks.	/	/	/	✓	✓
9.25	Verify when parsing JSON in browser or JavaScript based frameworks, that JSON.parse is used to parse the JSON document. Do not use eval() to parse JSON.	/	/	/	✓	✓
9.27	Verify the application for Server Side Request Forgery vulnerabilities.	/	/	/	✓	✓
9.28	Verify that the application correctly restricts XML parsers to only use the most restrictive configuration possible and to ensure that dangerous features such as resolving external entities are disabled.	/	/	/	✓	✓
9.29	Verify that deserialization of untrusted data is avoided or is adequately protected when deserialization cannot be avoided.	/	/	/	✓	✓
9.3	Verify that server side input validation failures result in required rejection and are logged.	/	/	/	✓	✓
9.8	Verify that input validation routines are enforced on the server side.	/	/	/	✓	✓
9.9	Verify that a centralized input validation control mechanism is used by the application.	/	/	/	✓	✓
9.11	Verify that all consumers of cryptographic services do not have direct access to key material, isolate cryptographic processes, including master secrets and consider the use of a virtualized or physical hardware key vault (HSM).	/	/	/	✓	✓
9.72	Verify that Personally Identifiable Information (PII) and other sensitive data is stored encrypted while at rest.	/	/	/	✓	✓

Figura 3.11 Visualizando la checklist de OWASP.

En la sección **Engagements**, podemos ver las diferentes pruebas que se han realizado y los test que se han ejecutado para cada una de ellas, así como el detalle de cada tipo de test.

Name	Lead	Date	Length	Tests	Open	All	Duplicate	Updated	Status
AdHoc Import - Web, 10 Apr 2018		10th April	1 day	6	1	21	21	0	16 hours ago
AdHoc Import - P4, 17 Aug 2018		17th August	1 day	1	1	10	10	0	7 months, 2 weeks ago
Merge PullRequest	T. admin	12th October - 10th October	3 days	3	1	0	3	0	7 months, 2 weeks ago - Blocked

Figura 3.12 Visualizando engagements.

Type	Date	Lead	Findings	Duplicate	Notes
Build Scan	1 Aug 17, 2018 - Aug 17, 2018		10	0	

**Description:**  
Start a new assessment

**Test (1) Result (1) from 2 min 1 sec to 10 min 10 sec (AdHoc Import - P4)**

**Risk Acceptance:**  
No risk acceptances found

**AdHoc Import - P4, 17 Aug 2018**  
18:20:55

**Status:** In Progress  
**Dates:** 17th August - 17th August  
**Length:** 1 day **Last updated:** 18:20:55  
**Lead:** None Assigned  
**Tracker:** Not Specified  
**Repo:** Not Specified  
**Updated:** 7 months, 2 weeks ago  
**Created:** 7 months, 2 weeks ago

Figura 3.13 Visualizando el detalle de un engagement.

## 3.2 SonarQube

SonarQube <https://www.sonarqube.org> es una plataforma open source que nos permite medir la calidad del código (evaluación estática y de caja blanca, principalmente) y detectar malas prácticas a nivel de código fuente. Sonar obtiene métricas del código, trabaja con muchos lenguajes y, para ello, utiliza plugins y otras herramientas (PMD o Checkstyle), y muestra los resultados en un cuadro de mando.

Tenemos también la posibilidad de usar la plataforma en la nube con el servicio SonarQube as a Service <https://sonarcloud.io/about/sq>



Figura 3.14 SonarQube as a Service.

Mediante el análisis del código se obtienen informes sobre:

- Código duplicado
- Estándares de codificación
- Test
- Cobertura de pruebas
- Complejidad ciclomática
- Bugs potenciales
- Comentarios
- Diseño y arquitectura

### 3.2.1 El cuadro de mando de SonarQube

A través de la interfaz de SonarQube podemos ver, de forma detallada, los puntos débiles de nuestro proyecto, como errores potenciales en el código, escasez de comentarios, clases demasiado complejas o escasez de cobertura de las pruebas unitarias.

El cuadro de mando que aparece en SonarQube es el resultado de analizar nuestro código con los plugins o herramientas que Sonar trae instalados por defecto como herramientas de análisis dinámico (caja negra) y análisis estático de

código (caja blanca) como Checkstyle, PMD o Findbugs para obtener distintas métricas del software.

En la figura 3.15 podemos ver la imagen típica del cuadro de mando de métricas de Sonar.

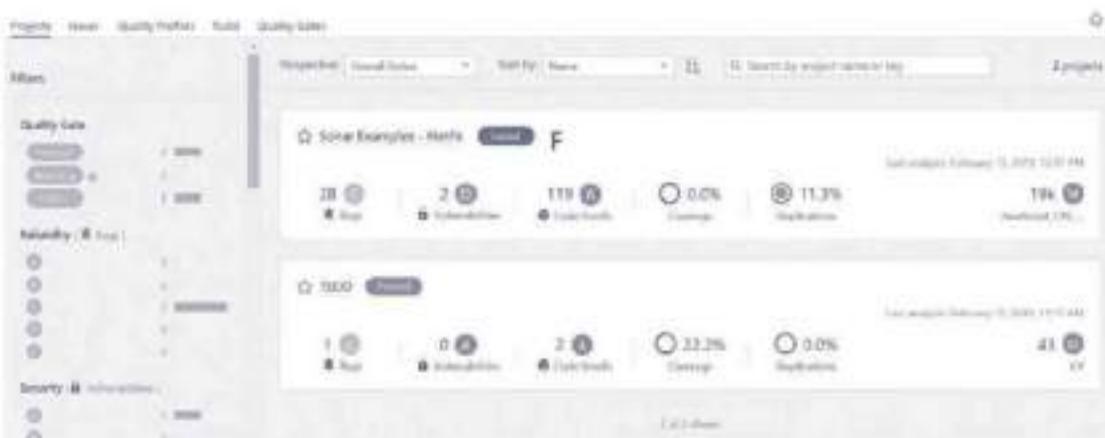


Figura 3.15 Cuadro de mando principal de SonarQube.

En el cuadro de mando principal podemos ver, para cada proyecto analizado, el número de bugs, las vulnerabilidades, las malas prácticas o Code Smells, el % de cobertura de pruebas y el porcentaje de código duplicado. De esta forma, obtenemos una visión general del estado de nuestro proyecto desde el punto de vista de desarrollo.

A nivel de cobertura de código, obtener un 100 % parece un objetivo alcanzable, sobre todo a medida que aumenta la complejidad; sin embargo, constituye un buen dato para que el equipo sepa qué porcentaje del código se ejecuta mediante las pruebas que han automatizado. Entornos de desarrollo como Visual Studio o IntelliJ pueden monitorizar qué código se ejecuta cuando lo hacen las pruebas para, de esta forma, saber qué partes de la aplicación no se están ejecutando.

Las métricas que muestra el cuadro de mando de Sonar por defecto para cada proyecto son las que veremos a continuación:

- **Complejidad.** Cuando en Sonar se habla de complejidad, se refiere a complejidad ciclomática. Para calcular la complejidad ciclomática, mantiene un contador de forma que, cuando el flujo de una función se altera, es decir, se produce un salto o se llama a una función, este contador de la complejidad aumenta en 1.

- **Código duplicado.** El código duplicado es una métrica básica de calidad y, junto con la complejidad ciclomática, es el mayor enemigo de la mantenibilidad. En Java, por ejemplo, se considera código duplicado si 10 líneas sucesivas de código están repetidas.
- **Issues.** Constituye el número de “malas prácticas” en el código, en función de las reglas definidas.
- **Diseño.** Esta métrica calcula los ciclos que hay entre paquetes en nuestro sistema. Cuando dos, tres o cuatro paquetes están involucrados en un ciclo, quiere decir que esos paquetes dependen unos de otros y no se pueden dividir. En consecuencia, la aplicación reduce su modularidad, lo que hace que sea más difícil de testear y menos mantenible.
- **Test.** Se muestra información acerca del porcentaje de cobertura de pruebas y, en el caso de que las haya, además, el número de test que han pasado y cuáles han fallado.

### 3.2.2 Issues por nivel de criticidad

Al ejecutar un análisis, SonarCloud plantea un problema cada vez que un fragmento de código rompe una regla de codificación. El conjunto de reglas de codificación se define a través del perfil de calidad asociado para cada lenguaje que se esté utilizando en el proyecto. Cada issue identificada se puede clasificar en las siguientes categorías:

- **Blocker (Bloqueante).** Se trata de un error con una alta probabilidad de afectar al comportamiento de la aplicación en producción; por ejemplo, por pérdida de memoria o conexión con base de datos no cerrada. El código tendría que ser arreglado antes de subirlo a producción, ya que estos errores pueden provocar una excepción del tipo `outofMemory`.
- **Critical (Crítico).** Se trata de un error que presenta una probabilidad menor, si lo comparamos con el caso anterior, de afectar al comportamiento de la aplicación en producción. Puede ser un problema de seguridad relacionado con inyección de SQL. El código tendría que ser revisado y modificado antes de subirlo a producción, aunque no es bloqueante para que la aplicación funcione.
- **Major.** Es un defecto de calidad, que puede tener un gran impacto en la productividad del desarrollador: código duplicado o uso incorrecto de parámetros.

- **Minor.** Se corresponde con un defecto de calidad, que puede afectar ligeramente a la productividad del desarrollador: el número de líneas por método es elevado o faltan por cubrir con test parte del código.
- **Info.** Se trata de un mensaje informativo a nivel de recomendaciones.



Figura 3.16 Visualización de issues por nivel de criticidad.

Usar Sonar puede ser uno de los primeros pasos a la hora de intentar mejorar el código de nuestra aplicación, y de descubrir zonas en las que, seguramente, sea necesario refactorizar.



Figura 3.17 Refactorización sugerida por Sonar.

### 3.2.3 Perfiles de calidad

Los perfiles de calidad son colecciones de reglas para aplicar durante un análisis. Para cada lenguaje, existe un perfil predeterminado. Todos los proyectos

no asignados explícitamente a algún otro perfil se analizarán con el valor pre-determinado. Idealmente, todos los proyectos usarán el mismo perfil para un lenguaje.



Figura 3.18 Perfiles de calidad en Sonar.

En SonarCloud, los analizadores contribuyen con reglas que se ejecutan en el código fuente y podemos encontrar cuatro tipos de reglas:

- **Code Smell** (dominio de mantenimiento)
- **Bug** (dominio de fiabilidad)
- **Vulnerabilidad** (dominio de seguridad)
- **Hotspot de seguridad** (dominio de seguridad)

Para las reglas definidas como Code Smell y bugs, no deberían aparecer falsos positivos. Al menos, este es el objetivo para que los desarrolladores no tengan que preguntarse si se necesita una solución. Para vulnerabilidades, el objetivo es que más del 80 % de los problemas sean reales y no falsos positivos. Las reglas de hotspot de seguridad están diseñadas a propósito para alertar sobre código que puede conllevar riesgos de seguridad. Se espera que más del 80 % de los problemas se resuelvan rápidamente después de que un auditor de seguridad los revise.

Para las reglas relacionadas con la seguridad, esto puede cambiar cuando se detectan falsos positivos; por ejemplo, muchas reglas de seguridad hacen referencia a cómo se deben manejar los datos "confidenciales" (como que no se almacenan sin cifrar). Pero, como en una regla no es posible decir qué datos son sensibles y cuáles no, este tipo de reglas cuentan con un abanico más amplio de opciones y es posible que obtengamos falsos positivos. La idea reside en que la regla marque cualquier cosa sospechosa y dejar que el auditor de seguridad eli-

mine los falsos positivos de forma manual. En este punto, podemos diferenciar entre vulnerabilidades y hotspots de seguridad.

Las **vulnerabilidades** son puntos en el código que están abiertos para que un posible atacante se aproveche de ese fallo.

**Los hotspots de seguridad** son un tipo especial de problema con el que se identifican áreas sensibles del código, que debe ser revisado por un auditor de seguridad para determinar si son realmente vulnerabilidades. El objetivo principal de los hotspots de seguridad consiste en **marcar problemas potenciales de originar una vulnerabilidad real** y ayudan a concentrar los esfuerzos de los auditores de seguridad, que revisan manualmente el código fuente de la aplicación para ver si el código sensible se está utilizando de una forma segura.

Una regla de vulnerabilidad resalta las amenazas de seguridad solo cuando existe un alto porcentaje de que constituyan una vulnerabilidad real, mientras que una regla de hotspot guía las revisiones de código al mostrar el código donde esos problemas pueden producirse, pero no tienen por qué originar una vulnerabilidad real.

Una vez que se haya detectado que realmente existe un problema en un punto concreto, se asignará al desarrollador apropiado, que realizará una corrección, y luego solicitará una revisión. Esta solicitud traslada el problema de vulnerabilidad a hotspot. A partir de ahí, depende del auditor de seguridad aceptar o rechazar la corrección. Si rechaza la solución, volverá al estado de vulnerabilidad para que la revise el desarrollador.

Por ejemplo, si filtramos por el tipo de vulnerabilidad, podemos ver todas aquellas **reglas asociadas a los diferentes lenguajes que permiten detectar posibles fallos de seguridad**.



Figura 3.19 Reglas asociadas por tipo de vulnerabilidad.



Figura 3.20 Propuesta de revisión para una función con vulnerabilidad crítica.

### 3.2.4 Reglas SonarQube

De forma predeterminada, al entrar en el elemento del menú superior “Reglas”, visualizará todas las reglas disponibles que traen los analizadores en SonarCloud. Puede restringir la selección según los criterios de búsqueda en el panel izquierdo:

- **Lenguaje:** El lenguaje al que se aplica una regla.
- **Tipo:** Error, vulnerabilidad, code smell o reglas de hotspot de seguridad.
- **Etiqueta:** Es posible agregar etiquetas a las reglas para clasificarlas y ayudar a descubrirlas más fácilmente.
- **Repositorio:** El motor que aporta reglas a SonarCloud.
- **Criticidad predeterminada:** La criticidad original de la regla, tal como lo define el analizador que contribuye con esta regla.
- **Estado:** Las reglas pueden tener tres estados diferentes.
  - **Beta:** La regla se ha implementado recientemente y aún no hemos recibido suficientes comentarios de los usuarios, por lo que puede haber falsos positivos o falsos negativos.
  - **En desuso:** La regla ya no debe usarse porque existe una regla similar más actualizada.
  - **Lista:** La regla está lista para ser utilizada con código en producción.

- **Disponible desde:** Fecha en la que una determinada regla se añadió en SonarCloud. Esto es útil para enumerar todas las nuevas reglas desde la última actualización de un complemento.
- **Plantilla:** Muestra las plantillas que permiten crear reglas personalizadas.
- **Perfil de calidad:** Inclusión o exclusión de un perfil específico.

La gran mayoría de las reglas relacionadas con la seguridad se origina a partir de estándares establecidos como **CWE**, **SANS Top 25** y **OWASP Top 10**. Para buscar reglas relacionadas con cualquiera de estos estándares, puede buscar reglas por etiqueta o por texto.

CWE es una lista o diccionario de las debilidades comunes que se producen a nivel de arquitectura, diseño, código o implementación del software que puede conducir a vulnerabilidades de seguridad explotables. CWE fue creado para servir como un lenguaje común para describir las debilidades de seguridad del software, además de proporcionar un estándar de referencia común para tareas de identificación, mitigación y prevención de vulnerabilidades.

Podemos consultar qué reglas de CWE están cubiertas para un determinado lenguaje.

- **C, C++:**
  - <https://rules.sonarsource.com/c/tag/cwe>
  - <https://rules.sonarsource.com/cpp/tag/cwe>
- **Java:** <https://rules.sonarsource.com/java/tag/cwe>
- **Objective-C:** <https://rules.sonarsource.com/objective-c/tag/cwe>

Rule ID	Description	Count	Language	Severity
S372	Constant number generated in IPNOC should not be used in secure contexts	1000	Java	Medium, Low, Info, High, Critical
S373	Asynchronous visibility should be specified	1000	Java	Medium, Low, Info
S374	"Access-Drop" strategy should be "permissive"	1000	Java	Medium, Low, Info
S375	Using a fixed seed with SecureRandom	1000	Java	Medium, Low, Info
S376	"Object::isFinal()" should return protocol (private/public) when overriding	1000	Java	Medium, Low, Info
S377	DSS (Data Encryption Standard) and Triple DES (3DES) should not be used	1000	Java	Medium, Low, Info
S378	Only standard cryptographic algorithms should be used	1000	Java	Medium, Low, Info
S379	SHA-1 and MessageDigest::hashAlgorithm should not be used	1000	Java	Medium, Low, Info
S380	Values passed to CT commands should be sanitized	1000	Java	Medium, Low, Info
S381	IP addresses should not be hardcoded	1000	Java	Medium, Low, Info

Figura 3.21 Reglas de seguridad en java.

### 3.2.5 Informes de seguridad en SonarQube

Los informes de seguridad están diseñados para ofrecer rápidamente una visión general de la seguridad de su aplicación, con detalles de su situación con respecto a cada una de las categorías top 10 de OWASP, top 25 de SANS y detalles específicos de CWE. Los informes de seguridad son alimentados por los analizadores de cada lenguaje, que se basan en las reglas activadas en los perfiles de calidad. Si no hay reglas correspondientes a una categoría OWASP determinada activada en su perfil de calidad, no obtendrá ningún problema vinculado a esa categoría específica, lo que significa que es posible que necesite activar más reglas.

A nivel de seguridad, podemos ver los reportes específicos para **OWASP Top10** y **SANS Top 25**. El top 10 de OWASP conforma una lista de categorías de vulnerabilidades, cada una de las cuales se puede asignar a muchas reglas individuales. Las etiquetas utilizadas para OWASP corresponden a las categorías **owasp-a1**, **owasp-a2**, **owasp-a3**, **owasp-a4**, **owasp-a5**, **owasp-a6**, **owasp-a7**, **owasp-a8**, **owasp-a9** y **owasp-a10**.

Categorías	Vulnerabilidades	Revisado	En revisión	Mirar más
A1 - Inyección	1	0	0	0
A2 - Riesgos de Autenticación	1	0	0	0
A3 - Desarrollo Data Exposición	1	0	0	0
A4 - XSS Universal (Cross Site Scripting)	1	0	0	0
A5 - Evitar Acceso Controlado	1	0	0	0

Figura 3.22 Top 10 OWASP.

Categorías	Vulnerabilidades	Revisado	En revisión	Mirar más
A1 - Inyección	1	0	0	0
CWE-01 - Improper Input Validation	1	0	0	0
CWE-02 - Improper Limitation of a Pathname to a Restricted Directory (Path Traversal)	1	0	0	0
CWE-03 - Improper Neutralization of Input During Web Page Generation (Cross-site Scripting)	1	0	0	0
CWE-04 - Cross-Site Request Forgery (CSRF)	1	0	0	0
A2 - Riesgos de Autenticación	1	0	0	0
A3 - Desarrollo Data Exposición	1	0	0	0
CWE-05 - Improper Input Validation	1	0	0	0

Figura 3.23 Top 10 OWASP.

**La lista de SANS** es una recopilación de los 25 errores más críticos enumerados en el CWE. La lista actual de SANS se divide en tres categorías: **sans-top25-insecure**, **sans-top25-risky** y **sans-top25-porous**. Para buscar reglas relacionadas con los 25 principales de SANS, puede realizar una búsqueda de texto para la categoría o el elemento CWE relevante.

The screenshot shows the SANS Top 25 website interface. At the top, there's a navigation bar with links like 'SANS Top 25', 'Search', 'About', 'Contact', and 'Logout'. Below the navigation, there's a message about security-related items not active in your profile. The main content area has three tabs: 'Categories' (selected), 'Vulnerabilities' (disabled), and 'Security Metrics' (disabled). Under 'Categories', there are three items: 'Memory Defenses' (selected), 'Identity Management' (disabled), and 'Resource Interactions Between Components' (disabled). Each item has a small icon and a link to its details.

Categories	Vulnerabilities	Security Metrics
Memory Defenses	0	0
Identity Management	0	0
Resource Interactions Between Components	0	0

Figura 3.24 Top 25 SANS.

### 3.2.6 SonarQube Plugins

Además de sus funciones básicas, la herramienta se puede ampliar con numerosos plugins Sonar, los cuales añaden diferentes funcionalidades, métricas, modos de visualizar datos, análisis de nuevos lenguajes de programación o integración con IDE. Disponemos de plugins para múltiples lenguajes: <https://docs.sonarqube.org/display/PLUG/Plugin+Library>



Figura 3.25 Plugins de SonarQube.

SonarCloud es capaz de realizar análisis en más de veinte lenguajes diferentes. En todos los lenguajes, se realiza un análisis estático del código fuente. Además,

para determinados lenguajes como Java y C#, es posible realizar un análisis estático del código compilado (archivos .class en Java o archivos .dll en C#). Cada lenguaje tendrá definidas sus propias reglas de detección de código, cuya lista de reglas podemos consultar en <https://rules.sonarsource.com>

Disponemos de plugins para analizar aplicaciones Android, entre los que podemos destacar:

- Sonar Android Lint plugin: <https://github.com/SonarCommunity/sonar-android>
- Quality Analysis Tools: <https://github.com/stephanenicolas/Quality-Tools-for-Android>

El plugin de sonar para Android mejora el plugin de Java, al proporcionar la capacidad de importar los informes de Android Lint. La idea consiste en visualizar los errores de Android Lint directamente en Sonar.

LANGUAGE	NAME	DESCRIPTION
Java (java)	Java	Analyze Java code with FindBugs 3.0.0.
XML (xml)	XML	SonarQube rule engine.

Figura 3.26 Plugin para Android.

Dentro del repositorio de github, encontramos un fichero rules.xml con todos los casos que es capaz de detectar.

This branch is 148 commits ahead of peter-budo/master.

PR dautureau-sonarsource and benzanico BUILD-82 Setup QA and setup module maven structure (#13) · 13

PR dautureau-sonarsource and benzanico BUILD-82 Setup QA and setup module maven structure (#13) · 13

Figura 3.27 Repositorio github para el plugin sonar-android.

Desde el punto de vista de la seguridad, podemos destacar las siguientes reglas y los casos que puede detectar:

- **Falta @JavascriptInterface en los métodos.** A partir de la API 17, debería anotar los métodos con @JavascriptInterface en el caso de aquellos objetos registrados con el método addJavascriptInterface.
- **Datos compartidos por el proveedor de contenido.** El elemento grant-uri-permission permite compartir rutas específicas. Esta regla busca una url que contenga solo el path '/', que faculta acceder a todo el contenido.
- **Usando setJavaScriptEnabled.** No debería utilizar setJavaScriptEnabled si no está seguro de que su aplicación realmente requiera soporte de JavaScript.
- **Exportación de proveedores de contenido.** Los proveedores de contenido se exportan de forma predeterminada y cualquier aplicación en el sistema puede usarlos para leer y escribir datos. Si el proveedor de contenido proporciona acceso a datos confidenciales, debe protegerse especificando export = false en el manifiesto o protegiéndolo con un permiso que pueda otorgarse a otras aplicaciones.
- **Exportación de servicios.** Los servicios exportados (servicios que configuran export = true o contienen un intent filter y no especifican export = false) deberían definir un permiso que una entidad debe tener para iniciar el servicio o vincularlo. De otra forma, cualquier aplicación puede utilizar este servicio.
- **Valor de android:debuggable en el manifiesto.** Resulta mejor no incluir este atributo en el manifiesto. Cuando realice una compilación y generación del APK, se configurará automáticamente como falso para evitar el modo depuración.
- **Uso de MODE\_WORLD\_READABLE en las llamadas a openFileOutput().** Hay casos en los que es apropiado que una aplicación permita la lectura de archivos por todos los usuarios y aplicaciones, pero estos deben revisarse, para asegurar que no contengan datos confidenciales.
- **Uso de MODE\_WORLD\_WRITEABLE en las llamadas a openFileOutput().** Existen casos en los que resulta apropiado que una aplicación permita la escritura de archivos por todos los usuarios y aplicaciones, pero estos deben revisarse, para asegurar que no contengan datos confidenciales.

- Uso del atributo allowBackup=true.** El atributo allowBackup determina si se pueden realizar copias de seguridad y restaurar los datos de una aplicación. Cuando este atributo se establece en valor true, el usuario puede realizar una copia de seguridad de los datos de la aplicación y restaurarlos mediante el comando adb backup. La configuración allowBackup = "false" excluye una aplicación, tanto de la copia de seguridad como de la restauración.

También disponemos de otros plugins para analizar aplicaciones desarrolladas con Objective-C: <https://github.com/octo-technology/sonar-objective-c>



Figura 3.28 Lenguajes soportados por Sonar.

Para cada lenguaje, podemos ver las reglas definidas por tipo (**Vulnerabilidad**, **Bug**, **Security Hotspot** o **Code Smell**) y por tag (android, api-design o bad-practice).

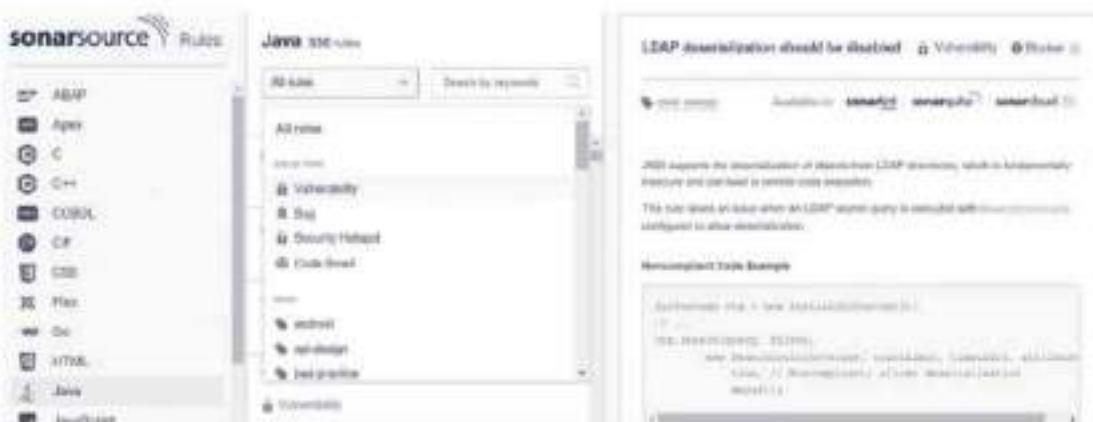


Figura 3.29 Reglas para el lenguaje Java.

### 3.2.7 Vulnerabilidades más comunes y explotadas

Las siguientes listas contienen información relacionada de errores comunes de programación con vulnerabilidades explotadas:

- <https://cwe.mitre.org/data/lists/900.html>
- <https://www.sans.org/top25-software-errors>

Una de las más explotadas es la correspondiente al **buffer overflow** [https://www.owasp.org/index.php/Buffer\\_Overflows](https://www.owasp.org/index.php/Buffer_Overflows), que se produce por no validar el tamaño de la información que se está recibiendo en una función con manejo de memoria. Podemos encontrar otro tipo de herramientas open source, como la de OWASP Dependency Check ([https://www.owasp.org/index.php/OWASP\\_Dependency\\_Check](https://www.owasp.org/index.php/OWASP_Dependency_Check)), para la detección de vulnerabilidades en bibliotecas de Java.

Para las vulnerabilidades que se detecten, se hará una evaluación de su exposición mediante el estándar CVSS y se tomarán las medidas apropiadas, con el fin de minimizar los riesgos asociados a dicha exposición. La mayoría de las fuentes de vulnerabilidades ya incluyen el nivel CVSS, junto a los detalles de la vulnerabilidad, pero, de no ser así, el responsable del entorno deberá determinar el nivel de criticidad de la vulnerabilidad.

La métrica de evaluación de vulnerabilidades CVSS permite clasificar las diferentes vulnerabilidades mediante una puntuación, con lo que proporciona un método estándar para estimar el impacto de una vulnerabilidad mediante tres componentes principales (Base, Temporal y Entorno).

- **Grupo Base:** Engloba las cualidades intrínsecas de una vulnerabilidad y que son independientes del tiempo y el entorno.
- **Grupo Temporal:** Incluye las características de la vulnerabilidad que cambian en el tiempo.
- **Grupo Entorno:** Contiene vulnerabilidades relacionadas con el entorno del usuario.

## 3.3 Find Security Bugs

Find Security Bugs <http://find-sec-bugs.github.io> es un software open source <https://github.com/find-sec-bugs/find-sec-bugs> que se puede instalar en los principales entornos de desarrollo para Java, como Eclipse, IntelliJ, Android Studio y SonarQube, para auditorías de seguridad de aplicaciones web Java.



Figura 3.30 Portada del proyecto FindSecurityBugs.

Entre las principales características, podemos destacar:

- Dispone de un diccionario extensible de vulnerabilidades, agrupado en categorías, y puede detectar 131 tipos diferentes de vulnerabilidades.
- Ofrece soporte a diferentes frameworks Java, como Spring-MVC y Struts.
- Se puede utilizar con sistemas de integración continua, como Jenkins y SonarQube.
- Se proporcionan referencias y enlaces OWASP top 10 y CWE, cada vez que se detecta un aviso en el que se indica el tipo de bug que podría producirse.

La instalación para cada uno de los entornos de desarrollo soportados se realiza a través del plugin <https://spotbugs.github.io>. **SpotBugs** es un plugin que realiza análisis estático en aplicaciones Java y es capaz de analizar programas compilados para cualquier versión de Java, desde 1.0 hasta 1.9.



Figura 3.31 Portada del proyecto SpotBugs.

El plugin se muestra compatible con proyectos basados en:

- **Eclipse:** <http://spotbugs.readthedocs.io/en/latest/eclipse.html>

- **Gradle:** <http://spotbugs.readthedocs.io/en/latest/gradle.html>
- **Maven:** <http://spotbugs.readthedocs.io/en/latest/maven.html>
- **Ant:** <http://spotbugs.readthedocs.io/en/latest/ant.html>

Por ejemplo, en el caso de integrar el plugin con un proyecto basado en gradle, bastaría con añadir la siguiente dependencia al fichero de gradle:

```
dependencies {
    spotbugsPlugins 'com.h3xstream.findsecbugs:findsecbugs-plugin:1.7.1'
}
```

En la url <http://find-sec-bugs.github.io/bugs.htm> se pueden ver los casos que es capaz de detectar para diferentes lenguajes de programación, como Java, Scala y JavaScript. A continuación vemos algunos ejemplos de código vulnerable y una posible solución.

### 3.3.1 Inyección potencial de Android SQL

Los valores de entrada en las consultas SQL deben pasarse de manera segura. Se pueden utilizar sentencias preparadas para resolver el riesgo de inyección de SQL.

Código vulnerable:

```
String query = "SELECT * FROM messages WHERE uid= '"+userInput+"'";  
Cursor cursor = this.getReadableDatabase().rawQuery(query,null);
```

Solución:

```
String query = "SELECT * FROM messages WHERE uid= ?";  
Cursor cursor = this.getReadableDatabase().rawQuery(query,new String[] {userInput});
```

### 3.3.2 Abrir un socket sin cifrar

Si el canal de comunicación utilizado no está encriptado, el tráfico puede ser leído por un atacante que intercepta el tráfico de la red. Además de usar un socket SSL, es importante asegurarse de que, al usar **SSLSocketFactory**, se realicen todas las comprobaciones de validación de certificados adecuadas para asegurarse de que no se halla sujeto a ataques de *man in the middle*.

Código vulnerable:

```
Socket socket = new Socket("www.google.com", 80);
```

Solución:

```
Socket socket = SSLSocketFactory.getDefault().createSocket("www.google.com", 443);
```

### 3.4 LGTM

LGTM <https://lgtm.com> es una herramienta que nos permite analizar los repositorios públicos de GitHub para la ejecución del análisis estático de código y análisis de vulnerabilidades. Entre las principales características, podemos destacar:

- Soporta los siguientes lenguajes: Java, TypeScript/JavaScript, Python, C/C++ y C#.
- Analiza el contenido de los proyectos cuyo código fuente se almacena en repositorios públicos alojados en BitBucket, GitHub y GitLab.
- Analiza cada revisión de un determinado proyecto que contenga vulnerabilidades.

Nos podemos autenticar con nuestra propia cuenta de GitHub. En la sección “My Projects” se muestra la lista de proyectos que tenemos analizados. Para añadir un nuevo proyecto, solo tenemos que poner la url de un repositorio público y pulsar en “Add”.



Figura 3.32 Añadir un proyecto para su posterior análisis.

LGTM tiene la capacidad de detectar los lenguajes de programación que se están utilizando en el proyecto y de realizar un análisis para cada uno de ellos.

For instant results: enable automated code review for pull requests. LGTM will catch new alerts in the code review process, and flag them up before the code is merged.

While LGTM analyzes the recent history of this project, results for the current state of the code base are already available on the Alerts tab. Whenever a new commit is added to the repository, the code is analyzed and the project history is updated.

LGTM will analyze recent commits first. If the code in this repository was last changed a while ago, the project history may take longer to appear.



Figura 3.33 Información sobre los proyectos que es capaz de analizar.

En la pestaña “Compare” nos da una comparación entre el proyecto analizado con respecto al estado de otros proyectos a nivel de alertas por líneas de código.

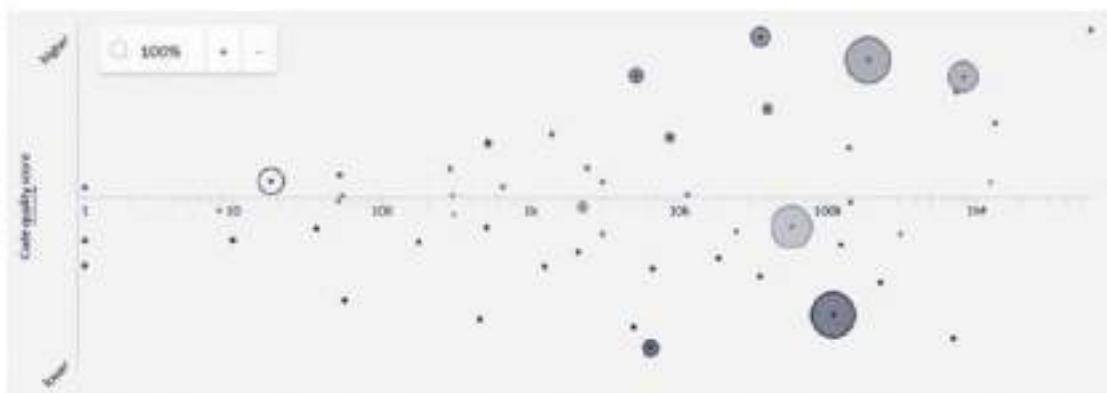


Figura 3.34 Información sobre el estado de nuestro proyecto a nivel de calidad de código.

Los proyectos se evalúan según la calidad del código y ofrecen información sobre el impacto de cada commit. Cuando se realiza un commit, se analiza frente a un conjunto de reglas, dependiendo de cada lenguaje, cada una de las cuales corresponde a un aspecto particular de las mejores prácticas de programación para ese lenguaje. El resultado son datos que muestran las tendencias en productividad y calidad para un determinado proyecto.

Encontrar problemas con el código y solucionarlos en la rama que estemos trabajando resulta útil antes de que el código realice el merge con el repositorio principal. Si posee o administra un repositorio analizado por LGTM, puede activar la revisión de código automatizada cada vez que se realice un merge con una rama. Para ello, podemos **activar el modo code review cada vez que efectuemos un pull request**.



Figura 3.35 Activación del modo code review en nuestro proyecto.

Podemos realizar una búsqueda de las reglas de seguridad definidas por lenguaje. Podríamos buscar las reglas de Python con la cadena de búsqueda “language:Python security”.

Rule	Description
Incomplete URL encoding sanitization	Sanitizes URLs by removing characters that are often used for injecting
Use of a broken or weak cryptographic algorithm	Includes cryptographic algorithms that have been broken or are considered unsafe
Incomplete regular expression for hostnames	Matching against an incomplete regular expression may cause a Denial of Service attack due to the incomplete regular expression
Request without certificate validation	Requests without certificate validation

Figura 3.36 Reglas de seguridad para python.

También podemos tener una visión por proyectos y el detalle de cada regla definida, junto con un ejemplo.



Figura 3.37 Visualización por proyectos.

Incomplete URL substring sanitization

Open in query console

Query Help

Security check on the substrings of unescaped URLs are often vulnerable to bypassing.

Query pack: [ossindex-packs.py](#)

Query ID: [py3/Homelab/URL%20encoding%20sanitization](#)

Language: Python

Severity: [Informational](#)

Tags: [Informational](#), [Security](#), [Insecure](#), [Cross-site-scripting](#), [CSRF](#)

Displayed by default? Yes. Alerts for this query are visible by default, but can be hidden on a per-project basis. Learn how.

Sanitizing untrusted URLs is an important technique for preventing attacks such as request forgeries and malicious redirections. Usually, this is done

Figura 3.38 Detalle de una regla.

### 3.5 OSS Index

OSS Index <https://ossindex.net> es una herramienta centrada en detectar vulnerabilidades en dependencias y librerías de terceros. Los datos se encuentran expuestos, a disposición de la comunidad, a través de una API REST, así como varias herramientas de código abierto. Se pone especial énfasis en los paquetes de software, tanto en los que se utilizan para las bibliotecas de desarrollo como en los paquetes de instalación.

Ecosystems

Identify open-source security vulnerabilities across a wide range of components

 Bower	 Cargo	 Chocolatey
 Composer	 CRAN	 Drupal
 Go	 Maven	 NPM
 NuGET	 PyPI	 RPM
 RubyGems		

Scan your entire app for vulnerabilities... for free!

Figura 3.39 Repositorios de código soportados.

OSSIndex soporta varias tecnologías (JavaScript, Python, Java o .net) y extrae información de dependencias de los principales repositorios, como NPM, Nuget,

Maven Central Repository o Bower. Podemos realizar la búsqueda por nombre de componentes y ver el detalle.

Components	
Query: angular	
Component	Description
 <a href="#">org.bower:angular</a>	Bower package for the stable branch of AngularJS
 <a href="#">org.bower:angular_animejs.css</a>	Anime.js animation for AngularJS
 <a href="#">org.bower:angular_dreamfactory_user_management_module</a>	AngularJS module that authenticates and manages a single user for the DreamFactory Services Platform

Figura 3.40 Componentes que usan angular.

## References

Type	URL
PACKAGE	<a href="https://bower.herokuapp.com/package/angular">https://bower.herokuapp.com/package/angular</a>
SOURCE	<a href="https://github.com/angular/bower-angular.git">https://github.com/angular/bower-angular.git</a>

3 references

## Vulnerabilities



Figura 3.41 Vulnerabilidades detectadas para un componente angular.

También podemos recuperar información sobre **vulnerabilidades de las bases de datos NIST, NVD y CVE**. Para ver el detalle de las vulnerabilidades, se necesita estar registrado.

## CVE

Common Vulnerabilities and Exposures

CVE is a list of entries, each containing an identification number, a description, and at least one public reference—for publicly known cybersecurity vulnerabilities.

### Attributions

CVE logo and marks are used in accordance with [CFI Terms of Use](#).

### Related

 Common Vulnerability Scoring System



Figura 3.42 Base de datos de vulnerabilidades CVE.



Figura 3.43 Base de datos de vulnerabilidades NVD.

### 3.6 Snyk

Snyk <https://snyk.io> es un servicio que permite detectar vulnerabilidades y librerías desactualizadas en diferentes tipos de proyectos, como Java, JavaScript o Python. Si se trata de un proyecto de **JavaScript**, es capaz de analizar las dependencias del fichero **package.json** e informar acerca del problema de seguridad para cada una de ellas. No solo ofrece herramientas para detectar vulnerabilidades conocidas en proyectos de JavaScript, sino que también ayuda a los usuarios a solucionar dichos problemas mediante actualizaciones guiadas y parches de código abierto que crea Snyk.



Figura 3.44 Ejemplo de vulnerabilidad detectada en una librería de JavaScript.

Snyk posee su propia base de datos de vulnerabilidades, obtenidos del NIST NVD y el NSP. El enfoque de Snyk reside en escalar el manejo de vulnerabilidades conocidas en toda la organización con mejores herramientas de colaboración e integraciones con repositorios de GitHub.



Figura 3.45 Detalle de la vulnerabilidad detectada en el módulo negotiator.

Desde el punto de vista de la seguridad, se recomienda tener siempre actualizadas las librerías de las que depende el proyecto, debido a que la mayoría de los proyectos que gestionan el desarrollo de estas librerías no proporcionan parches de seguridad para sus vulnerabilidades, sino que, simplemente, arreglan el problema en las siguientes versiones. Por eso, la actualización de las versiones más recientes resulta crítica.

### 3.7 Otras herramientas de análisis estático

La revisión estática permite un análisis de seguridad sobre el código fuente o compilado de la aplicación, con lo que se obtienen vulnerabilidades o indicios que pueden ser comprobados posteriormente en un proceso de análisis dinámico. Dentro del mercado, podemos encontrar otras herramientas para detectar fallos en las aplicaciones, no solo aquellos relacionados con la seguridad, en proyectos de código abierto Java, C/C++, C#, JavaScript, Ruby o Python:

- <https://matthias-endler.de/awesome-static-analysis>
- <https://github.com/mre/awesome-static-analysis>
- <https://scan.coverity.com>
- <https://security-code-scan.github.io>

Proyecto que se enfoca hacia la identificación de vulnerabilidades potenciales en proyectos .net, como la inyección de SQL, cross-site scripting (XSS), CSRF, uso de funciones de criptografía débil y contraseñas hardcodeadas.

### 3.8 Checklist de seguridad

Las checklists nos permiten realizar un seguimiento de aquellas tareas que podrían ser repetitivas dentro de un proyecto, con el objetivo de controlar el cumplimiento de una lista de requisitos o recolectar datos de forma sistemática. Se usan para hacer comprobaciones sistemáticas de tareas para asegurar, en cierta medida, que el desarrollador o tester no pasa por alto algo crítico para la aplicación. Los usos principales de las checklists son los siguientes:

- Realización de tareas en las que es importante que no se olvide ningún paso, o las cuales deben hacerse con un orden establecido.
- Realización de pruebas donde se debe dejar constancia de cuáles han sido los pasos realizados.
- Examen o análisis de la localización de fallos de seguridad.
- Recopilación de datos para su futuro análisis.

Por ejemplo, si estamos trabajando con **PHP** como lenguaje de programación a nivel de backend, en la url <https://github.com/ismailtasdelen/php-security-check-list> encontramos una lista de verificación de seguridad para aplicaciones o frameworks desarrollados en PHP, como **drupal** <https://www.drupal.org> y **joomla** <https://framework.joomla.org>

En la dirección <https://cheatsheets.pragmaticwebsecurity.com> encontramos otras *checklists* relacionadas con el desarrollo de **aplicaciones web con Angular y JWT**. La primera le proporciona las mejores prácticas de seguridad en el OWASP top 10, al desarrollar con Angular.

ANGULAR AND THE OWASP TOP 10

The OWASP Top 10 is one of the most influential security documents of all time. But how do these top 10 vulnerabilities translate in a front-end JavaScript application?

This cheat sheet offers practical advice on handling the most relevant OWASP top 10 vulnerabilities in Angular applications.

Disclaimer: This is an introductory checklist on the most relevant top 10 items applied to front-end angular applications. Many advanced attack vectors apply to the rest of the chapter.

**1 UNEXPECTED BEHAVIOR: UNINTENDED VULNERABILITIES**

OWASP TOP 10

Plan for a systematic release schedule  
 Version and/or revision for known vulnerabilities  
 Setup automated dependency checking to receive alerts (NuGet offers automatic dependency checking via a free service)  
 Implement dependency checking into your build pipeline

**2 BROKEN AUTHENTICATION**

OWASP TOP 10

From an Angular perspective, the most important aspect of broken authentication is maintaining state after authentication. Many alternatives exist, each with their specific security considerations.

Decide if a stateless backend is a requirement  
Some backends are more secure, and easier to audit than others.

**3 CROSSED-GATES SECURITY**

OWASP TOP 10

**Preventing HTML/SCRIPT INJECTION IN ANGULAR**

Use interpolation with {{ }} to automatically apply escaping  
 Use safe property binding such as [innerHTML] or <ng-bind>  
 Use Binding to [innerHTML] to safely insert HTML data  
 Do not use regular angular \$sanitize() on untrusted data

**Preventing CODE INJECTION OUTSIDE OF ANGULAR**

Avoid direct DOM manipulation  
Do this via ElementRef or other client-side elements  
 Do not combine Angular with server-side dynamic pages  
 Use Ahead-Of-Time compilation (AOT)

**4 BROKEN ACCESS CONTROL**

OWASP TOP 10

**AUTHENTICATION CRIMES**

Figura 3.46 Checklist de seguridad para Angular.

La segunda le proporciona un conjunto de buenas prácticas y una lista de comprobación, al desarrollar con la librería **JWT(JSON Web Tokens)**.

## JSON WEB TOKENS (JWT)

JSON Web Tokens (JWTs) have become extremely popular. JWTs seem deceptively simple. However, to ensure their security properties, they depend on complex and often misunderstood concepts. This cheat sheet focuses on the underlying concepts. The cheat sheet covers essential knowledge for every developer producing or consuming JWTs.

### INTRODUCTION

A JWT is a convenient way to represent claims securely. A claim is nothing more than a key/value pair. One common use case is a set of claims representing the user's identity. The claims are the payload of a JWT. Two other parts are the header and the signature.

- ⌚ JWTs should always use the appropriate signature scheme
- ⌚ If a JWT contains sensitive data, it should be encrypted
- ⌚ JWTs require proper cryptographic key management
- ⌚ Using JWTs for sessions introduces certain risks

### JWT INTEGRITY VERIFICATION

Claims in a JWT are often used for security-sensitive operations. Preventing tampering with previously generated claims is essential. The issuer of a JWT signs the token, allowing the receiver to verify its integrity. These signatures are crucial for security.

SYMMETRIC SIGNATURES

header + payload	header + payload signature
------------------	----------------------------

### VALIDATING JWTs

Apart from the signature, a JWT contains other security properties. These properties help enforce a lifetime on a JWT. They also identify the issuer and the intended target audience. The receiver of a JWT should always check these properties before using any of the claims.

- Check the `exp` claim to ensure the JWT is not expired  
Alternatively verify lifetime using creation time in the `iat` claim
- Check the `iss` claim to ensure the JWT can already be used
- Check the `aud` claim against your list of trusted issuers
- Check the `aud` claim to see if the JWT is meant for you

Some libraries offer support for checking these properties. Verify which properties are covered, and implement these checks with your own.

### CRYPTOGRAPHIC KEY MANAGEMENT

The use of keys for signatures and encryption requires careful management. Keys should be stored in a secure location. Keys also need to be rotated frequently. As a result, multiple keys can be in use simultaneously. The application has to foresee a way to manage the JWT key material.

Figura 3.47 Checklist de seguridad para JWT.

## **Capítulo 4. Seguridad en aplicaciones Android**

### **4.1 Introducción al protocolo HTTPS**

HTTPS es un protocolo de aplicación basado en el protocolo HTTP, destinado a la transferencia segura de datos. El protocolo HTTPS utiliza un cifrado basado en SSL/TLS para crear un canal cifrado y ofrece la infraestructura para las comunicaciones cifradas entre clientes y servidores.

Para que esta comunicación sea segura, no basta con el uso del protocolo, sino que, además, se deberán considerar diferentes aspectos, para evitar que entidades maliciosas puedan interceptar los datos de la comunicación.

En este punto, se pretende abordar la validación de certificados de forma correcta en la plataforma Android, que es uno de los problemas de seguridad recurrentes en el uso de infraestructuras de clave pública (PKI).

#### **4.1.1 Conceptos básicos sobre certificados**

La capa de sockets seguros (SSL), ahora conocida como seguridad de la capa de transporte (TLS), es un componente fundamental para las comunicaciones encriptadas entre clientes y servidores. Una aplicación podría usar SSL de forma incorrecta, con lo cual los datos de una aplicación podrían ser interceptados por otras entidades a través de la red mediante ataques *man in the middle*.

En un caso de uso de SSL típico, el servidor se configura con un certificado que contiene una clave pública y una privada. Como parte del acuerdo entre un servidor y un cliente SSL, el servidor confirma que tiene la clave privada firmando su certificado con criptografía de clave pública.

Un servidor web que acepte conexiones HTTPS debe estar configurado con un certificado que contenga una clave pública, así como su correspondiente clave privada. Cuando el servidor recibe una petición de conexión mediante el protocolo SSL/TLS, se lleva a cabo un proceso para establecer la conexión segura.

Sin embargo, cualquiera puede generar su propio certificado y la clave privada, por lo que este proceso, desde el punto de vista del cliente, solo prueba que el servidor conoce la clave privada que coincide con la clave pública del certificado, pero no prueba que ese servidor sea un servidor de confianza. Una manera de resolver tal problema estriba en hacer que el cliente posea un con-

junto de uno o más certificados en los que confía. Si el certificado no se encuentra en ese conjunto de certificados en los cuales el cliente confía, el servidor no sería de confianza y no se debería realizar ningún intento de conexión.

Este enfoque presenta varias desventajas, debido a que los servidores han de ser capaces de actualizar sus claves a lo largo del tiempo (rotación de las claves), lo que hace, a su vez, que se sustituya la clave pública en el certificado por una nueva.

En el caso de que la aplicación cliente dependa de un conjunto de certificados de confianza, en el momento en el que se actualice el certificado del servidor, esto llevará, necesariamente, a una actualización de los certificados en el lado cliente, lo que, en la mayoría de los casos, lleva asociado distribuir nuevamente la aplicación, debido a un cambio en la configuración del servidor, lo que puede ser especialmente problemático si estos cambios en la configuración del servidor no están bajo el control del desarrollador de la aplicación cliente. Dicho enfoque también resulta problemático en el caso de que la aplicación cliente tenga que comunicarse con diferentes servidores.

Para hacer frente a estos inconvenientes, los servidores suelen contener certificados emitidos por las llamadas **autoridades de certificación (CA)**. Generalmente, las plataformas sobre las que corren las aplicaciones contienen una lista con las entidades emisoras que son consideradas de confianza. Al igual que un servidor, una CA dispone de un certificado y una clave privada. Al expedir un certificado para un servidor, la CA firma el certificado de servidor utilizando su clave privada. El cliente puede verificar que el servidor cuenta con un certificado emitido por una CA conocida dentro de la plataforma.

En general, se recomienda configurar los servidores para que utilicen certificados emitidos por alguna de las CA reconocidas por las plataformas sobre las que se ejecutan las aplicaciones clientes. Haciendo uso de este tipo de certificados, se eliminan muchos de los problemas conocidos en el proceso de validación de certificados por parte de los clientes.

Las aplicaciones que manejen información sensible, como pueden ser datos personales, datos bancarios, nombres de usuario o contraseñas, deben utilizar protocolos que cifren la comunicación entre el cliente y el servidor. De esta forma, en el caso de que la comunicación sea interceptada, el tráfico nunca se obtendrá en texto claro. TLS es un protocolo que proporciona, con soporte criptográfico, un canal seguro para la protección, confidencialidad y autenticación sobre la información que se transmite. Por considerarse crítica esta implementación de seguridad, resulta importante verificar la utilización de un algoritmo de cifrado fuerte y su correcta implementación.

La autenticación TLS, también conocida como “autenticación TLS mutua”, consiste en que ambos, navegador y servidor, envíen sus respectivos certificados TLS durante el proceso de negociación TLS (handshaking). Así como se puede validar la autenticidad de un servidor mediante el certificado y preguntar a una Autoridad de Certificación conocida (CA, por las siglas en inglés de Certificate Authority) si la certificación es válida, el servidor puede autenticar al usuario recibiendo un certificado desde el cliente y realizar la validación correspondiente contra una CA o su propia CA. Para hacer esto, el servidor debe proveer al usuario de un certificado generado específicamente para él, en el cual se asignen valores que puedan ser usados para determinar que el usuario debe validar el certificado. El usuario instala los certificados en el navegador o en el dispositivo y los usa para conectarse a la aplicación o al sitio web.

A continuación veremos las principales recomendaciones centradas en la validación de certificados que tendría que realizar una aplicación Android para asegurar las comunicaciones con un servidor remoto.

#### 4.1.2 Despliegues en producción

Android contiene más de cien entidades emisoras de certificado, que se actualizan en cada versión. Si ya disponemos de nuestro servidor configurado correctamente sobre SSL/TLS con un certificado emitido por una CA conocida por la plataforma Android, para realizar la conexión con dicho servidor y validar los certificados, se podría implementar de la siguiente forma:

```
URL url = new URL ("https://mi-servidor.com");
URLConnection urlConnection = url.openConnection();
InputStream in = urlConnection.getInputStream();
copyInputStreamToOutputStream(in, System.out);
```

Android implementa la verificación de los certificados y los nombres del host mediante el uso de sus APIs, siempre que se cumpla que han sido emitidos por una CA de las reconocidas por la plataforma.

Por eso, la recomendación general para aplicaciones que se desplieguen en producción será siempre: **“utilizar las APIs proporcionadas por la plataforma, las cuales implementan de manera correcta la validación de certificados, y utilizar certificados emitidos por una CA de confianza en el lado del servidor”**.

En el caso de producirse una excepción en la validación de certificados al intentar establecer una conexión segura, puede ser debido a uno de los siguientes motivos:

- La CA que emitió el certificado de servidor no está incluida en la lista de CA de confianza para la plataforma.
- El certificado del servidor que se está usando no ha sido firmado por una CA, o se está haciendo uso de certificados autofirmados.
- No se encuentra un certificado firmado por una CA de confianza en la cadena de certificados enviada por el servidor.

Para minimizar este tipo de errores, se recomienda siempre configurar los servidores con certificados emitidos por una CA de confianza y hacer uso de las apis proporcionadas por la plataforma.

Aunque no debería ser el escenario habitual, se puede dar el caso de que sea necesario utilizar un certificado firmado por una CA que no se halla dentro de las CA reconocidas por la plataforma. Esto podría pasar porque la CA, aun siendo una CA de confianza, no forme parte del conjunto de CA de confianza para la plataforma o en el caso de que la propia organización se encargue de emitir certificados para un uso interno.

En estos casos, se puede modificar el comportamiento del api Android para que nuestra aplicación confie en este tipo de certificados. A continuación se muestra un ejemplo de código para realizar este tipo de implementación, donde estamos confiando en una CA identificada por el certificado contenido en el fichero **"certificate.crt"**.

```
CertificateFactory cf = CertificateFactory.getInstance("X.509");
// Una CA se identifica a partir de su certificado.
// Cargar el CA a partir de un InputStream, en este caso el fichero certificate.crt,
// puede ser a partir de un File, ByteArrayInputStream
InputStream caInput = new BufferedInputStream(new FileInputStream("certificate.crt"));
Certificate ca;
try {
    ca = cf.generateCertificate(caInput);
    System.out.println("ca=" + ((X509Certificate) ca).getSubjectDN());
} finally {
    caInput.close();
}
```

```
// Crear un KeyStore que contenga los CA de confianza
String keyStoreType = KeyStore.getDefaultType();
KeyStore keyStore = KeyStore.getInstance(keyStoreType);
keyStore.load(null, null);
keyStore.setCertificateEntry("ca", ca);

// Crear un TrustManager a partir del KeyStore anterior para que confie en los CA // contenidos en este
String tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm();
TrustManagerFactory tmf = TrustManagerFactory.getInstance(tmfAlgorithm);
tmf.init(keyStore);

// Crear un SSLContext que use el TrustManager anterior
SSLContext context = SSLContext.getInstance("TLS");
context.init(null, tmf.getTrustManagers(), null);

//Configurar la conexión URLConnection para que haga uso de un SocketFactory //creado a partir de nuestro SSLContext
URL url = new URL("https://host.con.certificado.firmado.por.nuestra.ca/CATest/");
HttpsURLConnection urlConnection =
(HttpsURLConnection)url.openConnection();
urlConnection.setSSLSocketFactory(context.getSocketFactory());
InputStream in = urlConnection.getInputStream();
copyInputStreamToOutputStream(in, System.out);
```

El procedimiento consiste en sobrescribir el comportamiento por defecto de la clase **HttpsURLConnection**, para lo cual se utilizará un **SSLSocketFactory**, que se configura mediante el uso de un contexto **SSLContext** personalizado.

Este contexto se inicializa con un **TrustManager**, que será el encargado de definir el proceso de validación de certificados recibidos del servidor. Creando este **TrustManager** a partir de un **KeyStore** en el que se incluyen los certificados raíz que representan nuestra CA, se consigue que estos sean los certificados que se tomarán como de confianza.

**Advertencia:** Se ha observado que muchas de las aplicaciones hacen uso de una alternativa que, en ningún caso, debe ser implementada por motivos de seguridad. Esta alternativa consiste en sobrescribir el comportamiento del **TrustManager** por defecto, para que sus métodos devuelvan siempre como válido cualquier certificado. Hacer esto significa lo mismo que no implementar una conexión segura, ya que un posible atacante podría, de forma sencilla, suplantar el certificado y obtener tráfico de la aplicación mediante técnicas de *man in the middle*.

#### **4.1.3 Certificado de servidor autofirmado**

La segunda causa de error reside en el uso de certificados autofirmados, lo que significa que el mismo servidor actúa como autoridad certificadora. Este tipo de implementación no debería estar presente en sistemas de producción, por lo que, en el caso de ser necesaria una implementación de este tipo para entornos de desarrollo, la manera correcta de validar los certificados, para mantener una comunicación segura, sería utilizar una solución similar a la del caso de validar un certificado procedente de una CA desconocida.

Podemos crear nuestro propio TrustManager, esta vez haciendo que confie en el certificado de nuestro servidor directamente. Para considerar esta implementación segura, se deberá tener en cuenta que el certificado autofirmado que se genere ha de contar con una clave razonablemente fuerte.

#### **4.1.4 CA no encontrada dentro de la cadena de certificados**

El tercer caso se produce cuando no se encuentra una CA en la cadena de certificados. La mayoría de las CA públicas no firman los certificados de servidor directamente. En su lugar, usan su certificado raíz, para firmar los certificados de las CA intermedias. Hacen esto para poder almacenar su certificado raíz en un lugar seguro y reducir el nivel de riesgo. Aunque los sistemas operativos como Android normalmente solo confían en los certificados raíz directamente, los servidores, por lo habitual, no mandan solo su certificado, sino que envían en realidad una cadena de certificados que incluyen su propio certificado y todos los CA intermedios, necesarios para alcanzar el certificado raíz de la CA de confianza.

En algunos casos, los servidores están configurados para no devolver la cadena total de certificados. Esto es así debido a que la mayoría de los navegadores web almacenan una caché de los CA intermedios de confianza que han sido validados previamente. Por motivos de eficiencia, una vez que el navegador ha visitado y cacheado los certificados intermedios (por ejemplo, al acceder a la página principal), el servidor no vuelve a mandar toda la cadena de certificados (por ejemplo, al acceder a recursos como imágenes, CSS o JavaScript). El problema viene cuando ese servidor ofrece un servicio web que va a ser consumido por una aplicación que no tiene cacheados los certificados de estas CA intermedias.

Para solucionar dicho problema, se recomienda configurar el servidor, con el fin de que devuelva la cadena completa de certificados, incluyendo todos aquellos correspondientes a las CA intermedias; por lo menos, en el caso de las peticiones a los servicios web que puedan ser consumidos directamente desde una aplicación.

#### 4.1.5 Configuración de seguridad

A partir de la versión 7 Android Nougat, se introducen algunas configuraciones respecto a las comunicaciones mediante HTTPS. En nuestro fichero **AndroidManifest.xml**, podemos añadir la siguiente configuración en el objeto application:

```
<manifest>
    <application android:networkSecurityConfig="@xml/network_security_config">
        </application>
    </manifest>
```

La entrada anterior hace referencia a un fichero que debemos añadir en la ruta `res/xml/network_security_config.xml` con el siguiente formato:

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="false">
        <domain includeSubdomains="true">example.com</domain>
        <trust-anchors>
            <certificates src="@raw/ca">
        </trust-anchors>
    </domain-config>
</network-security-config>
```

La nueva configuración establece, de forma personalizada, las autoridades de certificación (CA) de confianza para las conexiones seguras de una aplicación. La configuración anterior se podría simplificar de esta forma, donde le indicamos que use los **certificados de confianza** del sistema operativo:

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <base-config>
        <trust-anchors>
            <certificates src="system">
            <certificates src="user">
        </trust-anchors>
    </base-config>
</network-security-config>
```

#### 4.1.6 Actualización de proveedores criptográficos

Para actualizar los proveedores criptográficos de la aplicación en Android, deberá incluir los servicios de Google Play. En el archivo de módulo de **build.gradle** habría que añadir la siguiente línea a la sección de dependencias:

```
implementation 'com.google.android.gms:play-services-safetynet:15.0.1'
```

La API de servicios de SafetyNet tiene muchas más funciones, incluida la API de navegación segura que comprueba las url para ver si han sido marcadas como una amenaza conocida, y una API reCAPTCHA, para proteger su aplicación de spammers y otro tráfico malicioso:

- <https://developer.android.com/training/safetynet>
- <https://developer.android.com/training/safetynet/safebrowsing>
- <https://developer.android.com/training/safetynet/recaptcha>

Después de sincronizar Gradle, puede llamar al método **installIfNeededAsync** de la clase **ProviderInstaller**:

```
public class MainActivity extends Activity implements ProviderInstaller.ProviderInstall-
Listener
{
    @Override
    protected void onCreate(Bundle savedInstanceState)

        super.onCreate(savedInstanceState);
        ProviderInstaller.installIfNeeded(this, this);

    }
}
```

#### 4.1.7 Android Certificate Pinning

En el caso de que la aplicación se quiera conectar a un servicio externo o a uno desarrollado por nuestro equipo, se recomienda validar el certificado que expone el servidor. De esta forma, el compromiso de una CA sería detectable, con lo que se evitaría enviar datos sensibles a un usuario que se encuentre in-

terceptando la comunicación realizando un *man in the middle*. Este control se conoce como Certificate Pinning.

**Certificate Pinning** es la práctica de almacenar un conjunto de información por defecto (generalmente hashes) para certificados digitales y claves públicas a nivel de agente del usuario (ya sea navegador web, aplicación móvil o complemento del navegador), de modo que solo los certificados/claves públicas predefinidos se utilizarán para establecer una comunicación segura, y todos los demás darán error, incluso si el usuario confía de forma implícita o explícita en los otros certificados que tenga instalados.

El problema radica en la falta de validación del certificado que se expone en la comunicación. La idea detrás de Certificate Pinning reside, precisamente, en poder detectar cuándo una cadena de confianza ha sido modificada. Para ello, se busca asociar inequívocamente un certificado digital a un dominio concreto.

De esta forma, un dominio A, por ejemplo, google.com, estará vinculado a un certificado'autoridad de certificación B específica. Si una autoridad de certificación C diferente (que depende de una autoridad de certificación raíz en la que se confía) intenta emitir un certificado asociado al dominio A, este hecho generará una alerta. Por este motivo, se puede definir un pin como la relación entre un nombre de host y una identidad de cifrado (en este documento, una clave pública en una cadena de certificados X.509).

Para el caso de las aplicaciones Android, el Certificate Pinning se consigue a través de una implementación propia de X509TrustManager. A modo general, consiste en redefinir el método `checkServerTrusted` de la clase `javax.net.ssl.X509TrustManager` para, de esta forma, limitar el conjunto de CA válidos.

Entre las principales ventajas para usar Certificate Pinning, podemos destacar:

- En el caso de que alguna de las otras CA reconocidas por el sistema fuera comprometida, nuestra aplicación no se vería afectada.
- En aplicaciones donde los usuarios pueden no comprender las advertencias de los certificados y es probable que permitan la instalación de un certificado caducado no válido. Al usar Certificate Pinning, estamos haciendo que estos certificados inválidos queden aislados en el uso de las aplicaciones.
- En entornos en los que los usuarios se ven obligados a aceptar una CA raíz potencialmente maliciosa, como entornos corporativos o esquemas nacionales de PKI.

#### 4.1.8 Cifrado extremo a extremo

Hay una tendencia reciente, llamada “**encriptación de extremo a extremo**”, en la que solo los dos dispositivos que forman parte de una comunicación pueden leer el tráfico. Un buen ejemplo es una aplicación de chat cifrada en la que dos dispositivos móviles se comunican entre sí a través de un servidor; solo el remitente y el destinatario pueden leer los mensajes del otro.

Una analogía para ayudarlo a comprender el cifrado de extremo a extremo es imaginar que desea que alguien le envíe un mensaje que solo usted puede leer. Para hacer esto, les proporciona una caja con un candado abierto (la clave pública), mientras mantiene la llave del candado (clave privada). El usuario escribe un mensaje, lo coloca en la caja, bloquea el candado y se lo envía al receptor. De esta forma, solo el receptor puede leer el mensaje, porque es el único con la llave para desbloquear el candado.

Con el cifrado de extremo a extremo, ambos usuarios se envían mutuamente sus claves. El servidor solo proporciona un servicio para la comunicación, pero no puede leer el contenido de la comunicación. Si desea obtener más información sobre este enfoque, un buen lugar para comenzar es el repositorio de GitHub para la **aplicación Signal** de código abierto:

- <https://www.signal.org/es/>
- <https://github.com/signalapp>

#### 4.1.9 Firmando una aplicación Android

Android requiere que todas las aplicaciones estén firmadas digitalmente con un certificado antes de poder instalarlas. Android utiliza este certificado para identificar al autor de una aplicación. Para ejecutar la aplicación en el dispositivo, debe estar firmada. Cuando la aplicación está instalada en un dispositivo, el administrador de paquetes verifica si la aplicación se ha firmado correctamente con el certificado en el archivo APK. La solicitud puede ser autofirmada o puede firmarse a través de CA.

Una aplicación de Android está empaquetada como un archivo APK (paquete de Android), que es, esencialmente, un archivo ZIP que contiene el código compilado, los recursos, la firma, el manifiesto y todos los demás archivos que necesita para ejecutarse.

Es importante decirle a Android que nuestro paquete APK, en realidad, sí se trata de una aplicación para Android, pues estos tienen ciertas características

que el sistema lee antes de su ejecución como tal. Para esta tarea, podemos usar la herramienta de firmar que viene integrada en Android Studio.



Figura 4.1 Paso 1. Generación de un APK firmado con Android Studio.

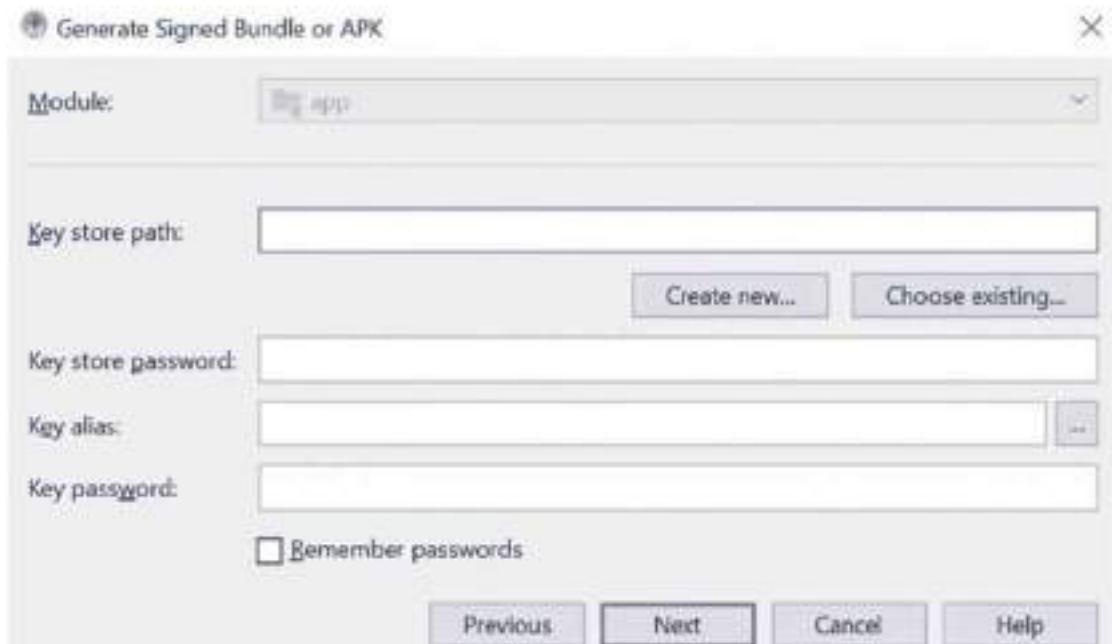


Figura 4.2 Paso 2. Generación de un APK firmado con Android Studio.

Aquí tenemos que seleccionar la ruta del almacén de claves o crear uno nuevo. Este almacén de claves se crea con la herramienta **keytool**, que viene cuando instala el JDK de Java en el sistema operativo.

Una vez hayamos firmado la aplicación, también podríamos optimizar el APK con una herramienta llamada “**zipalign**”, que se encuentra en la carpeta “tools”, donde tenemos instalado el Android SDK.

Zipalign se trata de una herramienta que viene incluida en el SDK (**PATH\android-sdk-windows\tools\zipalign.exe**), que está hecha para preparar nuestro APK para el sistema Android, es decir, para que su rendimiento sea lo más óptimo posible para dicho sistema.

El siguiente paso será copiar nuestro APK recién firmado a la carpeta “tools”, que es donde se encuentra la aplicación “**zipalign**”. Abrimos una consola, nos ubicamos en el directorio donde está la aplicación y ejecutamos el comando que nos permite generar un APK optimizado a partir del APK compilado original:

```
zipalign -f 4 fichero-compilado.apk fichero-final.apk
```

## 4.2 Principios fundamentales de desarrollo en Android

Android está integrado en el Kernel de Linux, que ofrece una funcionalidad básica del sistema, como la gestión de procesos, la gestión de memoria o el acceso a dispositivos como la cámara, el teclado o la pantalla. Como sistema operativo multiusuario, uno de los principales objetivos de seguridad reside en aislar los recursos de los usuarios entre sí. La filosofía de seguridad de Linux consiste en proteger los recursos de los usuarios unos de otros. El sistema operativo Android es un sistema Linux multiusuario, en el que cada aplicación constituye un usuario diferente; de esta forma se asegura que:

- El usuario A no acceda a los archivos del usuario B.
- El usuario A no acceda a la memoria del usuario B.
- El usuario A no acceda a los recursos de CPU del usuario B.
- El usuario A no acceda a los dispositivos del usuario B.

De forma predeterminada, el sistema asigna a cada aplicación un ID de usuario de Linux único. El sistema establece permisos para todos los archivos en una aplicación, de modo que solo el ID de usuario asignado a esa aplicación pue-

da acceder a ellos. Cada proceso cuenta con su propia máquina virtual (VM), por lo que el código de una aplicación se ejecuta de forma aislada de otras aplicaciones.

De manera predeterminada, cada aplicación se ejecuta en su propio proceso de Linux. Android inicia el proceso cuando se necesita ejecutar cualquiera de los componentes de la aplicación; luego, lo cierra, cuando ya no es necesario o cuando el sistema debe recuperar la memoria para otras aplicaciones. Esto crea un entorno seguro, en el que una aplicación no puede acceder a partes del sistema para las que no tiene permiso. Como todas las aplicaciones de Android, se ejecutan en su propio entorno de *sandbox* y no pueden afectar a otras aplicaciones de modo predeterminado.

Además, disponemos de las siguientes características:

- El SDK de Android es el encargado de compilar el código en un paquete con el sufijo .apk.
- Los ficheros APK son considerados como aplicaciones que se instalan en los dispositivos.
- Una vez instalado el paquete, cada aplicación se ejecuta en su propia sandbox de forma aislada del resto de aplicaciones, de manera que cada aplicación pertenece a un usuario y, cuando se instala una aplicación, tiene asignado un UID único.
- Solo el usuario cuyo UID esté asociado con una aplicación puede acceder a los ficheros de esta.
- Cada aplicación se ejecuta en una instancia de la máquina virtual de android Dalvik.
- Se fomenta la independencia entre procesos y aplicaciones.
- Es posible compartir información entre aplicaciones:
  - Pueden compartir el mismo UID, con lo que permiten acceder a los ficheros de ambas entre ellas.
  - Una aplicación puede solicitar permiso para acceder a la información de los dispositivos.
  - El usuario debe conceder tales permisos en tiempo de instalación.

#### 4.2.1 Componentes en Android

En la plataforma Android podemos encontrar los siguientes componentes, donde cada uno posee un propósito y un ciclo de vida distinto:

- **Actividades.** Representa una pantalla con una interfaz de usuario.
- **Servicios.** Son componentes que se ejecutan en segundo plano, realizan tareas que requieren un tiempo largo de ejecución. También pueden servir para procesar información de forma remota:
  - Un servicio es un componente que se ejecuta en segundo plano, para realizar operaciones que requieren de un largo tiempo de ejecución o para realizar peticiones que se ejecutan en remoto.
  - Un servicio, normalmente, no proporciona una interfaz de usuario.
  - Por ejemplo, un servicio puede reproducir música en segundo plano, mientras el usuario está en una aplicación diferente, o puede obtener datos de la red sin bloquear la interacción del usuario.
- **Proveedores de contenido:**
  - Podemos almacenar los datos en cualquier sitio que sea requerido posteriormente por nuestra aplicación.
  - Puede almacenar los datos en el sistema de archivos, en una base de datos SQLite o en cualquier otra ubicación de almacenamiento permanente a la que pueda acceder su aplicación.
  - A través del proveedor de contenido, otras aplicaciones pueden consultar, o incluso modificar, los datos (si el proveedor de contenido lo permite).
  - El proveedor de contenido resulta útil en los casos en que una aplicación quiere compartir datos con otra aplicación.
- **Receptores de broadcast.** Responden a mensajes de difusión en todo el sistema:
  - Un receptor de broadcast es un componente que responde a los anuncios de difusión en todo el sistema.
  - Aunque los receptores de broadcast no muestran una interfaz de usuario, pueden crear una notificación de barra de estado, para alertar al usuario cuando ocurre un evento de difusión.
  - Por ejemplo, una aplicación podría registrar un receptor para un determinado evento y cambiar su comportamiento en función de la información recibida.

#### 4.2.2 Android Lint

Incluso los mejores desarrolladores cometen errores en su código y, a lo largo de los años, se han desarrollado diferentes formas de solucionar determinados problemas que surgen al empezar con el desarrollo de una aplicación. Escribir pruebas unitarias ha demostrado ser muy efectivo y es algo que recomiendo a todos los desarrolladores. Sin embargo, incluso con pruebas unitarias, resulta difícil cubrir todas las situaciones posibles que pueden ocurrir en el código, por lo que es importante complementar las pruebas de unidad con otros métodos; por ejemplo, el **análisis de código estático**.

**Android Lint** <https://developer.android.com/studio/write/lint> constituye una herramienta muy importante en la fase de desarrollo, porque permite comprobar que sus implementaciones siguen las buenas prácticas de desarrollo Android y, de este modo, corregirlas. Nos permite realizar comprobaciones en los archivos de su proyecto, tanto XML como Java, y busca código mal estructurado, variables no utilizadas o funciones que se estén usando de forma insegura.

Para ejecutar la herramienta Lint en Android Studio, haga clic con el botón derecho en el proyecto y, a continuación, seleccione Analyze – Inspect Code (Analizar – Inspeccionar el código).

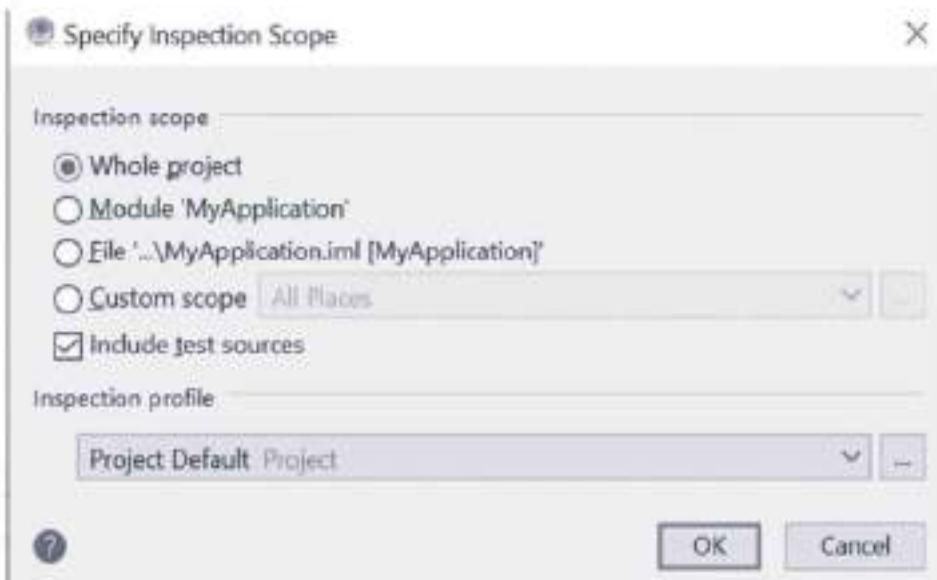


Figura 4.3 Ventana de ejecución del inspector de código.

En la consola podremos ver los resultados de analizar el código y ver los avisos/warnings y problemas detectados.



Figura 4.4 Resultados de análisis de código.

En Android Studio lo podemos configurar desde la opción **Configure inspections**.

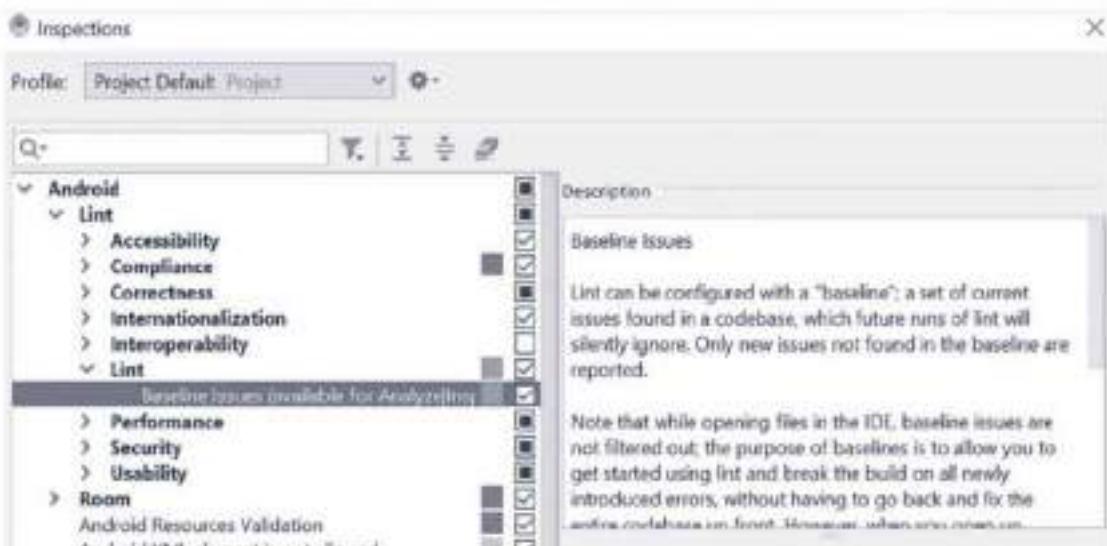


Figura 4.5 Configuración de Inspections.



Figura 4.6 Resultados de análisis con Android Lint.

Si usamos Eclipse en lugar de Android Studio, también podemos ver los avisos en la pestaña **Lint Warnings**.

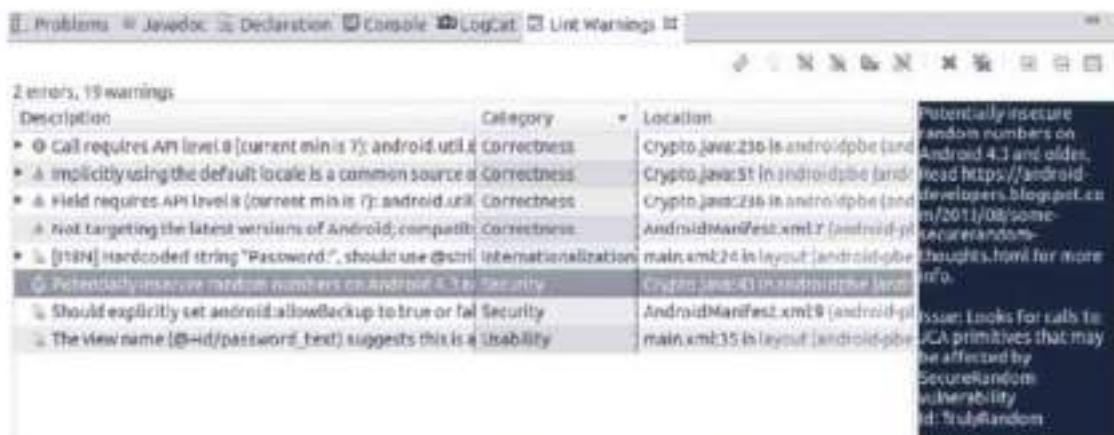


Figura 4.7 Lint Warnings en Eclipse.

También es posible ejecutar **Lint** desde una línea de comandos:

```
lint [flags] <project directory>
```

usage: lint [flags] <project directories>

Flags:

Flag	Description
--help	This message.
--help <topic>	Help on the given topic, such as "suppress".
--list	List the available issue id's and exit.
--version	Output version information and exit.
--exitcode	Set the exit code to 1 if errors are found.
--show	List available issues along with full explanations.
--show <ids>	Show full explanations for the given list of issue id's.

Enabled Checks:

Flag	Description
--disable <list>	Disable the list of categories or specific issue id's. The list should be a comma-separated list of issue id's or categories.
--enable <list>	Enable the specific list of issues. This checks all the default issues plus the specifically enabled issues. The list should be a comma-separated list of issue id's or categories.
--check <list>	Only check the specific list of issues. This will disable everything and re-enable the given list of issues. The list should be a comma-separated list of issue id's or categories.
-w, --nowarn	Only check for errors (ignore warnings)

Figura 4.8 Opciones del comando Lint.

Por ejemplo:

```
lint --check Accessibility --HTML informe.html myproject
```

Este comando permite comprobar si se cumplen las reglas definidas en su aplicación, y el resultado se encuentra disponible mediante un archivo HTML.



Figura 4.9 Lint Report como resultado de ejecutar el comando.

En el análisis estático, nos encargamos de revisar el código fuente de la aplicación; claro que no siempre tenemos a disposición el código fuente y habría que trabajar directamente con el \*.apk, siempre con el objetivo de averiguar qué es lo que hace. Una ventaja ante el análisis dinámico radica en que se puede saber exactamente qué es lo que realiza la aplicación en cuestión. También podríamos aplicar **ingeniería inversa** con el objetivo de analizar el código fuente, con el fin de buscar todas las llamadas a funciones, que podrían ser usadas de forma insegura como:

- `putString`
- `setJavaScriptEnabled(true)`
- `getExternalStorageDirectory()`
- `getBundleExtra()`

### 4.3 Ingeniería inversa en Android

Con el objetivo de conocer en profundidad el funcionamiento de la aplicación, entramos en la fase de reversing. La aplicación de ingeniería inversa se realiza con los siguientes objetivos:

- Intentar descubrir la lógica y el código de la aplicación. Esto nos ayudará a modificar el código de la aplicación, si es necesario.
- Identificar errores en la implementación o diseño y encontrar posibles métodos para explotarlos.
- Buscar claves secretas, como contraseñas, claves de cifrado y otros datos confidenciales en el código de la aplicación.

Para realizar ingeniería inversa en una aplicación Android, necesitamos utilizar las siguientes herramientas, para tener nuestro entorno preparado:

- **Java JDK:** Son herramientas de desarrollo para Java.
- **SDK Platform Tools:** Incluye herramientas que interactúan con la plataforma Android, como adb, fastboot y systrace.
- **JD-GUI:** Esta herramienta nos permite convertir código de bytes de Java a código fuente de Java: <http://java-decompiler.github.io/>
- **ApkTool:** Nos servirá para descompilar y volver a compilar nuestro APK, con las modificaciones que necesitemos:  
<http://code.google.com/p/android-apktool>
- **Dex2Jar:** Nos permite convertir un archivo DEX a JAR. Posibilita convertir código de bytes de DEX en código de bytes de Java:  
<http://code.google.com/p/dex2jar>
- **Dedexer:** Permite convertir los bytecodes de DEX en el ensamblador de DEX: <http://dedexer.sourceforge.net>
- **Jadx:** Nos permitirá ver el código fuente de un archivo JAR:  
<https://github.com/skylot/jadx>

- **Cmder:** Emulador de consola que permite ejecutar los comandos de Linux desde Windows: <https://cmder.net>
- **Apk Signer:** Nos permitirá firmar nuestra APK.

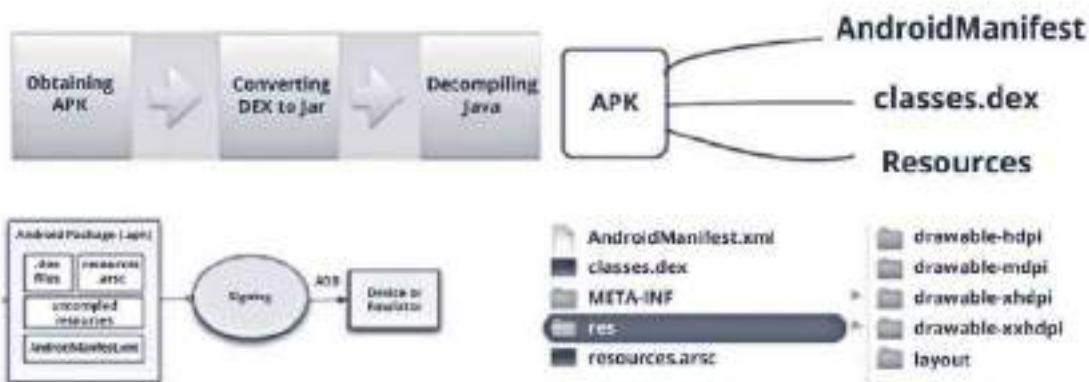


Figura 4.10 Proceso de ingeniería inversa sobre una APK.

#### 4.3.1 ADB (Android Debug Bridge)

ADB es una herramienta de línea de comandos muy útil que viene junto con el SDK de Android y le permite comunicarse desde su sistema operativo con el dispositivo Android en términos de transferencias de archivos, instalaciones de aplicaciones, trabajo en la shell del dispositivo, etc. ADB le ofrece una gran flexibilidad al interactuar con el dispositivo. Algunos de los comandos más utilizados que pueden ayudarlo son:

- **adb shell:** Inicia una shell remota en el emulador de destino y puede trabajar en el dispositivo, como si lo estuviera usando físicamente.
- **adb install <apk\_file>:** Instala el archivo APK en el dispositivo.
- **adb push:** Copia un archivo de la máquina a su dispositivo.
- **adb pull:** Copia un archivo del dispositivo a su máquina.
- **adb logcat:** Obtiene logs de aplicaciones instaladas en el dispositivo.

#### 4.3.2 Dex2jar

Se encarga de realizar la conversión del fichero de clases basado en los bytecodes de Dalvik a fichero de clases Java. Permite convertir el formato de ficheros

.dex para Android al formato .class de Java. Se compone de cuatro componentes, principalmente:

- **Dex-reader.** Permite leer ficheros con extensión .dex y .odex (ejecutables Dalvik).
- **Dex-translator.** Realiza el trabajo de conversión. Lee las instrucciones .dex, realiza procesos de optimización y, finalmente, realiza la conversión a ASM.
- **Dex-ir.** Utilizado por el dex-translator, se encarga de representar las instrucciones dex.
- **Dex-tools.** Son unas herramientas para trabajar con los ficheros .class.

Algunos enlaces interesantes sobre Dalvik y los OPCodes:

- <https://es.wikipedia.org/wiki/Dalvik>
- [http://pallergabor.uw.hu/androidblog/dalvik\\_OPCODES.html](http://pallergabor.uw.hu/androidblog/dalvik_OPCODES.html)
- <http://developer.android.com/reference/dalvik bytecode/Opcodes.html>

Estas son las opciones de que disponemos para ejecutar con dex2jar. Básicamente, habría que pasarle al script como parámetro el fichero APK.

```
C:\Users\User>d2j-dex2jar  
d2j-dex2jar -- convert dex to jar  
usage: d2j-dex2jar [options] <file0> [<file1> ... <fileN>]  
options:  
  -skip-exceptions           skip-exceptions  
  -d,--debug-info            translate debug info  
  -e,--exception-file <file>  detail exception file, default is $current_dir/file  
                               or name1-error.zip  
  -f,--force                  force overwrite  
  -h,--help                   Print this help message  
  -n,--not-handle-exception  not handle any exceptions thrown by dex2jar  
  -nc,--no-code               output .jar file, default is $current_dir/<file>-na  
                               me1-dex2jar.jar  
  -o,--optimize-synchronized optimize-synchronized  
  -p,--print-ir                print ir to System.out  
  -r,--reuse-reg              reuse register while generate java .class file  
  -s                          same with --topological-sort/-ts  
  -ts,--topological-sort      sort block by topological, that will generate more
```

Figura 4.11 Opciones de ejecución del comando dex2jar.

```
/path/to/dex2jar/d2j-dex2jar.sh application.apk
```

#### 4.3.3 JD-GUI

Java se ejecuta sobre una máquina virtual; se puede ejecutar el mismo código en cualquier ordenador que implemente la máquina virtual de Java. Esto tiene muchas ventajas a la hora de exportar nuestra aplicación. Sin embargo, esto también nos permite poder extraer el código de forma sencilla si nuestra aplicación no se encuentra ofuscada. Para realizar esta acción, existen programas que efectúan esta conversión y nos muestran el código Java original.

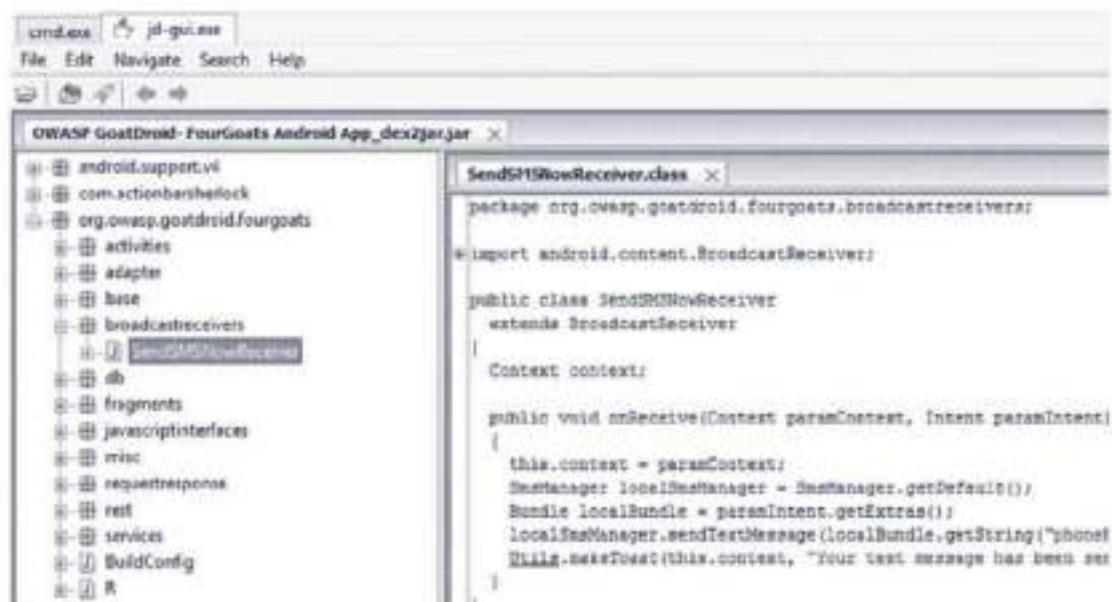


Figura 4.12 Java Decompiler de un fichero .jar.

#### 4.3.4 jadx - Dex to Java decompiler

El código generado para las aplicaciones Android no es exactamente "Java Bytecode"; por eso, los ejecutables para Android no contienen ficheros .class, sino ficheros con extensión .dex, que significa **Dalvik Executable**. De igual forma, estos ficheros .dex se pueden "descompilar" y lo que veremos posteriormente será un lenguaje de bajo nivel (Dalvik Bytecode).

El fichero **classes.dex** contiene el byte code de la aplicación Dalvik (la máquina virtual que ejecuta las aplicaciones de Android). Los archivos de origen de Java se compilan en este código de bytes que la máquina virtual de Dalvik ejecutará finalmente.

La máquina virtual Dalvik dispone de una arquitectura basada en registros y utiliza una herramienta llamada dx para convertir algunos ficheros .class en

ficheros con extensin .dex. Esta herramienta realiza diversas optimizaciones a la hora de transformar los ficheros a .dex, entre las que podemos destacar:

- En las llamadas a métodos virtuales, reemplaza el índice del método por un índice de vtable.
  - Reemplaza llamadas a métodos, como `string.length()`, por métodos inline.
  - Elimina métodos vacíos.
  - Añade datos precalculados.

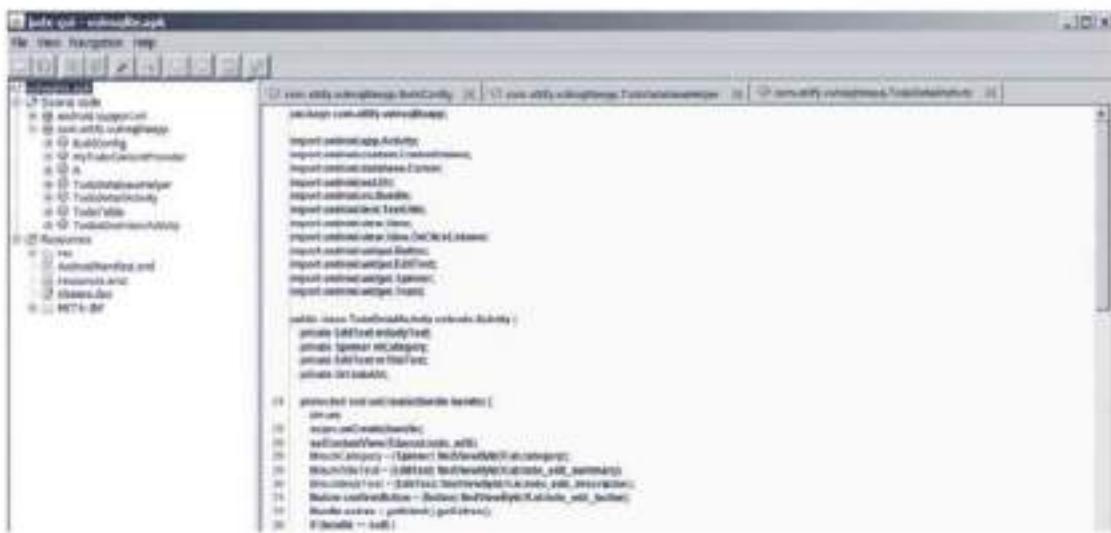


Figura 4.13 Java Decompile de un fichero .jar.

Una de las características de jadx <https://github.com/skylot/jadx> estriba en que posibilita la búsqueda de cadenas y simbolos que le permitirá buscar direcciones url, cadenas, métodos y cualquier cosa que desee encontrar dentro de la base de código de la aplicación.

Algunos de estos decompiladores de Java se pueden encontrar online en la página <http://www.javade.compilers.com>



Figura 4.14 Servicio Java Decompiler online.

Un proyecto interesante surgido a raíz de la herramienta jadx es un **módulo de python que te permite interactuar, de forma programática, con los APK desde python**: <https://github.com/romainthomas/pyjadex>

Pyjadex se encuentra disponible en el repositorio oficial de python y se puede instalar con el comando **pip install pyjadex**. Una vez instalado, podemos desarrollar un script que nos aporte más información sobre el contenido de un APK; por ejemplo, extraer las clases de forma programática:

```
import pyjadex

jadex = pyjadex.Jadx()
app = jadex.load("my.apk")

for cls in app.classes:
    print(cls.code)
```

Podríamos utilizar la clase **pyjadex.JadxDecompiler** para acceder al código de una determinada clase. En este ejemplo, el objeto decompiler representa una instancia de dicha clase:

```
import pyjadex

jadex = pyjadex.Jadx()
decompiler = jadex.load("com.example.apk")
clazz = decompiler.get_class("com.example.MyClass")
print(clazz.code)
```

#### 4.3.5 Apktool

Apktool es un kit de herramientas para trabajar con ficheros APK, con el que podemos generar el código de bajo nivel. Puede decodificar recursos de forma casi original y reconstruirlos, incluso después de hacer algunas modificaciones. Hace posible depurar el código smali paso a paso. También facilita el trabajo con la aplicación debido a la estructura de archivos y la automatización de algunas tareas repetitivas como la creación de APK.

En la página oficial <https://ibotpeaches.github.io/Apktool/install>, se explica el proceso de instalación. Los pasos de instalación se pueden resumir en los siguientes puntos:

- Una vez que descargamos el script, debemos renombrarlo como apktool.bat.
- Descargamos apktool.
- Renombramos el archivo descargado a apktool.jar.
- Si estamos en Windows, copiamos los ficheros apktool.bat y apktool.jar en la carpeta de c:\windows.
- También podemos guardarlos en la carpeta que queremos, C:\Program Files\apktool, y añadirla al path de Windows.
- Por último, validamos que está bien configurada la variable de entorno y ejecutamos el comando apktool en la consola.

```
+ apktool
Apktool v2.0.0-RC2 - a tool for reengineering Android apk files
with smali v2.0.3 and baksmali v2.0.3
Copyright 2010 Ryszard Wi?niewski <brut.alll@gmail.com>
Updated by Connor Tumbleson <connor.tumbleson@gmail.com>

usage: apktool
    -advance,--advanced      prints advance information.
    -version,--version       prints the version then exits
usage: apktool if|install-framework [options] <framework.apk>
    -p,--frame-path <dir>   Stores framework files into <dir>.
    -t,--tag <tag>          Tag frameworks using <tag>.
usage: apktool d[decode] [options] <file.apk>
    -f,--Force              Force delete destination directory.
    -o,--output <dir>        The name of folder that gets written. Default is apk.out
    -p,--frame-path <dir>   Uses framework files located in <dir>.
    -r,--no-res              Do not decode resources.
    -s,--no-src              Do not decode sources.
    -t,--frame-tag <tag>    Uses framework files tagged by <tag>.
usage: apktool b[uild] [options] <app_path>
    -f,--force-all           Skip changes detection and build all files.
    -o,--output <dir>        The name of apk that gets written. Default is dist/name.apk
    -p,--frame-path <dir>   Uses framework files located in <dir>.

For additional info, see: http://code.google.com/p/android-apktool/
For smali/baksmali info, see: http://code.google.com/p/smali/
```

Figura 4.15 Ejecución de apktool.

Para desensamblar una aplicación, utilizamos la opción d[decode]:

- **d[decode] [OPTS] <file.apk> [<dir>]**. Opción que desensambla la aplicación file.apk en el directorio dir:
  - s, --no-src. No desensambla el código fuente.
  - r, --no-res. No desensambla los recursos.
  - d, --debug. Desensambla en modo debug.
  - f, --force. Fuerza a eliminar el directorio de destino si existe.

- t <tag>, --frame-tag<tag>. Utiliza los ficheros framework etiquetados por <tag>.
- keep-broken-res. Se utiliza cuando obtenemos algún error con los recursos de una aplicación, pero queremos seguir igualmente con el proceso.

```
I: apktool d "OWASP GoatDroid- FourGoats Android App.apk"
I: Using Apktool 2.0.0-RC1 on OWASP GoatDroid- FourGoats Android App.apk
I: Loading resource table...
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\Aditya\apktool\framework\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...
```

Figura 4.16 Ejecución de apktool con la opción decode.

**Para ensamblar una aplicación, utilizamos la opción b[build]:**

- b[uild] [OPTS] [<app\_path>] [<out\_files>]. Ensambla una aplicación a partir del código fuente que encuentra en <app\_path>. Si se omite alguno de los dos directorios, tomará el de por defecto:
  - f, --force-all. Salta la detección de cambios en los ficheros y ensambla todos los ficheros.
  - d, --debug. Ensambla en modo debug.

APKTool es capaz de descompilar el archivo AndroidManifest a su formato XML original y también convertirá el archivo **classes.dex** en un lenguaje intermedio llamado **SMALI**, un lenguaje similar a assembler ASM utilizado para representar los códigos de operación de la máquina virtual de Dalvik como un lenguaje legible por humanos.

Al final del proceso de ingeniería inversa, obtendremos una estructura de carpetas como la de la figura 4.17.

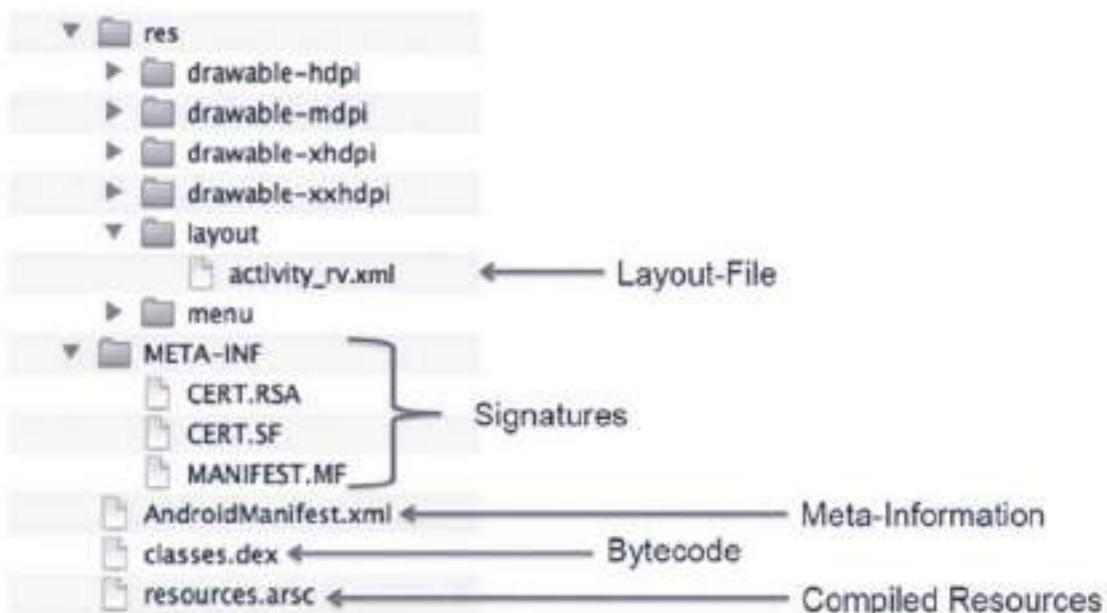


Figura 4.17 Estructuras de carpetas y ficheros de un APK.

#### 4.3.6 Código smali y Mobyizer

Se trata de un ensamblador/desensamblador de ficheros APK que nos servirá para generar el código fuente en lenguaje de bajo nivel tipo ensamblador. Se puede descargar de aquí: <http://code.google.com/p/smali>

```
.super Ljava/lang/Object;
.method public static main([Ljava/lang/String;)V
    .registers 2
    sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
    const-string   v1, "Hello World!"
    invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
    return-void
.end method
```

**Mobyizer** es un script en python <https://github.com/nkpanda/Android-Testing> para realizar un análisis estático de cualquier aplicación de Android o archivo

.apk. Permite automatizar el proceso de desensamblar la aplicación para obtener el código smali.

The screenshot shows the Mobilyzer interface. At the top, there is assembly code with labels like 'ARM64', 'ARM', and 'ARM64'. Below the assembly code, there is a section titled 'credits' listing names: Sudhamshu Chauhan, Rutan Kumar Pandey, Shubham Mittal. Underneath that, it says 'Enter apk filename: one.apk'. The main area contains a large block of command-line output from the tool:

```
I: Test started for one.apk
I: Using Apktool 2.4.0 on one.apk
I: Loading resource table...
I: Decoding androidManifest.xml with resources...
S: WARNING: Could not write to (C:\Users\jortegac\AppData\Local\apktool\framework), using C:\Users\jortegac\ApkTool\Temp\ instead.
S: Please be aware this is a volatile directory and frameworks could go missing, please utilize --framework-path if your storage directory is unavailable.
I: Loading resource table from file: C:\Users\jortegac\AppData\Local\Temp\1.apk
I: Regular manifest package...
I: Decoding File-resources...
I: Decoding values */@*res...
I: Baksmaling classes.dex...
I: Copying assets and lib...
I: Copying unknown files...
```

Figura 4.18 Ejecución de Mobilyzer a partir del fichero APK.

The screenshot shows the search results from the Mobilyzer tool. It lists several entries, each with a file name, string, and line number. The strings are related to the 'AccelListener' class and its methods.

File	String	Line
/one/smali/com/phonegap/AccelListener.smali	api "at Line number 24"	super Lcom/phonegap/api/Plugin;
/one/smali/com/phonegap/AccelListener.smali	api "at Line number 24"	invoke-direct {p0}, Lcom/phonegap/api/Plugin->onInit:D
/one/smali/com/phonegap/AccelListener.smali	api "at Line number 108"	method public execute(Ljava/lang/String;Lorg/json/JSONObject;Ljava/lang/String;)Lcom/phonegap/api/PluginResult;
/one/smali/com/phonegap/AccelListener.smali	api "at Line number 126"	sget-object v5, Lcom/phonegap/api/PluginResult\$Status;-->Lcom/phonegap/api/PluginResult\$Status;
/one/smali/com/phonegap/AccelListener.smali	api "at Line number 126"	sget-object v5, Lcom/phonegap/api/PluginResult\$Status;-->Lcom/phonegap/api/PluginResult\$Status;
/one/smali/com/phonegap/AccelListener.smali	api "at Line number 129"	local v5, "status":Lcom/phonegap/api/PluginResult\$Status;

Figura 4.19 Ejecución de Mobilyzer para obtener el código smali.

La herramienta también puede buscar ciertas cadenas que nos ayudarán a encontrar determinados elementos como bases de datos.

```
logfile.log x
String ' user ' at line number 123
Line:    const-string v11, "userName"

File: ./Goat/smali/org/owasp/goatdroid/fourgoats/base/BaseActivity.smali
String ' user ' at line number 142
Line:    .end local v10      #userName:Ljava/lang/String;

File: ./Goat/smali/org/owasp/goatdroid/fourgoats/db/CheckinDBHelper
$CheckinOpenHelper.smali
String ' sql ' at line number 2
Line:    .super Landroid/database/sqlite/SQLiteOpenHelper;

File: ./Goat/smali/org/owasp/goatdroid/fourgoats/db/CheckinDBHelper
$CheckinOpenHelper.smali
String ' sql ' at line number 38
Line:    invoke-direct {p0, p2, v6, v1, v2}, Landroid/database/sqlite/
SQLiteOpenHelper;-><init>(Landroid/content/Context;Ljava/lang/String;Landroid/database/
sqlite/SQLiteDatabase$CursorFactory;I)V
```

Figura 4.20 Busca de cadenas relacionadas con bases de datos.

#### 4.3.7 Androwarn

Androwarn <https://github.com/maaaaz/androwarn> es un analizador de código estático para aplicaciones Android cuyo objetivo principal es detectar y advertir al usuario sobre posibles comportamientos maliciosos desarrollados por una aplicación. La detección se realiza con el análisis estático del código de bytes Dalvik de la aplicación, representado como código smali (<https://github.com/JesusFreke/smali>), con la biblioteca de androguard.

Este análisis conduce a la generación de un informe, de acuerdo con un nivel de detalle técnico elegido por el usuario. Entre los análisis que realiza, podemos destacar:

- **Análisis estructural y de flujo de datos** en función de diferentes categorías de comportamientos maliciosos.
- **Filtración de datos relacionados con identificadores de telefonía:** IMEI, IMSI, MCC o nombre del operador.
- **Filtración de datos relacionados con la configuración del dispositivo:** Versión del software, estadísticas de uso, configuración del sistema o registros.
- **Fuga de información de geolocalización:** Geolocalización GPS/WiFi.

- **Filtración de datos relacionados con las interfaces de conexión:** Credenciales WiFi o dirección MAC Bluetooth.
- **Uso de servicios de telefonía:** Envío de SMS premium o llamadas telefónicas.
- **Intercepción de flujo de audio/vídeo:** Grabación de llamadas o captura de vídeo.
- **Operaciones relacionadas con el uso de memoria externa:** Acceso a archivos en tarjeta SD.
- **Ejecución de código arbitrario:** Código nativo mediante JNI, comando UNIX o escalada de privilegios.
- **Denegación de servicio:** Desactivación de notificación de eventos, eliminación de archivos, eliminación de procesos, desactivación del teclado virtual o apagado/reinicio de terminal.

Para realizar el análisis, ejecutamos el script en Python para lo que pasamos como parámetro el fichero APK:

```
$ python androwarn.py -h

usage: androwarn [-h] -i INPUT [-o OUTPUT] [-v {1,2,3}] [-r {txt,html,json}]
                  [-d]
                  [-L {debug,info,warn,error,critical,DEBUG,INFO,WARN,ERROR,CRITICAL}]
                  [-w]

version: 1.4

optional arguments:
  -h, --help      show this help message and exit
  -i INPUT, --input INPUT
                  APK file to analyze
  -o OUTPUT, --output OUTPUT
                  Output report file (default
                  "./<apk_package_name>_<timestamp>.<report_type>")
  -v {1,2,3}, --verbose {1,2,3}
                  Verbosity level (ESSENTIAL 1, ADVANCED 2, EXPERT 3)
                  (default 1)
  -r {txt,html,json}, --report {txt,html,json}
                  Report type (default "html")
  -d, --display-report Display analysis results to stdout
  -L {debug,info,warn,error,critical,DEBUG,INFO,WARN,ERROR,CRITICAL}, --log-level {de-
bug,info,warn,error,critical,DEBUG,INFO,WARN,ERROR,CRITICAL}
                  Log level (default "ERROR")
  -w, --with-playstore-lookup
                  Enable online lookups on Google Play
```

De forma predeterminada, genera un informe HTML, opción que podemos cambiar con el parámetro `-r`, para indicarle si queremos exportar los resultados en un fichero .html, .txt o .json.

```
$ python androwarn.py -i my_application_to_be_analyzed.apk -r html -v 3  
$ python androwarn.py -i my_application_to_be_analyzed.apk -r txt -v 3  
$ python androwarn.py -i my_application_to_be_analyzed.apk -r json -v 3
```

#### 4.3.8 Mobile Security Framework (MobSF)

Mobile Security Framework (MobSF) es una aplicación capaz de realizar análisis estático, dinámico y de malware. Se puede utilizar para el análisis de seguridad efectivo y rápido de las aplicaciones móviles de Android, iOS y Windows, y admite tanto binarios (APK, IPA y APPX) como código fuente comprimido. MobSF puede realizar pruebas dinámicas de aplicaciones en tiempo de ejecución para aplicaciones de Android y dispone de un analizador de seguridad específico de API web.

El repositorio oficial de Docker ofrece una imagen de Docker donde podemos encontrar la aplicación preinstalada:

<https://hub.docker.com/r/opensecurity/mobile-security-framework-mobsf/>

Si disponemos de Docker instalado en nuestra máquina, bastaría con ejecutar el comando `docker pull`, para descargarnos la imagen, y el comando `docker run`, para ejecutar dicha imagen:

```
docker pull opensecurity/mobile-security-framework-mobsf  
docker run -it -p 8000:8000 opensecurity/mobile-security-framework-mobsf:latest
```

El código fuente se puede descargar directamente desde el repositorio <https://github.com/MobSF/Mobile-Security-Framework-MobSF>, donde podemos ver que la herramienta está desarrollada con el framework Django Python, como se observa en el fichero de requirements.txt que podemos encontrar en el repositorio.

Puede instalar las dependencias usando el siguiente comando de python a partir del fichero de requirements:

```
python3 -m pip install -r requirements.txt
```

Una vez que se haya completado la instalación, pruebe los ajustes de configuración con el comando **python3 manage.py test**.

Para aplicar la configuración de las tablas, habría que migrar la instalación actual de la aplicación con el comando **python3 manage.py migrate**.

```
[INFO] 01/Apr/2019 14:05:24 - Mobile Security Framework v1.0.7 Beta
REST API Key: 53641df4df06bad3318550975cc5cbe6d91d57a719fe5b325923ee3aa606f76e
[INFO] 01/Apr/2019 14:05:24 - OS: Windows
[INFO] 01/Apr/2019 14:05:24 - Platform: windows-10-10.0.15063-sp0
[INFO] 01/Apr/2019 14:05:24 - MobsF Basic Environment check
[INFO] 01/Apr/2019 14:05:24 - Checking for update...
[INFO] 01/Apr/2019 14:05:25 - No updates available.

operations to perform:
  Apply all migrations: auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0001_initial... OK
  Applying auth.0002.Alter_permission_name_max_length... OK
  Applying auth.0003_Alter_user_email_max_Length... OK
  Applying auth.0004_Alter_user_username_opts... OK
  Applying auth.0005_Alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_Alter_validators_add_error_messages... OK
  Applying auth.0008_Alter_user_username_max_length... OK
  Applying auth.0009_Alter_user_last_name_max_length... OK
  Applying sessions.0001_initial... OK
```

Figura 4.21 Instalación de Mobile Security Framework.

Ejecute la aplicación utilizando el comando **python manage.py runserver <direccion\_ip>:<número\_puerto>**, como se muestra en la captura de pantalla de la figura 4.22.

```
Performing system checks...

[INFO] 01/Apr/2019 14:21:55 - Mobile Security Framework v1.0.7 Beta
REST API Key: 53641df4df06bad3318550975cc5cbe6d91d57a719fe5b325923ee3aa606f76e
[INFO] 01/Apr/2019 14:21:55 - OS: Windows
[INFO] 01/Apr/2019 14:21:55 - Platform: windows-10-10.0.15063-sp0
[INFO] 01/Apr/2019 14:21:55 - MobsF Basic Environment check
[INFO] 01/Apr/2019 14:21:55 - Checking for update...
[INFO] 01/Apr/2019 14:21:55 - No updates available.

System check identified no issues (0 silenced).
April 01, 2019 - 14:21:56
Django version 2.1.7, using settings 'MOBSF.settings'.
```

Figura 4.22 Ejecución del servidor de Mobile Security Framework.

El resultado del análisis proporcionará toda la información de configuración de la aplicación, como actividades, servicios, content providers, etc. En ocasiones, esta información de configuración proporciona credenciales codificadas en base 64 o claves de API que pueden utilizarse en otros servicios.



Figura 4.23 Resultados del análisis de un APK.

Para más información acerca de la instalación y configuración, consulte la documentación: <https://github.com/MobSF/Mobile-Security-Framework-MobSF/wiki/1.-Documentation>

#### 4.3.9 ClassyShark

ClassyShark <https://github.com/google/android-classyshark> es una herramienta de inspección binaria para aplicaciones Android que permite navegar de forma confiable en cualquier ejecutable de Android y mostrar información importante, como las interfaces de clase y las dependencias. ClassyShark admite varios formatos, incluidas bibliotecas (.dex, .aar o .so), ejecutables (.apk, .jar o .class) y los XML binarios de Android, como el AndroidManifest.



Figura 4.24 Visualización del AndroidManifest.xml.



Figura 4.25 Estadísticas sobre el número de métodos.

#### 4.3.10 Drozer

Drozer <https://labs.mwrinfosecurity.com/tools/drozer> es una herramienta que permite evaluar la seguridad de aplicaciones para Android a nivel de componentes, llamadas a red. Drozer permite automatizar la presencia de elementos que podemos encontrar en aplicaciones Android, como actividades, servicios y proveedores de contenidos exportados. Además, prueba las aplicaciones para detectar debilidades comunes, como la inyección de SQL, las ID de usuario compartidas o dejar habilitado el indicador de depuración.

En la página de github <https://github.com/mwrlabs/drozer> podemos encontrar las instrucciones de instalación. Básicamente, necesitamos tener instalado Python, el JDK de Java, además de las utilidades de Android, como el Android Debug Bridge.

Drozer se compone de una serie de módulos desarrollados en Python encargados de comunicarse con el dispositivo Android a través del runtime de Java mediante un agente que actúa como servidor, que se instala en el dispositivo.

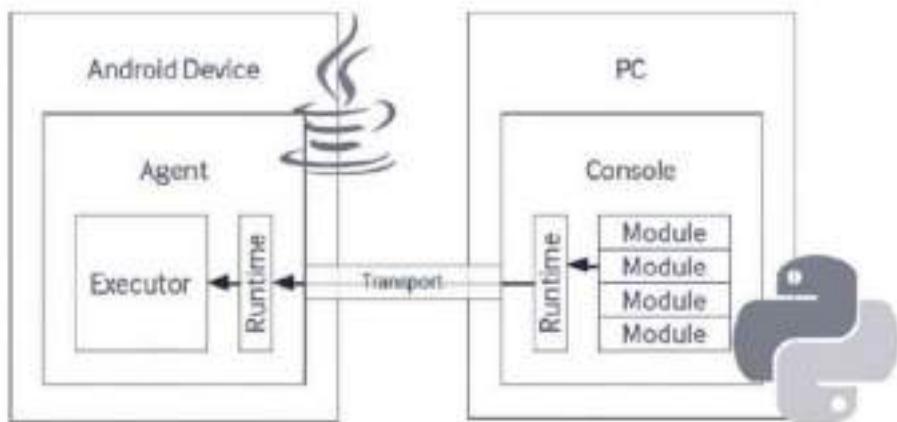


Figura 4.26 Arquitectura de drozer en forma de módulos.

Para ejecutarlo, primero tenemos que instalar el agente de drozer en el dispositivo físico o emulador, abrir la aplicación drozer en el emulador en ejecución y hacer clic en el botón de la parte inferior de la aplicación, que iniciará un servidor incorporado escuchando en el puerto 31415.



Figura 4.27 Ejecución del agente de drozer.

De forma predeterminada, el servidor está escuchando en el número de puerto 31415, por lo que, para enviar todos los comandos del cliente drozer al servidor drozer, usaremos Android Debug Bridge (ADB) para reenviar las conexiones. Para ello, lo podemos hacer a través del comando `adb forward tcp:31415 tcp:31415`. Posteriormente, lanzamos el comando `drozer console connect`, con el fin de abrir el cliente de drozer. Con el **comando list en la consola de drozer**, mostraremos una lista de todos los módulos que vinieron preinstalados con Drozer.

```
dz> list
app.activity.forintent      Find activities that can handle the given intent
app.activity.info           Gets information about exported activities.
app.activity.start          Start an Activity
app.broadcast.info          Get information about broadcast receivers
app.broadcast.send          Send broadcast using an intent
app.package.attacksurface   Get attack surface of package
app.package.backup           Lists packages that use the backup API (returns true
                            on FLAG_ALLOW_BACKUP)
```

Figura 4.28 Consola de drozer con la ejecución del comando list.

Drozer tiene un módulo llamado **app.package.manifest**, que obtiene el archivo androidmanifest.xml de la aplicación y se muestra en la consola de drozer. Cada módulo requiere diferentes opciones. Si desea ver las diferentes opciones para un módulo en particular, use ejecutar seguido del nombre del módulo seguido de -h.

```
dz> run app.activity.start -h
usage: run app.activity.start [-h] [--action ACTION] [--category CATEGORY [CATEGORY ...]]
                               [--component PACKAGE COMPONENT] [--data-uri DATA_URI]
                               [--extra TYPE KEY VALUE] [--flags FLAGS [FLAGS ...]]
                               [--mimetype MIMETYPE]

Starts an Activity using the formulated intent.

Examples:
Start the Browser with an explicit intent:

dz> run app.activity.start
      --component com.android.browser
          com.android.browser.BrowserActivity
      --flags ACTIVITY_NEW_TASK

If no flags are specified, drozer will add the ACTIVITY_NEW_TASK flag. To launch an activity with no flags:

dz> run app.activity.start
      --component com.android.browser
          com.android.browser.BrowserActivity
      --flags 0x0

Starting the Browser with an implicit intent:
```

Figura 4.29 Ejecución de una actividad desde la consola de drozer.

Drozer permite llamar a actividades especificando acciones y parámetros adicionales.

```
optional arguments:
-h, --help
--action ACTION      specify the action to include in the Intent
--category CATEGORY [CATEGORY ...]
                     specify the category to include in the Intent
--component PACKAGE COMPONENT
                     specify the component name to include in the Intent
--data-uri DATA_URI  specify a Uri to attach as data in the Intent
--extra TYPE KEY VALUE
                     add an field to the Intent's extras bundle
--flags FLAGS [FLAGS ...]
                     specify one-or-more flags to include in the Intent
--mimetype MIMETYPE  specify the MIME type to send in the Intent
```

Figura 4.30 Parámetros para llamar a una actividad.

Podemos ver también los servicios que tiene exportados una aplicación con el módulo **app.service.info**.

```
dz> run app.service.info --package org.owasp.goatdroid.fourgoats
Package: org.owasp.goatdroid.fourgoats
    org.owasp.goatdroid.fourgoats.services.LocationService
    Permission: null
```

Figura 4.31 Obtener servicios exportados.

De la anterior captura, podemos deducir que la aplicación tiene un servicio llamado **org.owasp.fourgoats.goatdroid.LocationService**, que se exporta y no necesita ningún permiso. Por lo tanto, cualquier aplicación que se encuentre instalada en el dispositivo con la aplicación FourGoats accederá a la ubicación del dispositivo.

También podemos ver los componentes a nivel de servicios, content providers y actividades que tiene exportados la aplicación con el módulo **app.package.attacksurface**. Igualmente, observaremos si la aplicación es debuggable o no.

Durante las evaluaciones de seguridad de la aplicación, es posible que necesite obtener la superficie de ataque de una determinada aplicación. **La superficie de ataque para una aplicación se determina por el número de componentes exportados:**

```
dz> run app.provider.attacksurface <nombre_paquete>
```

```
dz> run app.package.attacksurface com.mwr.example.sieve
Attack Surface:
  3 activities exported
  0 broadcast receivers exported
  2 content providers exported
  2 services exported
  is debuggable
```

Figura 4.32 Obtener la superficie de ataque de un paquete.

Si utilizamos el módulo **app.provider.finduri** y le pasamos por parámetro el nombre del paquete, podemos ver algunas de las URI expuestas por el proveedor de contenido a las que se puede acceder mediante otras aplicaciones instaladas en el mismo dispositivo:

```
dz> run app.provider.finduri <nombre_paquete>
```

```
dz> run app.provider.finduri com.mwr.example.sieve  
Scanning com.mwr.example.sieve...  
content://com.mwr.example.sieve.DBContentProvider/  
content://com.mwr.example.sieve.FileBackupProvider/  
content://com.mwr.example.sieve.DBContentProvider/  
content://com.mwr.example.sieve.DBContentProvider/Passwords/  
content://com.mwr.example.sieve.DBContentProvider/Keys/  
content://com.mwr.example.sieve.FileBackupProvider/  
content://com.mwr.example.sieve.DBContentProvider/Passwords/  
content://com.mwr.example.sieve.DBContentProvider/Keys
```

Figura 4.33 Obtener las URI expuestas de un paquete.

app.provider.query	Query a content provider
app.provider.read	Read from a content provider that supports files
app.provider.update	Update a record in a content provider
app.service.info	Get information about exported services
app.service.send	Send a message to a service, and display the reply
app.service.start	Start Service
app.service.stop	Stop Service
auxiliary.webcontentresolver	Start a web service interface to content providers.
exploit.pilfer.general.apnprovider	Reads APN content provider
exploit.pilfer.general:settingsprovider	Reads Settings content provider
information.datetime	Print Date/Time
information.deviceinfo	Get verbose device information

Figura 4.34 Comandos disponibles.

Podemos usar el comando de ayuda con cualquiera de los módulos mencionados anteriormente para obtener más información sobre la funcionalidad de un módulo en particular.

```
dz> run app.package.info --help
usage: run app.package.info [-h] [-a PACKAGE] [-d DEFINES_PERMISSION] [-f FILTER] [
-g GID]
                           [-p PERMISSION] [-u UID] [-i]

List all installed packages on the device with optional filters. Specify optional
keywords to search for in the package information, or granted permissions.

optional arguments:
  -h, --help
  -a PACKAGE, --package PACKAGE
                        the identifier of the package to inspect
  -d DEFINES_PERMISSION, --defines-permission DEFINES_PERMISSION
                        filter by the permissions a package defines
  -f FILTER, --filter FILTER
                        keyword filter conditions
  -g GID, --gid GID    Filter packages by GID
  -p PERMISSION, --permission PERMISSION
                        permission filter conditions
  -u UID, --uid UID    filter packages by UID
  -i, --show-intent-filters
                        show intent filters
```

Figura 4.35 Obtener información sobre un comando.

También podemos comprobar si existen vulnerabilidades basadas en la inyección en base de datos usando el módulo `scanner.provider.injection`.

```
dz> run scanner.provider.injection -a com.attify.vulnsqliteapp
Scanning com.attify.vulnsqliteapp...
Not Vulnerable:
  content://com.attify.vulnsqliteapp.contentprovider/
  content://com.attify.vulnsqliteapp.contentprovider

Injection in Projection:
  No vulnerabilities found.

Injection in Selection:
  content://com.attify.vulnsqliteapp.contentprovider/todos
  content://com.attify.vulnsqliteapp.contentprovider/todos/
```

Figura 4.36 Comprobar vulnerabilidades en proveedores de contenido.

Podríamos realizar consultas SQL sobre estos proveedores de contenido junto con un argumento de selección, como `1 = 1`, para acceder a los datos.

```

dz> run app.provider.query content://com.attify.vulnsqliteapp.contentprovider/todos/ --selection "_id=1"
| _id | category | summary | description |
| 1 | Urgent | Message | Description |
dz> run app.provider.query content://com.attify.vulnsqliteapp.contentprovider/todos/ --selection "_id=2"
| _id | category | summary | description |
| 2 | urgent | Financial Summary | Submit Annual Report |
dz> run app.provider.query content://com.attify.vulnsqliteapp.contentprovider/todos/ --selection "1+"
| _id | category | summary | description |
| 1 | Urgent | Message | Description |
| 2 | urgent | Financial Summary | Submit Annual Report |

```

Figura 4.37 Consultas sobre los proveedores de contenido vulnerables.

#### 4.3.11 QARK

QARK <https://github.com/linkedin/qark> es una herramienta de análisis de código estático, diseñada para reconocer posibles vulnerabilidades de seguridad para las aplicaciones de Android basadas en Java, y proporciona funciones útiles para mejorar las pruebas de seguridad manuales de las aplicaciones de Android.

QARK muestra a los desarrolladores información sobre los riesgos potenciales relacionados con la seguridad de las aplicaciones de Android, al proporcionar descripciones claras de los problemas y enlaces a fuentes de referencia autorizadas. QARK también intenta proporcionar comandos ADB (Android Debug Bridge) generados dinámicamente para ayudar en la validación de las vulnerabilidades potenciales que detecta. Entre las principales características, podemos destacar:

- Una interfaz de línea de comandos interactiva.
- Integración con herramientas dentro del ciclo SDLC (ciclo de vida del desarrollo de software).
- Generación de informes y seguimiento del histórico.
- Capacidad de inspeccionar fuentes Java sin compilar o archivos APK compilados.
- Análisis del `AndroidManifest.xml` para localizar problemas potenciales.
- Validación automática mediante comandos ADB generados dinámicamente.
- Combinación de los resultados de múltiples descompiladores para recrear el código fuente original, mejora lo que un descompilador lograría por sí mismo.

- Examen de las configuraciones de webview y dotación de archivos HTML con plantillas para la validación de vulnerabilidades.
- Búsqueda de problemas comunes de validación de certificados X.509.
- Búsqueda de vulnerabilidades que se originan dentro de la aplicación, inspeccionando actividades e intents.
- Búsqueda de claves privadas en el código y algoritmos criptográficos que se estén usando de forma insegura.

```

.d88888b.      d88888b.      88888888b.      888      d8P
d88P" "Y88b    d88888b.      888      Y88b      888      d8P
888      888    d88P888      888      888      888      d8P
888      888    d88P 888      888      d88P      888d88K
888      888    d88P  888      88888888P"      8888888b
888  Y88b 888    d88P   888      888 T88b      888      Y88b
Y88b. Y8b88P    d8888888888      888   T88b      888      Y88b
"Y888888"      d88P       888      888   T88b      888      Y88b
                  Y8b

```

```

INFO - Initializing...
INFO - Identified Android SDK installation from a previous run.
INFO - Initializing QARK

```

```

Do you want to examine:
[1] APK
[2] Source

```

```
Enter your choice:1
```

Figura 4.38 Ejecución de QARK.

```

Do you want to examine:
[1] APK
[2] Source

```

```
Enter your choice:1
```

```

Do you want to:
[1] Provide a path to an APK
[2] Pull an existing APK from the device?

```

```
Enter your choice:1
```

```

Please enter the full path to your APK (ex. /foo/bar/pineapple.apk):
Path:../testapp.apk

```

Figura 4.39 Seleccionar la ruta del APK para analizar.

QARK permite identificar problemas que pueden originar potenciales vulnerabilidades debido al hecho de que el valor “android: debuggable” se establece en

true. Una vez que finaliza el análisis del archivo de manifiesto, QARK comienza con la descompilación, que se requiere para el análisis del código fuente.

```
Press ENTER key to begin Static Code Analysis
INFO - Running Static Code Analysis...
INFO - Looking for private key files in project

Crypto issues 32%|#####
Broadcast issues 35%|#####
Webview checks 47%|#####
X.509 Validation 33%|#####
Pending Intents 23%|#####
File Permissions (check 1) 50%|#####
File Permissions (check 2) 0%
```

Figura 4.40 Proceso de análisis de código estático.

Una vez ha finalizado el análisis, se genera un **informe** en formato .html con el resultado del análisis estático.



Figura 4.41 Resultados del proceso de análisis.

QARK ha identificado dos actividades que se exportan, lo que puede suponer un riesgo para la aplicación.

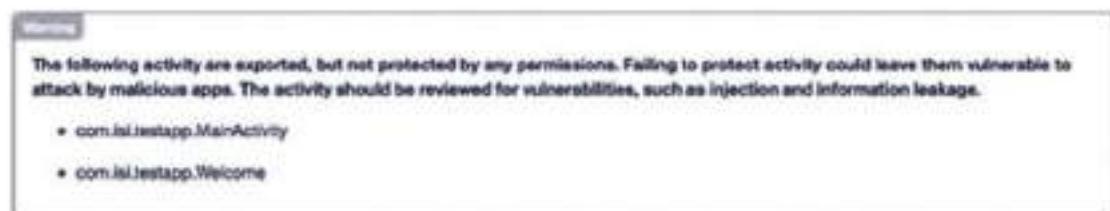


Figura 4.42 Mensaje en modo de aviso indicando una brecha de seguridad.

#### 4.3.12 SanDroid

Sandroid <http://sanddroid.xjtu.edu.cn>, desde el punto de vista de **análisis estático**, permite las siguientes operaciones:

- **Extracción de información básica:** Tamaño de archivo, hash de archivo, nombre del paquete o versión del SDK.
- **Análisis de certificados.**
- **Análisis de permisos:** Extracción de permisos con el objetivo de ver si estos están siendo usados.
- **Análisis de componentes:** Obtención de todos los componentes (actividades, servicios y content providers) y analizar si están siendo exportados.
- **Análisis de características de código:** Comprobación de código nativo y código por reflexión de Java.

SanDroid			
Activities			
Name	Main Activity	Exposed	
org.simplecloud.Main + android.intent.action.MAIN	●	●	
Services			
Name	Exposed		
org.simplecloud.MainService	●		
org.torproject.android.service.TorService + org.torproject.android.service.ITorService + org.torproject.android.service.TOR_SERVICE	●		

Figura 4.43 Análisis estático de actividades y servicios.

Desde el punto de vista del **análisis dinámico**, permite las siguientes operaciones:

- **Registro de datos de red:** Permite capturar todos los datos de red en tiempo de ejecución.
- **Recuperación de peticiones http:** Recupera datos a nivel de peticiones http.
- **Análisis de dominios y direcciones IP:** Permite analizar la información de IP a partir de las url extraídas.
- **Análisis de archivos:** Permite analizar con qué ficheros está trabajando la aplicación.
- **Análisis de llamadas telefónicas y SMS:** Permite registrar los mensajes enviados y las llamadas telefónicas.

#### Broadcast Receivers

Name	Dynamically registered	Exposed
android.support.v4.content.LocalBroadcastReceiver	○	●
android.support.v4.media.TransportMediator\$OnVolumeUpdateReceiver	○	●
org.simplesigner.IDCardService\$ServiceReceiver	○	●
+ android.intent.action.ACTION_EXTERNAL_APPLICATIONS_AVAILABLE		
org.simplesigner.Service\$ServiceReceiver	○	●
+ android.intent.action.BOOT_COMPLETED		

Figura 4.44 Análisis dinámico de actividades y servicios.

#### 4.3.13 Yaazhini

Yaazhini <https://www.vegabird.com/yaazhini> es un explorador de vulnerabilidades para Android que puede escanear un fichero APK para obtener información sobre permisos y servicios, y detectar vulnerabilidades relacionadas con la información que exponen las actividades, los servicios y los content providers.

#	Name	Description
1	android.permission.INTERNET	Allows application to open network sockets.
2	android.permission.WRITE_EXTERNAL_STORAGE	Allows an application to write to external storage. Starting in API level 16, this permission is no longer required to read/write to your application's private directory, introduced by Context.getExternalFilesDir(String) and Context.getExternalFilesDir(String).
3	android.permission.DND_TINT	Allows an application to send DND message.
4	android.permission.VIBRATE	Allows an application to send VIBRATE messages.
5	android.permission.GET_ACCOUNTS	Allows access to the list of accounts in the Accounts service.
6	android.permission.READ_PROFILE	Allows access to the list of accounts in the Accounts service.
7	android.permission.READ_CONTACTS	Allows an application to read the user's contact data.
8	android.permission.READ_PHONE_STATE	Allows read-only access to phone state, including the phone number of the device, current cellular network information, the status of any ongoing calls, and a list of any VoIP accounts registered on the device.

Figura 4.45 Permisos expuestos por la aplicación.

Además, para cada vulnerabilidad, proporciona el nivel de criticidad y la puntuación que mide el impacto de esa vulnerabilidad.

Home	Vulnerabilities	View Code	Permissions	Activities	Receivers	Services
Number	Issue				Risk	Analysis
1	Android Debuggable enabled				Medium	Issue
2	Android backup vulnerability				Medium	Issue
3	Improper Export of Activity com.android.insecurebankv2.LoginActivity				Medium	Issue
4	Improper Export of Activity com.android.insecurebankv2.PostLogin				Medium	Issue
5	Improper Export of Activity com.android.insecurebankv2.OnTransfer				Medium	Issue
6	Improper Export of Activity com.android.insecurebankv2.ViewStatement				Medium	Issue
7	Improper Export of Receiver com.android.insecurebankv2.MyBroadCastReceiver				Medium	Issue
8	Improper Export of Activity com.android.insecurebankv2.ChangePassword				Medium	Issue

Figura 4.46 Vulnerabilidades por nivel de criticidad.

No	Vulnerability Name	Risk	Severity	Cvss score	Occurrences
1	Improper Export of Activity com.android.insecurebankv2.ChangePassword	Medium	Medium	4.8	1
2	Improper Export of Receivers com.android.insecurebankv2.MyBroadCastReceiver	Medium	Medium	4.8	1
3	Improper Export of Activity com.android.insecurebankv2.ViewStatement	Medium	Medium	4.8	1
4	Improper Export of Activity com.android.insecurebankv2.OnTransfer	Medium	Medium	4.8	1
5	Improper Export of Activity com.android.insecurebankv2.PostLogin	Medium	Medium	4.8	1
6	Android backup vulnerability	Medium	Medium	4.8	1
7	Android Debuggable enabled	Medium	Medium	4.8	1

Figura 4.47 Puntuaciones asignadas a cada vulnerabilidad.

#### 4.4 Buenas prácticas de desarrollo seguro en Android

No existen recetas mágicas cuando nuestro objetivo es la seguridad de la aplicación. Más bien, uno debe adoptar un enfoque que abarque una serie de medidas diseñadas para proteger los datos y sistemas de una variedad de métodos de ataque diferentes. Lograr este objetivo significa tomar medidas proactivas antes de que la aplicación suba a producción o se distribuya en el entorno de aplicaciones del cliente. A continuación se pasan a enumerar las **recomendaciones y prácticas de seguridad** que todos los desarrolladores deben seguir:

- Evitar exportar componentes relacionados con proveedores de contenido, también llamados content providers.

- Desde el punto de vista del desarrollador, no abusar de la solicitud de permisos en el **androidmanifest.xml**.
- Usar librerías criptográficas conocidas, utilizando mecanismos de cifrado y seguridad que ofrezca la plataforma para guardar la información que generan las aplicaciones.
- Establecer los permisos apropiados para los ficheros. Si utilizamos **MODE\_WORLD\_WRITEABLE** y **MODE\_WORLD\_READABLE**, al fichero se podrá acceder en modo lectura y escritura desde fuera de la aplicación. En este punto, la mejor opción estriba en usar **MODE\_PRIVATE**. En general, habría que evitar almacenar información sensible en ficheros que sean world readable.
- Evitar guardar datos en áreas de almacenamiento externo y público (principalmente, tarjetas SD).
- Evitar exponer información de log en producción, de forma que la información de log solo debería verse en modo debuggable. A nivel de debug, se recomienda establecer la opción debug en false o eliminarlo en el archivo **AndroidManifest.xml**, debido al hecho de que un modo de depuración se deshabilita por defecto cuando se realiza un release de la aplicación.

#### 4.4.1 Seguridad en **AndroidManifest.xml**

El archivo de manifiesto XML describe qué permisos solicitará la aplicación, qué actividades forman parte de esta, qué servicios se ejecutan en segundo plano sin necesidad de disponer de interfaz de usuario y qué clases son las encargadas de recibir y manejar eventos del sistema; por ejemplo, al arrancar el dispositivo o cuando recibimos un SMS entrante:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.company.appname" platformBuildVersionCode="24" platformBuildVersionName="7.0">
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <application android:allowBackup="true" android:icon="@mipmap/ic_launcher" android:label="@string/app_name"
        android:supportsRtl="true" android:theme="@style/AppTheme">
        <activity android:name="com.company.appname.MainActivity">
            <intent-filter>
```

```
<action android:name="android.intent.action.MAIN"/>
<category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
</application>
</manifest>
```

El fichero XML contiene todas las actividades, intents, receivers y permisos que solicitará la aplicación al ser instalada y que necesitará para el correcto funcionamiento.

La herramienta <https://www.nccgroup.trust/us/our-research/manifest-explorer/> **Manifest Explorer** es una utilidad que le permite revisar este archivo XML con facilidad. Desde el punto de vista de la seguridad, resulta importante verificar que la aplicación sigue el principio de mínimos privilegios y que no se registran permisos que no se necesitan para su funcionamiento.

Otorgar excesivos permisos y privilegios conforma una de las vulnerabilidades más comunes que crea la mayor parte de problemas de privacidad en las aplicaciones móviles. Las aplicaciones que residen en el dispositivo móvil tienen privilegios de acceso excesivos, tales como el acceso a la lista de contactos, la recepción y el envío de mensajes, el acceso a micrófono, cámara, etc.

Al igual que los desarrolladores de aplicaciones deben restringir el uso de permisos a aquellos que sean necesarios para el funcionamiento de la aplicación, es aconsejable que los usuarios comprueben periódicamente la configuración del dispositivo y las aplicaciones, para ver si tienen permisos adicionales que no cuadren con las funcionalidades ofrecidas por estas.

Estas son las cosas que deberíamos analizar en el archivo AndroidManifest.xml:

- Declaraciones en componentes intents, servicios, actividades y proveedores de contenido y los permisos usados por dichos componentes.
- Flag de depuración.
- Versión del SDK que estamos utilizando.

El AndroidManifest también realiza otra serie de acciones adicionales, entre las que podemos destacar:

- Identifica cualquier permiso que requiera la aplicación.
- Declara el nivel mínimo de API que requiere la aplicación para funcionar.

- Declara las características de hardware/software que va a utilizar la aplicación.
- Librerías de la API que la aplicación necesita enlazar para ser utilizadas.

#### 4.4.2 Modelo de permisos en Android

Existen diferentes niveles de protección de permisos:

- **Normal.** Permiso de menor riesgo que el sistema otorga automáticamente en el momento de la instalación y, en condiciones normales, los usuarios no pueden revocar tales permisos.
- **Dangerous.** Permisos de mayor riesgo con acceso a datos privados del usuario. Para utilizar un permiso que se puede considerar peligroso para el usuario, la aplicación debería solicitar a este que otorgue permiso en tiempo de ejecución con la propiedad **protectionLevel = "dangerous"**. Este nivel de protección se dará cuando la aplicación desea proporcionar esta capacidad a otras aplicaciones, pero es necesario advertir al usuario primero de esta característica.
- **Signature.** Se concede en el momento de la instalación, pero solo cuando la aplicación está firmada por el mismo certificado que la aplicación que define el permiso. Solo otorga permisos para las aplicaciones firmadas por el mismo desarrollador.

```
<permission android:name="com.permission_signature"
    android:label="@string/example_perm"
    android:description="@string/example_perm"
    android:icon="@drawable/example_perm"
    android:protectionLevel="signature" />
...
<service android:name=".ServiceExample"
    android:permission="com.permission_signature">
    <intent-filter>...</intent-filter>
</service>
```

También podría utilizar el permiso declarado de la siguiente forma:

```
<uses-permission android:name="com.permission_signature"/>
```

Podríamos definir una función **checkPermission** para que, si se utiliza un permiso definido a nivel de firma, se verifique que la aplicación tiene definido ese permiso:

```
private boolean checkPermission(Context context) {
    String permission = "com.permission_signature";
    int res = context.checkSelfPermission(permission);
    return (res == PackageManager.PERMISSION_GRANTED);
}
```

Un ejemplo de uso de esta función se puede dar cuando estamos implementando un servicio; se utiliza normalmente para el procesamiento en segundo plano. Al igual que en actividades, los servicios pueden ser invocados por aplicaciones externas y deben protegerse por permisos y el atributo de exportación según corresponda.

Un servicio puede tener más de un método, que se invoca desde un componente externo. Es posible definir permisos arbitrarios para cada método y verificar si el paquete que llama tiene el permiso correspondiente con **checkPermissionO**.

Cuando llame a un servicio con datos confidenciales, es posible que quiera validar que el servicio es el correcto. Si conoce el nombre exacto del componente al que está tratando de conectarse, puede especificarlo en la Intención utilizada para conectarse. De lo contrario, puede usar **checkPermission()** para verificar si el paquete que llama tiene el permiso particular necesario para recibir su Intención. Constituye responsabilidad del usuario, en el momento de la instalación, otorgar el permiso a la aplicación.

#### 4.4.3 Asegurando la capa de aplicación

Asegurar la capa de aplicación implica una serie de prácticas; en general:

- Almacenamiento de datos confidenciales de forma segura
- Manejo de autenticación y sesiones correctamente
- Diseño y codificación de la aplicación utilizando correctamente los mecanismos de seguridad y cifrado
- Evitación de la fuga de información en la aplicación en logs o ficheros de configuración

- Aplicación de técnicas de ofuscación de código, para dificultar el proceso de ingeniería inversa

Las aplicaciones móviles brindan la oportunidad de añadir medidas de seguridad, pero también requieren la ingeniería y el código adecuados para evitar vulnerabilidades y brechas de seguridad. Las siguientes prácticas recomendadas cubren estas áreas con más detalle; en algunos casos, mostrando ejemplos de código.

#### **4.4.4 Evitar almacenar datos confidenciales en el dispositivo**

En la medida de lo posible, no deberíamos almacenar datos confidenciales en el dispositivo, tales como claves o credenciales de acceso a un sistema. Hay que tener en cuenta que casi todos los datos almacenados, incluso si están cifrados, podrían verse comprometidos. Para muchas aplicaciones, es funcional y técnicamente posible simplemente no almacenar datos confidenciales.

Si no queda más remedio que almacenar datos confidenciales en el dispositivo, estos deben almacenarse con algún tipo de cifrado. Si bien no es 100 % seguro, se puede añadir una complejidad extra a un posible atacante. Se podrían utilizar algoritmos como PBKDF2, que utilizan una función de derivación de clave con un número muy alto de iteraciones, con el fin de evitar intentos de ataque por fuerza bruta, que requieren mucho tiempo de cálculo.

Las opciones para reducir el almacenamiento de información del usuario incluyen:

- Transmitir y visualizar los datos, pero sin persistir en la memoria.
- Almacenar solo en memoria RAM (borrar datos y ficheros temporales al cerrar la aplicación).
- Cifrar los datos combinando una clave maestra cifrada con una de contraseña requerida, cuando se inicia la aplicación.

Los desarrolladores a menudo pasan por alto algunas de las formas en que se pueden almacenar los datos, incluidos los archivos de registro/depuración, cookies, historial web, caché, archivos y bases de datos SQLite.

En este punto, se recomienda que se tomen medidas para evitar el almacenamiento en caché del historial de url y los datos del usuario para cualquier proceso web, como el registro. El protocolo HTTP admite una directiva de “no

almacenar” en caché las cabeceras de petición y de la respuesta que indica al navegador que no debe almacenar ninguna parte de la respuesta ni de la solicitud que la provocó. Puede encontrar información adicional sobre los encabezados de almacenamiento en caché aquí: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html>

#### 4.4.5 Uso adecuado del componente WebView

El componente WebView crea una instancia del navegador nativo Android que ejecuta código HTML y JavaScript. El uso inadecuado de estos controles puede generar problemas de seguridad web, tales como XSS o inyección de código HTML. Android proporciona una serie de mecanismos para reducir el alcance de estos problemas de seguridad mediante la posibilidad de limitar la capacidad del control WebView a la funcionalidad mínima requerida por la aplicación.

Si la aplicación que se está desarrollando no ejecuta código JavaScript, debe restringirse su uso mediante la no llamada al método `setJavaScriptEnabled()` de la clase WebView. Mediante la no activación de este método, se pueden evitar ataques de inyección de código JavaScript.

Además de prevenir ataques de inyección de código, si se usan formularios web en la vista de WebView donde se introducen los datos del usuario, se recomienda realizar una limpieza de la caché del navegador de forma periódica desde el código mediante la llamada al método `clearCache()` de la clase WebView.

El componente WebView puede presentar una serie de problemas de seguridad y debe implementarse con cuidado; por ello, se aconseja seguir un conjunto de buenas prácticas cuando estemos usando este componente.

- **Es recomendable deshabilitar la compatibilidad con JavaScript** y el complemento si no son necesarios. Están deshabilitados de forma predeterminada, pero conforma una buena práctica establecerlos explícitamente.
- **Deshabilitar el acceso a archivos locales.** Esto restringe el acceso al directorio de recursos de la aplicación, con el objetivo de minimizar un posible ataque desde una página web que busca obtener acceso a archivos que solo son accesibles por la aplicación de forma local.
- **Evitar la carga de contenido de terceros.** Esto es difícil de evitar por completo dentro de una aplicación, pero, desde el punto de vista del desarrollador, si es posible interceptar, inspeccionar y validar la mayoría de las solicitudes iniciadas desde el componente web.

- **Un esquema de lista blanca** también se puede implementar utilizando la clase URI para inspeccionar los componentes de una URI y asegurarse de que coincide con una lista blanca de recursos permitidos.

Se recomienda también deshabilitar la ejecución de código JavaScript dentro del WebView de la app, ya que podría darse el caso de que dicho código realizará acciones maliciosas o no permitidas sobre la aplicación. Para ello, podemos utilizar la instrucción `webview.getSettings().setJavaScriptEnabled(false)`.

Además, se recomienda:

- **Deshabilitar soporte de plugins:** `webview.getSettings().setPluginsEnabled(false);`
- **Desactivar el acceso a ficheros:** `webview.getSettings().setAllowFileAccess(false);`
- **Prevenir la carga de contenido de sitios de terceros,** controlando que el WebView pueda acceder únicamente a los sitios web que la aplicación necesite utilizando, a ser posible, listas blancas a tal efecto
- **Deshabilitar la carga de contenido desde un content provider** instalado en el sistema: `webview.getSettings().setAllowContentAccess(false);`
- **Deshabilitar la carga de ficheros desde url externas a través de JavaScript:** `webview.getSettings().setAllowFileAccessFromFileURLs(false);`
- **Deshabilitar la carga de contenido desde url externas a través de JavaScript:** `webview.getSettings().setAllowUniversalAccessFromFileURLs(false);`

#### 4.4.6 Usar método POST para el envío de datos confidenciales

Para todas aquellas peticiones que trabajen con datos del usuario, es recomendable utilizar el método POST en lugar del método GET; además, como medidas extras podemos configurar un formulario para que se envie un token CSRF.

Si las credenciales se transmiten como parámetros de cadena de consulta mediante GET, a diferencia del cuerpo de una solicitud POST, es probable que se registren en varios lugares; por ejemplo, en el historial del navegador del usuario y en los registros del servidor web. Si un atacante logra comprometer

cualquiera de estos recursos, podría ser capaz de obtener privilegios capturando las credenciales o la información del usuario.

Ya sea POST o GET, se deben usar cookies de sesión temporales. El cifrado de datos utilizando un vector de inicialización distinto de cero y claves de sesión temporales también puede ayudar a prevenir determinados ataques.

#### 4.4.7 Validar los certificados SSL/TLS

Muchas aplicaciones no validan correctamente los certificados SSL, lo que las hace susceptibles a los ataques de *man in the middle*. Los certificados deben ser completamente validados por el cliente y firmados por una CA raíz de confianza.

Resulta importante comprobar HTTPS y la dirección del host. Aunque el código de la aplicación nativa en condiciones normales se halla a prueba de manipulaciones indebidas, el JavaScript del lado del cliente puede usarse para verificar la presencia de HTTPS, la url del host adecuado y la dirección IP del host. Como mejora, los navegadores implementan cabeceras HTTP de seguridad de transporte estricto, lo que hace que el navegador requiera HTTPS para un sitio, una vez que se haya realizado la petición y se haya recibido la cabecera de respuesta.

En muchos casos, no es necesario usar certificados de una autoridad de certificación pública, ya que fueron diseñados para sistemas en los que el cliente no sabía si el sistema al que se estaba conectando era confiable. Dado que las aplicaciones saben exactamente dónde se conectan y confían intrínsecamente en la infraestructura a la que se conectan, es a veces más seguro utilizar una infraestructura de clave pública “privada”, separada de las autoridades de certificación públicas. De esta manera, un atacante requeriría las claves privadas del servidor para realizar un ataque MITM u otros tipos de ataques similares.

Si la aplicación realiza llamadas a servidores de forma segura a través de SSL, siempre será mejor utilizar los certificados ofrecidos por el servidor, en lugar de emplear el comportamiento predeterminado que emplea las firmas de CA de propósito general para validar la autenticidad de la conexión. Como sabe la dirección del servidor a la que se conectarán la aplicación, puede validar que el certificado presentado a su aplicación sea el certificado correcto.

#### 4.4.8 Restricción de uso de la aplicación a determinados dispositivos

Para restringir el uso de una aplicación a determinados usuarios, se puede realizar una comparación por el código IMEI (International Mobile Equipment

Identity) del dispositivo móvil que vaya a ejecutarla. El código IMEI identifica al dispositivo de forma única; por lo tanto, si se realiza una comparación en función de dicho código, se establece con seguridad que el dispositivo que ejecuta la aplicación es el autorizado.

La aplicación podría realizar una comprobación inicial del IMEI del dispositivo que la esté utilizando contra una lista de IMEI asignados por la organización. Si dicho IMEI no se encuentra en la lista, debe finalizarse la ejecución de la aplicación. Para consultar el IMEI de un dispositivo móvil, basta con añadir el permiso **permission.READ\_PHONE\_STATE** y utilizar el método `getDeviceId()`, perteneciente a la clase `TelephonyManager`.

A continuación se muestra un pequeño código de ejemplo que obtiene el código IMEI del dispositivo móvil Android que lo ejecute:

```
TelephonyManager telephonyManager = (TelephonyManager) getSystemService(Context.TELEPHONE_SERVICE);
telephonyManager.getDeviceId();
```

Se recomienda al desarrollador que la lista de IMEI permitidos nunca se almacene en el dispositivo. Tampoco se recomienda que se escriba directamente en código, ya que podría ser fácilmente accedida decompilando dicho código o mediante la realización de un volcado de memoria. Se recomienda almacenar dicha lista en un servidor seguro, perteneciente a la organización, y realizar el protocolo de comunicación mediante un canal seguro utilizando una conexión SSL.

#### 4.4.9 Gestión de logs

Los registros de depuración pueden almacenar datos confidenciales. Es importante asegurarse de que los registros o logs de depuración no estén habilitados en la compilación de la aplicación para desplegarla en producción.

Se deberá deshabilitar o eliminar el registro de información sensible en logs de la aplicación. Para registrar trazas de información en los logs del dispositivo, se utiliza el método Log. Dichos logs pueden ser accesibles por cualquier otra aplicación instalada en el dispositivo, existiendo un riesgo en la confidencialidad de dicha información ante posibles fugas de información. La mejor práctica en este punto estriba en no exponer los datos a través de logcat en producción. Podríamos detectar el modo de compilación con `BuildConfig.DEBUG`:

```
//Código que habilita o deshabilita el registro de logs, dependiendo si se está en modo
//depuración
public static final boolean SHOW_LOG = BuildConfig.DEBUG;

public static void log_tag_msg(final String tag, final String msg) {
    if (SHOW_LOG)
        Log.d(tag, msg);
}
```

#### 4.4.10 Comprobar la conexión de red

Antes de que su aplicación intente conectarse a la red, es importante verificar si hay una conexión de red disponible usando los métodos `getActiveNetworkInfo()` e `isConnected()` de la clase `ConnectivityManager`. Recuerde: El dispositivo puede estar fuera del alcance de una red o el usuario puede haber inhabilitado el acceso a datos móviles y wifi.

```
public void myClickHandler(View view) {
    ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
    if (networkInfo != null && networkInfo.isConnected()) {
        // obtener datos
    } else {
        //mostrar error de conexión
    }
}
```

Para más información sobre estos métodos, puede consultar la documentación oficial: <https://developer.android.com/training/basics/network-ops/managing.html>

#### 4.4.11 Realizar operaciones de red en un hilo separado

Las operaciones de red pueden implicar retrasos impredecibles. Para evitar que esto provoque una experiencia de usuario deficiente, realice siempre las operaciones de red en un subproceso separado de la interfaz de usuario. La clase `AsyncTask` proporciona una de las formas más sencillas de iniciar una nueva tarea desde el subproceso de la interfaz de usuario.

En el siguiente fragmento de código, el método **myClickHandler()** invoca el método `DownloadWebpageTask().execute(stringUrl)`. La clase `DownloadWebpageTask` es una subclase de `AsyncTask` que implementa los siguientes métodos:

- **doInBackground()** ejecuta el método `downloadUrl()`. Pasa la url de la página web como un parámetro. El método **downloadUrl()** recupera y procesa el contenido de la página web. Cuando termina, devuelve una cadena de resultados.
- **onPostExecute()** toma la cadena devuelta y la muestra en la interfaz de usuario.

En este ejemplo, utilizamos `AsyncTask` para crear una tarea fuera del hilo principal de la interfaz de usuario. Dicha tarea toma una url de la cadena y la usa para crear una `HttpURLConnection`. Una vez que la conexión se ha establecido, `AsyncTask` descarga el contenido de la página web como un `InputStream`. Finalmente, el `InputStream` se convierte en una cadena, que se muestra en la interfaz de usuario mediante el método **onPostExecute** de `AsyncTask`:

```
public class HttpExampleActivity extends Activity {  
    private static final String DEBUG_TAG = "HttpExample";  
    private EditText urlText;  
    private TextView textView;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        urlText = (EditText) findViewById(R.id.myUrl);  
        textView = (TextView) findViewById(R.id.myText);  
    }  
  
    public void myClickHandler(View view) {  
        String urlString = urlText.getText().toString();  
        ConnectivityManager connMgr = (ConnectivityManager)  
            getSystemService(Context.CONNECTIVITY_SERVICE);  
        NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();  
        if (networkInfo != null && networkInfo.isConnected()) {  
            new DownloadWebpageTask().execute(stringUrl);  
        } else {  
            textView.setText("No network connection available.");  
        }  
    }  
}
```

```
}

private class DownloadWebpageTask extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String... urls) {
        try {
            return downloadUrl(urls[0]);
        } catch (IOException e) {
            return "Unable to retrieve web page. URL may be invalid.";
        }
    }

    @Override
    protected void onPostExecute(String result) {
        textView.setText(result);
    }
}
```

Para más información sobre estos métodos en la documentación oficial, consulte: <https://developer.android.com/reference/android/os/AsyncTask>

#### 4.4.12 Permisos de localización

Las aplicaciones que utilizan los servicios de ubicación pueden solicitar permisos relacionados con la localización del dispositivo. Android tiene dos permisos de ubicación: **ACCESS\_COARSE\_LOCATION** y **ACCESS\_FINE\_LOCATION**. El permiso que elija controla la precisión de la ubicación actual. Si solicita solo un permiso de ubicación general, los servicios de localización ofuscan la ubicación con una precisión que es, aproximadamente, equivalente a una manzana de la ciudad. La solicitud **ACCESS\_FINE\_LOCATION** implica una solicitud para **ACCESS\_COARSE\_LOCATION**.

```
<uses-permission
    android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

#### 4.4.13 Optimizar el código en Android y memoria caché

Uno de los factores más importantes a la hora de programar aplicaciones de cierto tamaño en Android radica en que trabajamos con recursos con impor-

tantes limitaciones. Al principio, puede que esto no tenga consecuencias, pero, cuando el proyecto va creciendo, esto nos obliga a trabajar en la optimización de nuestro programa.

Existen unos aspectos fundamentales que no debemos olvidar. El más importante es el hecho de que la gestión de memoria puede complicarnos bastante el desarrollo. El entorno en el que se ha querido desarrollar la plataforma Android está condicionado por el dispositivo. Este siempre tendrá unos recursos inferiores a un equipo de sobremesa.

El GC (Garbage Collector) de Java se utiliza para liberar memoria de aquellos objetos sin referencias que no se vayan a usar más. Se puede ejecutar haciendo una llamada desde una línea de código. Pero la llamada al GC no implica necesariamente que se vaya a liberar memoria. Con esa instrucción, estamos ordenando a la máquina virtual que, cuando le sea posible, borre memoria. Por tanto, en la línea siguiente a la petición de llamada, no se deberá suponer que hemos conseguido liberar memoria, solo que, probablemente, se libere memoria en los próximos segundos en que esto sea factible.

De igual forma, reutilizar el código es útil para la aplicación, sobre todo por temas de organización del código. Por ello, la programación orientada a objetos debería definir muy bien nuestras clases y utilizar con acierto los modificadores de variables.

Por tanto, resulta muy importante dividir aquellas partes de código que ejecuten funciones específicas. Tener los métodos y las funciones bien definidos, orientados a tareas concretas, nos ayudará a depurar el código que causa un desbordamiento de memoria. La posibilidad de estos errores, por ejemplo, los memory leaks, errores de software que ocurren cuando un bloque de memoria reservada no se ha liberado de forma correcta, se deberá tener siempre en cuenta.

Hay que considerar también que, aunque la aplicación funcione bien en el emulador, no lo hará igual en el dispositivo. Se debe intentar crear un código eficiente, para garantizar el mejor rendimiento posible en una variedad de dispositivos móviles.

Existe una serie de reglas básicas para sistemas con recursos limitados, entre las que podemos destacar:

- **Optimizar la creación de objetos.** En general, habría que evitar la creación de objetos temporales, si es posible. Menor número de objetos creados significa una menor frecuencia de la recolección de basura, que posee un impacto directo en la experiencia del usuario.

- **Evitar concatenaciones de strings** y utilizar StringBuffer o StringBuilder para tal caso. La concatenación de strings produce, cada vez, un nuevo objeto y, por tanto, mayor consumo de memoria y mayor recolección de basura.
- **Liberar recursos tan pronto como sea posible**, como conexiones de red, a streams o a ficheros. Normalmente, se suele liberar este tipo de recursos dentro de la cláusula finally, para asegurarnos de que los recursos se liberan aun cuando se produzca alguna excepción.
- **Uso de la memoria caché**. Tanto en el almacenamiento interno como en el externo, podremos usar la memoria caché, la cual puede ser borrada si el sistema necesita recursos y, por otro lado, los ficheros serán eliminados automáticamente cuando el usuario desinstale la aplicación. Se ha de tener un control sobre su uso en la memoria interna, ya que puede ralentizar el sistema.
- Para hacer uso de la **caché en memoria interna**, podemos usar el método `getCacheDir()`. Por el contrario, si queremos usar la caché de la memoria externa, podemos usar los métodos `getExternalCacheDir()` para un nivel de API 8 o superior, o `getExternalStorageDirectory()`, para un nivel de API 7 o inferior.

#### 4.4.14 Implementación segura de proveedores de contenido

Android ofrece varias formas de guardar datos en una aplicación. Todo dependerá de la necesidad que tengamos a la hora de guardarlos; es decir, si los datos van a ser privados de la propia aplicación o van a ser públicos accesibles desde otras aplicaciones, de la cantidad de datos que queremos guardar.

En el caso de hacer los datos públicos a otras aplicaciones, deberemos usar un **content provider** que, simplemente, es un componente que nos proporciona lectura/escritura de datos en una aplicación sin ningún tipo de restricción.

ContentProviders es una forma estándar para que las aplicaciones comparten datos utilizando un esquema de direccionamiento URI y un modelo de base de datos relacional. También se pueden utilizar como un medio para acceder a “archivos” a través del esquema URI.

El uso de ContentProviders también permite la delegación a nivel de registro para compartir un registro específico sin compartir la base de datos completa; por ejemplo, puede acceder a un determinado contacto de los almacenados en su dispositivo sin que la aplicación externa tenga acceso a todos los contactos

almacenados en el dispositivo. Cuando la aplicación externa vuelve a la aplicación de origen, la delegación finaliza.

#### 4.4.15 Almacenamiento de preferencias compartidas (SharedPreferences)

La clase SharedPreferences nos permite almacenar y recuperar tipos de datos primitivos asociados a una clave. Estos datos se guardan en un fichero XML ubicado en la ruta “`data/data/'NOMBREdelPAQUETE'/shared_prefs/`”.

Para hacer uso de esta clase, deberemos crear una instancia “SharedPreferences” y usar uno de estos dos métodos:

- **getSharedPreferences(name,mode):** Usaremos este método cuando necesitemos utilizar varios ficheros para guardar los datos; como primer parámetro “name”, indicaremos el nombre del fichero.
- **getPreferences(mode):** Este método lo usaremos cuando necesitemos un solo fichero de preferencias.

Como parámetro “mode”, indicaremos el modo de acceso a esos datos; los más utilizados son:

- **MODE\_PRIVATE:** Es el modo por defecto y quiere decir que el archivo creado solo será accesible por la propia aplicación que lo ha creado.
- **MODE\_WORLD\_READABLE:** En este caso, todas las aplicaciones pueden leer los datos guardados en el fichero, pero solo puede modificarlos la aplicación que lo ha creado.
- **MODE\_WORLD\_WRITABLE:** Todas las aplicaciones pueden consultar y modificar los datos.

No se recomienda tratar los archivos con los permisos de MODE\_WORLD\_READABLE o MODE\_WORLD\_WRITABLE, a menos que sea necesario, ya que cualquier aplicación podría leer o escribir el archivo; por ejemplo, el inspector de código de Android Studio tiene la capacidad de detectar estos casos (como se indica en la figura 4.48).

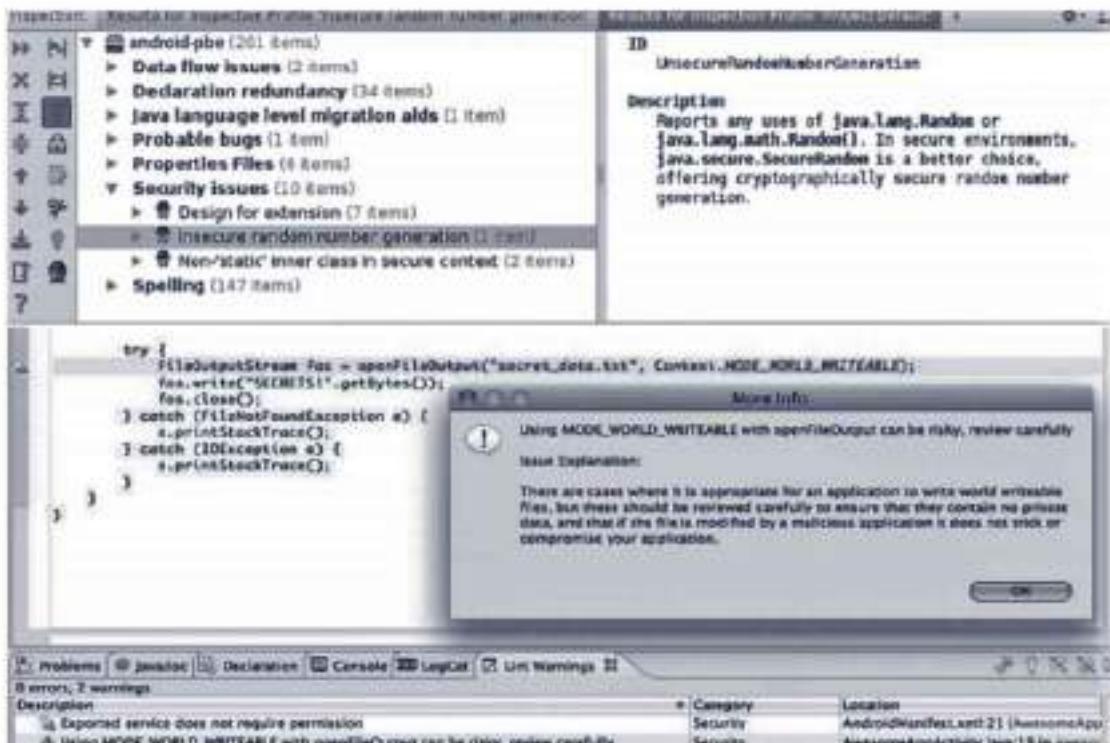


Figura 4.48 Aviso de seguridad al usar MODE\_WORLD\_WRITEABLE.

Para **escribir datos**, tendremos que crear una instancia “**SharedPreferences**” y llamar al método “**edit()**”. Una vez hecho esto, podremos escribir datos con los siguientes métodos:

```
putBoolean(key, value)
putFloat(key, value)
putInt(key, value)
putLong(key, value)
putString(key, value)
```

Como primer parámetro “key”, indicaremos la clave que asociaremos a ese dato guardado y, como segundo parámetro “value”, el valor que queremos guardar. Para terminar, deberemos llamar al método “**commit()**”, para confirmar que estamos guardando datos. Si, por otro lado, queremos leer esos datos guardados, usaremos la instancia “**SharedPreferences**” y uno de los siguientes métodos:

```
getBoolean(key, defaultValue)
getFloat(key, defaultValue)
getInt(key, defValue)
getLong(key, defaultValue)
getString(key, defaultValue)
```

Como primer parámetro “key”, indicaremos la clave que queremos recuperar y, como segundo parámetro “**defaultValue**”, indicaremos un valor por defecto, para devolver en caso de que el dato que queremos recuperar no exista.

En este ejemplo, declaramos una activity, donde el método `onCreate` recupera los datos mediante `getSharedPreferences`, y el método `onSaveData()` crea una instancia de la clase “**SharedPreferences**” y utiliza también la clase “**SharedPreferences.Editor**”, para almacenar los valores:

```
public class MySharedPreferences extends Activity {

    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);

        // RECUPERAR DATOS
        // Creamos la instancia de "SharedPreferences" en MODE_PRIVATE
        SharedPreferences settings = getSharedPreferences("PREFERENCES", 0);
        boolean bol = settings.getBoolean("booleano", false);
        int entero = settings.getInt("booleano", 0);
        String str = settings.getString("cadena", "");
    }

    protected void onSaveData(){

        // GUARDAR DATOS
        // Creamos la instancia de "SharedPreferences"
        // Y también la "SharedPreferences.Editor"
        SharedPreferences settings = getSharedPreferences("PREFERENCES", 0);
        SharedPreferences.Editor editor = settings.edit();
        editor.putBoolean("booleano", true);
        editor.putInt("entero", 10);
        editor.putString("cadena", "string");
        editor.commit();
    }
}
```

Desde Android 6.0 Marshmallow, las preferencias compartidas que guarda la aplicación se configuran automáticamente con la constante **MODE\_PRIVATE**. Esto significa que solo se puede acceder a los datos mediante su propia aplicación. En cualquier caso, resulta una buena idea configurar este valor explícitamente al guardar una preferencia compartida o al guardar un archivo:

```
SharedPreferences.Editor editor = getSharedPreferences("preferenceName", MODE_PRIVATE).edit();
```

```
editor.putString("key", "value");  
editor.commit();
```

```
FileOutputStream fos = openFileOutput(filenameString, Context.MODE_PRIVATE);  
fos.write(data);  
fos.close();
```

#### 4.4.16 Almacenamiento seguro de preferencias

También disponemos de librerías, como por ejemplo la de secure preferences <https://github.com/scottyab/secure-preferences> que ofrece la posibilidad de securizar las preferencias de tal forma que los datos no queden en claro en el fichero XML que se guarda dentro de la carpeta data de la aplicación.

Para cifrar estos datos, utiliza los algoritmos AES 128, CBC y PKCS5 con verificación de integridad, donde las claves se almacenan en forma de hash SHA 256. Tanto las claves como los valores están codificados en base64, antes de almacenarlos en el archivo XML. De forma predeterminada, la clave generada se almacena en el archivo de preferencias de backup y, por lo tanto, el usuario root puede leerla y extraerla.

Para usar esta librería dentro de nuestra aplicación, podríamos añadir la siguiente dependencia en el fichero de gradle:

```
dependencies {  
    implementation 'com.scottyab:secure-preferences-lib:0.1.7'  
}
```

Para usar esta librería, cuando vayamos a declarar nuestro objeto de la clase **SharedPreferences**, ahora realizaremos un new con la nueva clase **SecurePreferences**, y pasaremos por parámetro el contexto de la aplicación:

```
SharedPreferences prefs = new SecurePreferences(context);
```

Si declaramos el objeto a nivel de activity, podemos utilizar el **objeto this**, en lugar de context.

```
public class MainActivity extends Activity {  
    EditText et;  
    TextView tv;  
    Button btn1,btn2;  
    SharedPreferences shared;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        tv = (TextView) findViewById(R.id.tvOut);  
        et = (EditText) findViewById(R.id.editText1);  
        btn1 = (Button) findViewById(R.id.btn1);  
        btn2 = (Button) findViewById(R.id.btn2);  
  
        shared = new SecurePreferences(this);  
    }
```

Figura 4.49 Uso de SharedPreferences dentro de una Activity.

Ahora podemos usar el objeto shared, y crear un objeto Editor para insertar datos en él con el método **putString()**.

```
String input = et.getText().toString();  
Editor editor = shared.edit();  
editor.putString("PASSWORD", input);  
editor.commit();  
Toast.makeText(getApplicationContext(), "Success", Toast.LENGTH_LONG).show();  
et.setText("");
```

Figura 4.50 Creación del objeto editor y uso del método putString.

Para recuperar datos de este objeto, podemos usar el método **getString()**, con los argumentos necesarios:

```
String out = shared.getString("PASSWORD",null);  
tv.setText(out)
```

De esta forma, al visualizar el fichero XML de preferencias, vemos los datos ofuscados.

```
root@generic:/data/data/com.securepreferences.sample/shared_prefs # ls  
com.securepreferences.sample_preferences.xml  
:xml  
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>  
<map>  
    <string name="cgASLY/nymum+yv+S+2itmlSHPFDMElhAT9p0/5DcA">uKJlx6JdM0JaCABUhxETBRvUliCmjJidGLtierH/6s</string>  
</map>  
:xml  
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>  
<map>  
    <string name="a5lqxitsA/Eag2EbNqSA9A">0v08+ILj9keS3fU3r//lRw</string>  
    <string name="7cFwXWcNqYvBRzgh555Pw">Jg3xcDITiyK4rN3+4biPwg</string>  
    <string name="JD1l6inbJWv30KurgJkpAg">Zeg6wpqkWjwNEOLQMHzsuQ</string>  
    <string name="cgASLY/nymum+yv+S+2itmlSHPFDMElhAT9p0/5DcA">uKJlx6JdM0JaCABUhxETBRvUliCmjJidGLtierH/6s</string>  
    <string name="UZFTE9ZT5s49H1G4enERPA">W8KyltBxcICu10mMi7ys0Q</string>  
</map>
```

Figura 4.51 Cifrado de la información en el fichero de preferencias.

Internamente, la librería utiliza el algoritmo AES para su cifrado. Se genera aleatoriamente una clave de cifrado cuando creamos una instancia de SharedPreferences por primera vez en la aplicación. Esta clave encripta los pares clave-valor y, luego, los datos encriptados se codifican utilizando el algoritmo base64 antes de almacenarlos en un archivo XML. Dado que AES es un algoritmo simétrico, la misma clave se utilizará para descifrar todos los pares de clave-valor cifrados.

```
private static String generateAesKeyValue() throws NoSuchAlgorithmException {  
    // Do "not" seed secureRandom. Automatically seeded from system entropy  
    final SecureRandom random = new SecureRandom();  
  
    // Use the largest AES key length which is supported by the OS  
    final KeyGenerator generator = KeyGenerator.getInstance("AES");  
    try {  
        generator.init(KEY_SIZE, random);  
    } catch (Exception e) {  
        try {  
            generator.init(192, random);  
        } catch (Exception e1) {  
            generator.init(128, random);  
        }  
    }  
    return SecurePreferences.encode(generator.generateKey().getEncoded());  
}
```

Figura 4.52 Método que realiza el cifrado a partir de la clave generada.

#### 4.4.17 Almacenamiento en ficheros

Por defecto, los archivos guardados en la memoria del dispositivo son privados y el resto de las aplicaciones no podrán acceder a ellos. Para crear el archivo donde vamos a almacenar los datos, usaremos la clase “`FileOutputStream`”, y llamaremos a su método “`openFileOutput(name, mode)`”. Una vez creada la instancia, podremos escribir en el archivo con el método “`write()`”. Por el contrario, si queremos leer un archivo, usaremos la clase “`InputStream`”, y llamaremos a su método “`openFileInput(name)`”. Leeremos el archivo con el método “`read()`”.

Como primer parámetro “`name`”, indicaremos el nombre del archivo para crear/modificar y, como segundo parámetro “`mode`”, el modo de acceso para manejar esos datos. Los modos más usados en este caso son los que siguen:

```
MODE_PRIVATE  
MODE_WORLD_READABLE  
MODE_WORLD_WRITEABLE  
MODE_APPEND
```

Los tres primeros modos nos crearán o reemplazarán el archivo que se va a escribir; es decir, nos eliminarán el antiguo y crearán uno nuevo. En cambio, el último modo nos permite seguir escribiendo el mismo archivo. Por último, para finalizar de leer o escribir en el archivo, deberemos llamar al método “`close()`”, para cerrar el archivo. En ambos casos de lectura o escritura, podremos usar los filtros de Java, como pueden ser búferes, readers o writers.

#### 4.4.18 Almacenamiento externo

Antes de usar el almacenamiento externo, podríamos realizar algunas comprobaciones extra, como comprobar si está montada la unidad, ya que podría estar inaccesible, al tener conectado el móvil como almacenamiento masivo en el PC y tener permiso de escritura en ella. Para comprobar si está montada la unidad externa, lo podemos hacer usando la clase “`Environment`” y su método “`getExternalStorageState()`”:

```
boolean mExternalStorageAvailable = false;  
boolean mExternalStorageWriteable = false;  
String state = Environment.getExternalStorageState();
```

```
// COMPROBACION DEL ALMACENAMIENTO EXTERNO
if (Environment.MEDIA_MOUNTED.equals(state)) {
    // MONTADO EN MODO LECTURA Y ESCRITURA
    mExternalStorageAvailable = mExternalStorageWriteable = true;
} else if
    (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        // MONTADO EN MODO LECTURA
        mExternalStorageAvailable = true;
        mExternalStorageWriteable = false;
} else {
    // No podremos leer ni escribir en ella
    mExternalStorageAvailable = mExternalStorageWriteable = false;
}
```

Tenemos varios medios de comprobación en la clase **Environment**:

- **MEDIA\_BAD\_REMOVAL**: El dispositivo se retiró antes de que fuera desmontado.
- **MEDIA\_CHECKING**: El dispositivo se está comprobando.
- **MEDIA\_MOUNTED**: El dispositivo está montado y preparado para lectura o escritura.
- **MEDIA\_MOUNTED\_READ\_ONLY**: El dispositivo está montado en modo de solo lectura.
- **MEDIA\_NOFS**: El dispositivo tiene un sistema de archivos no soportado.
- **MEDIA\_REMOVED**: El dispositivo se ha retirado.
- **MEDIA\_SHARED**: El dispositivo está compartido a través del almacenamiento masivo en USB.
- **MEDIA\_UNMOUNTABLE**: El dispositivo está presente, pero no se puede montar.
- **MEDIA\_UNMOUNTED**: El dispositivo está presente, pero no está montado.

Por otro lado, para tener permisos de lectura y escritura, deberemos añadir a nuestro archivo Manifest del proyecto los siguientes permisos:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

#### 4.4.19 Implementación segura de Intents

Los Intents son componentes de Android que se utilizan como mecanismo para **pasar información entre componentes** y se pueden usar de diferentes formas:

- Para iniciar una actividad; por ejemplo, para abrir una interfaz de usuario.
- Para iniciar, detener y comunicarse con un servicio en segundo plano.
- Para acceder a los datos a través de **ContentProviders**.
- Como mecanismo de *callbacks* para manejar eventos.

Los componentes a los que se accede mediante Intents pueden ser públicos o privados. El valor predeterminado depende del filtro que se esté utilizando y es fácil permitir que el componente se haga público por error. De esta forma, resulta posible configurar el componente como **exported = false** en el manifiesto de la aplicación para evitar esto. Los componentes públicos declarados en el manifiesto se encuentran abiertos por defecto, para que cualquier aplicación pueda acceder a ellos. Si no estima necesario que todas las demás aplicaciones accedan a un componente, considere establecer un permiso en el componente declarado en el manifiesto.

A veces, presenta un componente (por ejemplo, un servicio) que no desea exponer al resto del sistema y aplicaciones por motivos de seguridad. Para hacer esto, puede establecer el atributo **android:exported=false** dentro de la etiqueta `<activity>`, en el fichero `AndroidManifest.xml`, para todas aquellas activities a las cuales se necesite restringir el acceso por parte de tercera app del mismo dispositivo, lo que permite ocultar ese componente al resto de aplicaciones.

Por ejemplo, si tenemos los siguientes paquetes `com.example.A.X` y `com.example.B.Y`, que la clase `A.X` pueda interactuar con `B.Y` depende de cómo se configure este atributo. Por defecto, el valor `android:exported` es "false", excepto si el componente define un intent-filter; en este caso, el valor por defecto pasa a ser true:

```
<activity
    android:name="com.example.B.Y"
    android:exported="true" or (android:exported="false")
</activity>
```

El valor predeterminado variará de acuerdo con los filtros de Intents que tenga la actividad. La ausencia de filtros significa que la actividad solo se puede

invocar si se especifica el nombre exacto de la clase. Esto quiere decir que la actividad solo está destinada al uso interno de la aplicación (dado que otras entidades no conocerán el nombre de la clase). Entonces, en este caso, el valor predeterminado es "false". Por otro lado, la presencia de uno o más filtros implica que la actividad está destinada al uso externo; en consecuencia, el valor predeterminado es "true".

Si desea que un componente esté disponible para otras aplicaciones, pero quiere proporcionar un nivel de seguridad adecuado, puede proporcionar un permiso específico en el archivo manifiesto mediante la directiva **uses-permission**. Posteriormente, puede usar este permiso y aplicarlo a un determinado componente a través del atributo **android:permission**.

Con este atributo, resulta posible definir el nombre del permiso que los clientes deben tener para poder iniciar la actividad, o para que la actividad responda a un determinado Intent. Si el emisor de la actividad a través del Intent no cuenta con el permiso especificado, el Intent no funcionará y no se iniciará la actividad asociada al Intent.

#### 4.4.20 Implementación segura de servicios

Lo mismo que hemos visto en el punto anterior para actividades, se puede aplicar también a servicios. Por defecto, los servicios no se exportan ni pueden ser invocados por otras aplicaciones. Se recomienda protegerlos con permisos en el caso de ser exportados.

Se deberá controlar el acceso a los diferentes services de una aplicación por parte de terceras app instaladas en el mismo dispositivo. Se recomienda que la app especifique la directiva **android:exported="false"** dentro de la etiqueta **<service>** en el fichero **AndroidManifest.xml**, para todos aquellos servicios a los cuales se necesite restringir el acceso por parte de terceras app del mismo dispositivo:

```
<service android:name="com.example.MyService" android:exported="false" />
```

#### 4.4.21 Implementación segura de broadcast receivers

Se deberá controlar el acceso a los diferentes Broadcast Receivers de una app por parte de terceras app instaladas en el mismo dispositivo. Se recomienda que la app especifique la directiva **android:exported="false"** dentro de la etiqueta

<receiver> en el fichero AndroidManifest.xml, para todos aquellos Broadcast Receivers a los cuales se necesite restringir el acceso por parte de terceras app del mismo dispositivo. Se deberá controlar quién puede acceder a estos eventos a través de permisos:

```
<receiver android:name="MyBroadCastReceiver" android:exported="false">
<intent-filter>
<action android:name="MyBroadcast"></action>
</intent-filter>
</receiver>
```

#### 4.4.22 Implementación segura de content providers

Se deberá controlar el acceso a los diferentes content providers de una app por parte de terceras app instaladas en el mismo dispositivo. Se recomienda que la app especifique la directiva android:exported="false" dentro de la etiqueta <provider> en el fichero AndroidManifest.xml, para todos aquellos content providers a los cuales se necesite restringir el acceso por parte de terceras app del mismo dispositivo. Se deberá limitar el intercambio de datos con aplicaciones externas personalizando los permisos:

```
<provider android:name="MyProvider"
  android:authorities= "com.example.MyProvider" android:exported="false">
</provider>
```

#### 4.4.23 Invocar actividades de forma segura

Normalmente, en las aplicaciones de Android, una actividad es una “pantalla” en una aplicación. Una actividad puede ser invocada por cualquier aplicación si se exporta. Algunas cosas que se deben tener en cuenta son estas:

- Si define un intent filter para una actividad, automáticamente lo hará público y será accesible para otras aplicaciones.
- Incluso si una actividad no es pública, puede invocarse con privilegios de root. Esto podría permitir que un atacante se saltara la protección o sandbox que trae una aplicación por defecto o de una manera que el desarrollador no contemplaba. Para evitar esto, la actividad puede verificar si la aplicación se encuentra en estado “desbloqueado” y, si no, volver a la pantalla de bloqueo.

- No se deben enviar datos confidenciales en los Intents utilizados para iniciar actividades. Un programa malintencionado puede insertar un intent filter con mayor prioridad y capturar los datos.

#### 4.4.24 Implementar almacenamiento de datos seguro

El almacenamiento de datos de forma segura en un dispositivo móvil requiere de la implementación de las técnicas adecuadas. Para almacenar datos confidenciales, se recomienda utilizar las funciones criptográficas proporcionadas por la propia plataforma. Las bibliotecas criptográficas de Android proporcionan una serie de funciones criptográficas estándar que, si se implementan correctamente, pueden proporcionar una implementación criptográfica razonablemente segura.

Para cifrar los datos del usuario, resulta importante hacerlo con una clave maestra generada de forma aleatoria, que también se cifra con una contraseña proporcionada por el usuario cada vez que se accede a los datos. Esto evitara que los datos se recuperen fácilmente si un atacante extrae la clave maestra del dispositivo. Debido a la cantidad de vulnerabilidades en las API y la falta de cifrado en la mayoría de los dispositivos Android, se recomienda que la clave maestra se almacene en un servidor externo a la aplicación.

Android implementa bibliotecas criptográficas estándar, como AES, que pueden usarse para proteger los datos, tanto credenciales del usuario como información confidencial que maneja la aplicación. Recuerde que los datos cifrados con este método son tan seguros como la contraseña utilizada para obtener la clave y la administración de claves.

Hay que tener en cuenta que el uso del proveedor criptográfico estándar AES se establecerá de forma predeterminada en el AES-ECB, el modo menos seguro. Se recomienda especificar AES-CBC con una clave de 256 bits y un IV aleatorio generado por **SecureRandom**. Se aconseja que la clave se derive de una frase de contraseña utilizando el algoritmo PBKDF2 (función de derivación de clave basada en contraseña).

El estándar AES utiliza cifrado simétrico y permite cifrar y descifrar los datos a partir de la misma clave. Existen diferentes tamaños de clave y **AES 256 bits** constituye la longitud preferida cuando estamos tratando con datos confidenciales.

La seguridad en este punto al utilizar AES es que va a depender de la longitud y complejidad de la clave que usemos. Esta es la razón por la que no se

recomienda emplear una contraseña directamente para cifrar los datos. Aquí entra en juego el algoritmo PBKDF2.

PBKDF2 permite obtener una clave a partir de una contraseña y un valor aleatorio inicial llamado salt o semilla. Este valor es una secuencia aleatoria de datos y hace que la clave derivada sea única, incluso si otro usuario utilizó la misma contraseña. Podemos generar dicho valor aleatorio con la función **SecureRandom**, que se encuentra dentro del paquete **java.security**.\*. La clase SecureRandom garantiza que la salida generada resulte difícil de predecir:

```
SecureRandom random = new SecureRandom();
byte salt[] = new byte[256];
random.nextBytes(salt);
```

Android también proporciona las clases **javax.crypto.spec.PBEKeySpec** y **javax.crypto.SecretKeyFactory**, para facilitar la generación de las claves de cifrado a partir de la contraseña.

Una vez hayamos generado el salt, podemos utilizar la clase **PBEKeySpec**. El constructor de esta clase acepta por parámetros la contraseña, el *salt* y el número de iteraciones. Al aumentar el número de iteraciones, también lo hace el tiempo que llevaría realizar un ataque de fuerza bruta. El objeto **PBEKeySpec** se pasa luego al **SecretKeyFactory**, y genera la clave como una matriz de bytes []. Finalmente, convertimos esa matriz de bytes en un objeto **SecretKeySpec**:

```
char[] passwordChar = passwordString.toCharArray();
PBEKeySpec pbKeySpec = new PBEKeySpec(passwordChar, salt, 1324, 256);
SecretKeyFactory secretKeyFactory = SecretKeyFactory.getInstance("PBKDF2WithHmac-
SHA1");
byte[] keyBytes = secretKeyFactory.generateSecret(pbKeySpec).getEncoded();
SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");
```

Antes de cifrar los datos, se ha de señalar que vamos a usar AES en modo de cifrado CBC y un vector de inicialización (IV). Un IV es un bloque de bytes aleatorios donde cada bloque depende de todos los bloques procesados hasta ese momento:

```
SecureRandom ivRandom = new SecureRandom();
byte[] iv = new byte[16];
ivRandom.nextBytes(iv);
IvParameterSpec ivSpec = new IvParameterSpec(iv);
```

Una vez disponemos del objeto IvParameterSpec, podemos empezar con el cifrado:

```
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding");
cipher.init(Cipher.ENCRYPT_MODE, keySpec, ivSpec);
byte[] encrypted = cipher.doFinal(plainTextBytes);
```

Para el cifrado, utilizamos la cadena "AES/CBC/PKCS7Padding". Esto especifica el cifrado AES con encadenamiento de bloques cifrados. La última parte de esta cadena se refiere a PKCS7, que es un estándar establecido para datos de relleno que no encajan perfectamente en el tamaño del bloque (los bloques son de 128 bits y el relleno se realiza antes del cifrado).

Para completar nuestro ejemplo, pondremos este código en un método de cifrado llamado **cipherBytes**, que devolverá el resultado en un HashMap que contenga los datos cifrados, junto con la semilla y el vector de inicialización necesarios para el descifrado:

```
private HashMap<String, byte[]> cipherBytes(byte[] plainTextBytes, String passwordString)
{
    HashMap<String, byte[]> map = new HashMap<String, byte[]>();

    try
    {
        //Generar semilla inicial aleatoria
        SecureRandom random = new SecureRandom();
        byte salt[] = new byte[256];
        random.nextBytes(salt);

        //algoritmo PBKDF2
        char[] passwordChar = passwordString.toCharArray();
        PBEKeySpec pbKeySpec = new PBEKeySpec(passwordChar, salt, 1324, 256);
        SecretKeyFactory secretKeyFactory = SecretKeyFactory.getInstance("PBKDF2WithH-
macSHA1");
        byte[] keyBytes = secretKeyFactory.generateSecret(pbKeySpec).getEncoded();
        SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");

        //Crear vector de inicialización
        SecureRandom ivRandom = new SecureRandom();
        byte[] iv = new byte[16];
        ivRandom.nextBytes(iv);
        IvParameterSpec ivSpec = new IvParameterSpec(iv);

        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding");
        cipher.init(Cipher.ENCRYPT_MODE, keySpec, ivSpec);
        byte[] encrypted = cipher.doFinal(plainTextBytes);

        map.put("salt", salt);
        map.put("key", keyBytes);
        map.put("iv", iv);
        map.put("encrypted", encrypted);
    }
}
```

```

//Ciframos con el modo ENCRYPT_MODE
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding");
cipher.init(Cipher.ENCRYPT_MODE, keySpec, ivSpec);
byte[] encrypted = cipher.doFinal(plainTextBytes);

map.put("salt", salt);
map.put("iv", iv);
map.put("encrypted", encrypted);
}

catch(Exception e)
{
    Log.e("MYAPP", "encryption exception", e);
}

return map;
}

```

Para **descifrar** los datos, el único cambio se produce en el constructor de clase Cipher, donde se cambia el modo ENCRYPT\_MODE a **DECRYPT\_MODE**.

El método de descifrado tomará el HashMap obtenido en el proceso de cifrado y devolverá un array de bytes con el contenido descifrado. El método de descifrado regenerará la clave de cifrado a partir de la contraseña:

```

private byte[] decipherData(HashMap<String, byte[]> map, String passwordString)
{
    byte[] decrypted = null;
    try
    {
        byte salt[] = map.get("salt");
        byte iv[] = map.get("iv");
        byte encrypted[] = map.get("encrypted");

        //regenerar clave a partir de la password
        char[] passwordChar = passwordString.toCharArray();
        PBEKeySpec pbKeySpec = new PBEKeySpec(passwordChar, salt, 1324, 256);
        SecretKeyFactory secretKeyFactory = SecretKeyFactory.getInstance("PBKDF2WithH-
macSHA1");
        byte[] keyBytes = secretKeyFactory.generateSecret(pbKeySpec).getEncoded();
        SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");
    }
}

```

```
//Desifrar con el modo DECRYPT_MODE
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding");
IvParameterSpec ivSpec = new IvParameterSpec(iv);
cipher.init(Cipher.DECRYPT_MODE, keySpec, ivSpec);
decrypted = cipher.doFinal(encrypted);
}
catch(Exception e)
{
    Log.e("MYAPP", "decryption exception", e);
}

return decrypted;
}
```

Ahora podemos probar nuestros métodos, para asegurarnos de que los datos se descifran correctamente después del cifrado:

```
//Cifrar
String string = "My sensitive string that I want to encrypt";
byte[] bytes = string.getBytes();
HashMap<String, byte[]> map = cipherBytes(bytes, "UserPassword");

//Descifrar
byte[] decrypted = decipherData(map, "UserPassword");
if(decrypted != null)
{
    String decryptedString = new String(decrypted);
    Log.e("MYAPP", "Decrypted String is : " + decryptedString);
}
```

Ahora que tenemos una matriz de bytes con los datos cifrados, podemos guardarla a nivel de fichero con la clase **FileOutputStream**. También podemos guardar el **HashMap** de cifrado con la clase **ObjectOutputStream**:

```
FileOutputStream fos = openFileOutput("test.dat", Context.MODE_PRIVATE);
fos.write(encrypted);
fos.close();

FileOutputStream fos = openFileOutput("map.dat", Context.MODE_PRIVATE);
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(map);
oos.close();
```

También podemos guardar los datos cifrados en las **preferencias compartidas** de la aplicación:

```
SharedPreferences.Editor editor = getSharedPreferences("prefs", Context.MODE_PRIVATE).edit();
String keyBase64String = Base64.encodeToString(encryptedKey, Base64.NO_WRAP);
String valueBase64String = Base64.encodeToString(encryptedValue, Base64.NO_WRAP);
editor.putString(keyBase64String, valueBase64String);
editor.commit();
```

Dado que SharedPreferences es un sistema XML que acepta solo datos primitivos y objetos específicos como valores, debemos convertir nuestros datos a un formato compatible, como un objeto String. Base64 nos permite convertir los datos sin procesar en una representación de cadena que contiene solo los caracteres permitidos por el formato XML.

En el ejemplo anterior, encryptedKey y encryptedValue son matrices cifradas que se devuelven desde nuestro método cipherBytes(). Para recuperar los bytes cifrados de las SharedPreferences, podemos aplicar una decodificación Base64 en la cadena almacenada:

```
SharedPreferences preferences = getSharedPreferences("prefs", Context.MODE_PRIVATE);
String base64EncryptedString = preferences.getString(keyBase64String, "default");
byte[] encryptedBytes = Base64.decode(base64EncryptedString, Base64.NO_WRAP);
```

#### 4.4.25 Algoritmos criptográficos

Las aplicaciones Android pueden necesitar almacenar información sensible de forma cifrada, como contraseñas propias o de servicios externos, datos personales, etc. El SDK Android, en todas sus versiones, incluye el framework JCE (Java Cryptography Extension), que provee de una interfaz para acceder, de forma sencilla, a las operaciones comunes de cifrado de manera nativa.

Aun así, se puede realizar un uso de estos algoritmos de cifrado que provoque que la aplicación se muestre vulnerable, debido al uso de algoritmos obsoletos o a la elección de modos de cifrado que no sean los más correctos desde el punto de vista de la seguridad. El mal uso de estas librerías conforma una fuente común de vulnerabilidades de seguridad, como demuestran la mayoría de los estudios realizados sobre el incorrecto uso de métodos criptográficos.

Para poder llevar a cabo la implementación de un **cifrado simétrico de forma segura** en Android, se han de tener en cuenta las posibles alternativas para su correcta implementación. A continuación se enumeran las vulnerabilidades más frecuentes en el cifrado en la plataforma Android y las alternativas que se recomiendan.

No se deberían usar algoritmos de cifrado que actualmente se consideran vulnerables, por ejemplo, el algoritmo DES (Data Encryption Standard). En su lugar, se recomienda el uso del algoritmo AES (Advanced Encryption Standard), considerado como seguro y presente en todas las versiones de la plataforma Android. Existen tres versiones del algoritmo AES: 128, 192 y 256.

Se recomienda el uso de AES con una longitud de clave de 256 bits, aunque también sería admisible el uso de una longitud de clave de 128 bits si fuera necesario, por compatibilidad con las versiones de Android o por motivos de rendimiento de la aplicación.

Los algoritmos de hashing como MD5 y SHA-1 están obsoletos y criptográficamente rotos. Los hashes o las funciones de resumen son algoritmos que consiguen crear, a partir de una entrada (ya sea un texto, una contraseña, un archivo, etc.), una salida alfanumérica de longitud normalmente fija, que representa un resumen de toda la información de entrada (es decir, a partir de los datos de la entrada, crea una cadena que solo debería poderse volver a crear con esos mismos datos). Estas características hacen a los algoritmos de hash apropiados para asegurar la integridad de los datos ya que, si se producen modificaciones en los datos, el hash será distinto y podremos detectar que se han modificado.

Estos algoritmos de hashing se utilizan junto a los mecanismos de cifrado, para asegurar la integridad de los datos cifrados, como se explica más adelante en el apartado de modos de operación AES.

Los algoritmos de cifrado en bloque transforman bloques de texto en claro y los convierten en bloques de texto cifrado. Sin embargo, el algoritmo base puede estar influido de distintas formas por las aplicaciones del algoritmo a los bloques anteriores en la cadena de caracteres del texto en claro. A estas distintas formas de influencia se las llama "modos de operación". Existen diferentes modos de operación que garantizan diversos grados de confidencialidad e integridad en los datos manejados.

Por defecto, AES utiliza el modo ECB (Electronic Code Book Mode), que cifra cada bloque de la misma forma, lo que provoca que sea vulnerable a técnicas de análisis estadístico. Además, el carecer de comprobación de integridad, junto al echo de que cada bloque se descifra exactamente de la misma manera, lo hace vulnerable a ataques por repetición.

Una posible alternativa consiste en utilizar el modo de operación CBC (Cipher Block Chaining Mode), que se considera más seguro y que está presente en la mayoría de las versiones de Android. Este modo de operación, igualmente, carece de comprobación de integridad.

Si se necesita que la aplicación se muestre compatible con versiones más antiguas de Android, se recomienda el uso de CBC realizando esta comprobación de integridad, con el fin de asegurarse de que los mensajes no hayan sido modificados. Una posible implementación de la comprobación de integridad podría estar basada en el uso de HMAC-SHA256 sobre el texto para cifrar con una clave de cifrado diferente, y que podría ser almacenado junto al texto cifrado, lo que forma como resultado una clave compuesta donde los 128 bits primeros se usan para cifrar y los 256 restantes, para comprobar la integridad.

#### 4.4.26 Uso de `java.util.String` para almacenar información sensible

Se recomienda evitar el uso de la clase `java.util.String` para almacenar las cadenas de texto que contengan información sensible, como pueda ser la información relativa a las claves de cifrado o contraseñas. Resulta preferible usar para su almacenamiento y tratamiento objetos de tipo array de caracteres (`char[]`) o array de bytes (`byte[]`).

Esto se debe a que los objetos de la clase `String` resultan inmutables, lo que significa que no pueden ser alterados. Una vez que un objeto de tipo `String` se crea, no se puede modificar ni tampoco borrar. Cualquier modificación sobre este objeto producirá un nuevo objeto, y dejará intacto el original. Por este motivo, una vez que se ha utilizado un objeto de tipo `String` para almacenar un valor, este permanecerá en memoria, y será vulnerable a que cualquier otro proceso pueda hacer un volcado de memoria y acceder a dicha información.

Esta característica hace que los objetos de tipo `String` no resulten apropiados para el almacenamiento de información sensible como contraseñas. Siempre se debería recoger este tipo de información en objetos de tipo array de caracteres.

#### 4.4.27 Proteger la configuración de la aplicación

Los desarrolladores de Android normalmente almacenan configuraciones en un archivo XML de preferencias compartidas (Shared Preferences) o SQLite DB, que no están cifradas de forma predeterminada y que se pueden leer o incluso modificar con permisos de root.

Resulta importante no almacenar ninguna configuración crítica en diccionarios u otros archivos, a menos que se encuentre cifrado primero. Lo ideal es cifrar todos los archivos de configuración con una clave maestra encriptada con una frase de contraseña proporcionada por el usuario, o con una clave proporcionada de forma remota, cuando un usuario inicia sesión en la aplicación.

#### 4.4.28 Cifrado en base de datos SQLite

Las aplicaciones móviles almacenan datos del lado del cliente en la base de datos SQLite en el dispositivo. La información en esta base de datos, a menudo, no se halla encriptada y, por lo tanto, puede contener información confidencial, como números de cuenta o SSN. También puede contener información del estado de la aplicación, que podría modificarse para omitir la lógica de la aplicación.

Desde una perspectiva de mejores prácticas, los datos confidenciales nunca deben almacenarse en el lado del cliente, en la medida de lo posible, y siempre deben mantenerse en el lado del servidor.

El cifrado de los datos en la base de datos SQLite resulta posible gracias a soluciones como la de sqlcipher <https://www.zetetic.net/sqlcipher/open-source>. SQLCipher es una extensión SQLite que proporciona un cifrado AES transparente de 256 bits de los archivos de base de datos.

Para usar SQLCipher, primero tenemos que añadir las **librerías sqlcipher.jar**, **commons-codec.jar** y **guava.jar**, que se pueden descargar desde el sitio web [sqlcipher.net](http://sqlcipher.net). Posteriormente, tenemos que añadir el import correspondiente a la clase **import net.sqlcipher.database.SQLiteDatabase;**, para importar SQLCipher, inicializar las librerías con **SQLiteDatabase.loadLibs(this)**, y crear la base de datos con la instrucción **SQLiteDatabase.openOrCreateDatabase(databaseFile,"password",null);**:

```
package com.example.sqlcipher;

import java.io.File;
import android.os.Bundle;
import android.app.Activity;
import net.sqlcipher.database.SQLiteDatabase;

public class MainActivity extends Activity {
```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    initializeSQLCipher();
}

private void initializeSQLCipher() {
    SQLiteDatabase.loadLibs(this);
    File databaseFile = getDatabasePath("database.db");
    databaseFile.mkdirs();
    databaseFile.delete();
    SQLiteDatabase database =
        SQLiteDatabase.openOrCreateDatabase(databaseFile,"password",null);
    database.execSQL("create table user(id integer primary key autoincrement,
    " +
        "first text not null, last text not null, " +
        "username text not null, password text not null)");
    database.execSQL("insert into user(first,last,username, password) " +
        "values('name','lastname','user','password')");
}
}

```

#### 4.4.29 Optimización y ofuscación del código con ProGuard

ProGuard <https://www.guardsquare.com/en/products/proguard> es una herramienta que permite optimizar y ofuscar el código, detectando y eliminando las clases no utilizadas, campos, métodos y atributos. Se optimiza el código en bytes y elimina las instrucciones que no estén siendo utilizadas. Se cambia el nombre de las clases restantes, campos y métodos por nombres cortos sin sentido aparente. Entre las principales características, podemos destacar:

- Por defecto, al instalar el SDK de Android, viene incluido.
- Solo debemos activarlo modificando el fichero **project.properties** y activar el flag **proguard.config=proguard.cfg**.
- Todas las opciones que queramos incluir las escribiremos en el fichero **proguard.cfg**.
- Cuando exportemos la aplicación, automáticamente nos generará el fichero APK con el código de esta ofuscado.
- Podemos utilizar la plantilla que ya está predefinida en el directorio ProGuard.

Una aplicación de Android es el resultado de la compilación de los archivos Java, XML y otros recursos. El código Java se compila en un formato binario llamado dex, que es lo que interpreta la máquina virtual Dalvik cuando ejecuta el código. Este formato no ha sido diseñado para ser legible por humanos y existen herramientas que permiten descompilar un archivo dex de nuevo a un formato legible por humanos.

En algunos casos, descompilar el código puede constituir un problema de seguridad; por ejemplo, cuando el código contiene claves secretas u otros valores que no deberían ser accesibles.

Aunque no resulta posible evitar completamente la descompilación de su código, se puede hacer mucho más difícil usando técnicas de ofuscación antes de publicarlo. Este método hará que se dificulte el proceso de ingeniería inversa sobre la aplicación.

Gracias a la herramienta ProGuard, es posible ofuscar el código para aplicaciones Android usando la herramienta integrada en el SDK de Android. La herramienta se muestra compatible con el sistema de compilación Gradle, y todo lo que necesita hacer es añadir las siguientes líneas de configuración a la sección de Android en `build.gradle`:

```
buildTypes {  
    release {  
        runProguard true  
        proguardFile getDefaultProguardFile('proguard-android.txt')  
    }  
}
```

Esto permitirá que ProGuard se aplique a la versión de compilación de su aplicación. Otra razón para ofuscar su código estriba en que, al hacerlo, se realizan algunas optimizaciones adicionales, además de reducir el binario dex resultante, al eliminar el código no utilizado. Esto es especialmente útil cuando ha incluido bibliotecas de terceros, ya que puede reducir significativamente el tamaño del archivo final y el uso de memoria en tiempo de ejecución.

ProGuard también reduce la cantidad de código, al eliminar los métodos, campos y atributos no utilizados, y hace que se ejecute más rápido mediante el uso de código optimizado. Un buen recurso a nivel de configuración se puede encontrar en la dirección <http://proguard.sourceforge.net/manual/examples.html#androidapplication>

```
-injars bin/classes
-injars bin/resources.ap_
-injars libs
-outjars bin/application.apk
-libraryjars /usr/local/android-sdk/platforms/android-28/android.jar

-android
-dontpreverify
-repackageclasses ""
-allowaccessmodification
-optimizations !code/simplification/arithmetic
-keepattributes *Annotation*

-keep public class * extends android.app.Activity
-keep public class * extends android.app.Application
-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep public class * extends android.content.ContentProvider

-keep public class * extends android.view.View {
    public <init>(android.content.Context);
    public <init>(android.content.Context, android.util.AttributeSet);
    public <init>(android.content.Context, android.util.AttributeSet, int);
    public void set*(...);
}

-keepclasseswithmembers class * {
    public <init>(android.content.Context, android.util.AttributeSet);
}

-keepclasseswithmembers class * {
    public <init>(android.content.Context, android.util.AttributeSet, int);
}

-keepclassmembers class * extends android.content.Context {
    public void *(android.view.View);
    public void *(android.view.MenuItem);
}

-keepclassmembers class * implements android.os.Parcelable {
    static ** CREATOR;
}

-keepclassmembers class **.R$* {
    public static <fields>;
}

-keepclassmembers class * {
    @android.webkit.JavascriptInterface <methods>;
}
```

También podríamos utilizar la interfaz de usuario que ofrece ProGuard, que proporciona guía a nivel visual a través de los ajustes de configuración. En la documentación oficial de Android <https://developer.android.com/studio/build/shrink-code>, podemos consultar más información acerca de minimizar el tamaño de las aplicaciones.

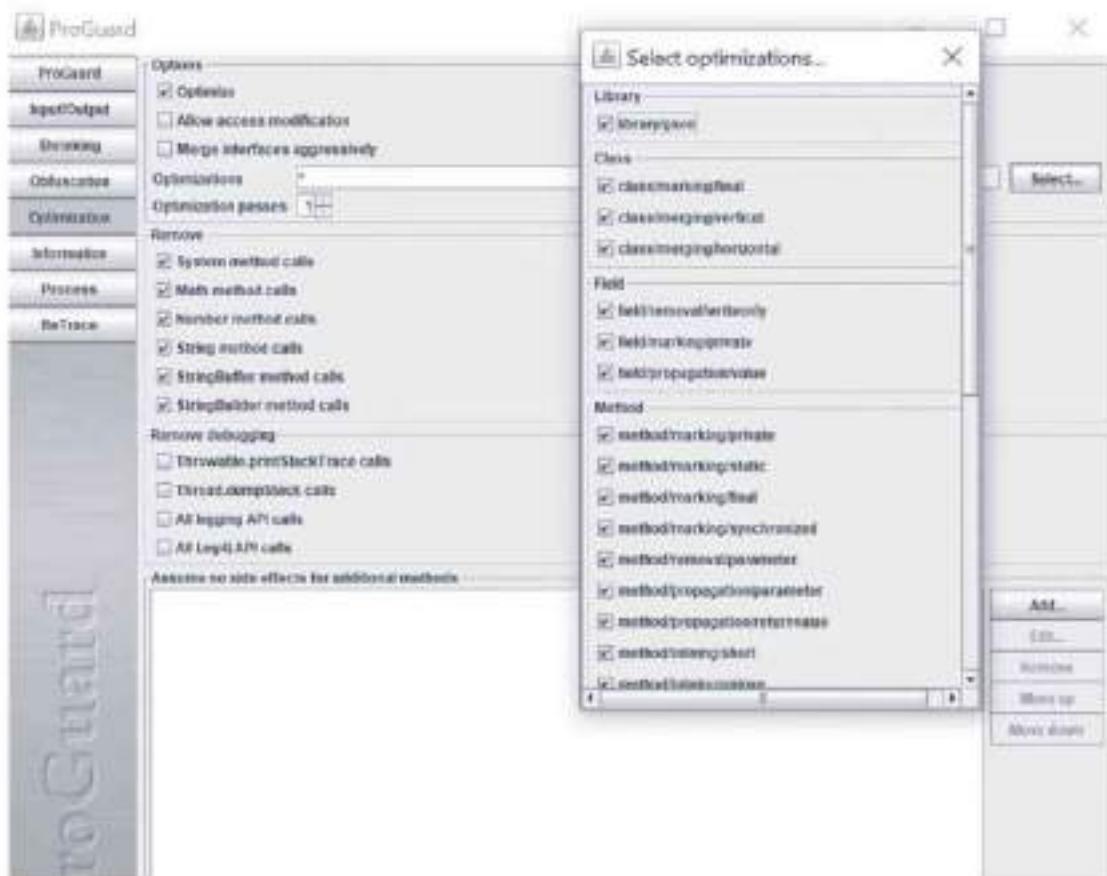


Figura 4.53 Interfaz visual de ProGuard para seleccionar optimizaciones.

## 4.5 Metodología OASAM

OASAM <https://github.com/b66l/OASAM> es el acrónimo de **Open Android Security Assessment Methodology**, que tiene como objetivo conformarse como una metodología de análisis de seguridad de aplicaciones bajo la plataforma Android y es una referencia para el análisis. Dicha metodología cuenta con controles de seguridad que se estructuran en diferentes secciones, que son:

- **OASAM-INFO - Information Gathering.** Obtención de información y definición de una superficie de ataque.
- **OASAM-CONF - Configuration and Deploy Management.** Análisis de la configuración e implantación.
- **OASAM-AUTH - Authentication.** Análisis de la autenticación.
- **OASAM-CRYPT - Cryptography.** Análisis del uso de criptografía.
- **OASAM-LEAK - Information Leak.** Análisis de fugas de información sensible.
- **OASAM-DV - Data Validation.** Análisis de gestión de la entrada de usuario.
- **OASAM-IS - Intent Spoofing.** Análisis de la gestión en la recepción de Intents.
- **OASAM-UIR - Unauthorized Intent Receipt.** Análisis de la resolución de Intents.
- **OASAM-BL - Business Logic.** Análisis de la lógica de negocio de la aplicación.

En el caso de la criptografía, se desglosaría en los siguientes puntos:

- **OASAM-CRYPT-001:** Contraseñas almacenadas en el código fuente.
- **OASAM-CRYPT-002:** Vulnerabilidades relacionadas con el almacenamiento de información confidencial.
- **OASAM-CRYPT-003:** Vulnerabilidades relacionadas con el transporte de información no segura.
- **OASAM-CRYPT-004:** Vulnerabilidades relacionadas con las cadenas de certificaciones digitales de confianza.
- 

		Last commit (viewed) on 29 Dec 2016
1	blob update README.md	
2	oasam-auth-authentication	Update README.md 3 years ago
3	oasam-bl-business-logic	Create README.md 3 years ago
4	oasam-conf-configuration-and-deploy-management	Update README.md 3 years ago
5	oasam-crypt-cryptography	Update README.md 3 years ago
6	oasam-dv-data-validation	Update README.md 3 years ago
7	oasam-info-information-gathering	Update README.md 3 years ago
8	oasam-is-intent-spoofing	Update README.md 3 years ago
9	oasam-leak-information-leak	Update README.md 3 years ago
10	oasam-uir-unauthorized-intent-receipt	Update README.md 3 years ago
11	README.md	Update README.md 3 years ago
12	.config.yml	Set theme jekyll-theme-hacker 3 years ago

Figura 4.54 Repositorio de GitHub donde se describe la metodología.

## Capítulo 5. Seguridad en proyectos NodeJS

Este capítulo se puede complementar con conferencias impartidas por el autor, tanto en español como en inglés. Algunas de las presentaciones se pueden encontrar en el espacio de speakerdeck <https://speakerdeck.com/jmortega/hacking-nodejs-applications-for-fun-and-profit>

The screenshot shows a presentation slide with the following content:

- Hacking NodeJS applications for fun and profit**
- Testing NodeJS Security**
- by @jmortega**
- Agenda**
  - Introduction nodeJS security
  - Npm security packages
  - Node Goat project
  - Tools

Figura 5.1 Presentación de Hacking NodeJS applications for fun and profit.

### 5.1 Introducción a NodeJS

NodeJS surge en el año 2009 como respuesta a algunas necesidades encontradas a la hora de desarrollar sitios web específicamente orientados a ofrecer concurrencia y manejar las peticiones de una forma más rápida. NodeJS es una plataforma especialmente diseñada para realizar operaciones de entrada/salida en redes por medio de distintos protocolos. Es, además, una de las plataformas que ha provocado, junto con HTML5, que JavaScript gane gran relevancia en los últimos tiempos, pues ha conseguido llevar al lenguaje a nuevas fronteras, como es el trabajo del lado del servidor.

El framework ha sido construido con el motor de JavaScript V8 de Google, que permite que JavaScript se ejecute del lado del servidor. Dentro del ecosistema de JavaScript, ha alcanzado una gran popularidad el conocido como stack MEAN, compuesto por:

- **MongoDB:** Base de datos NoSQL orientada a documentos.
- **Express:** Framework MVC y para servicios REST, que se ha convertido en el estándar *de facto* para las aplicaciones web basadas en NodeJS.
- **AngularJS.**
- **NodeJS.**

Las principales características de Node.js son las siguientes:

- OpenSource, independiente de plataforma.
- Desarrollado sobre CJR v8 (Chrome JavaScript Runtime).
- Diseñado para ser rápido y escalable.
- Permite la ejecución de JavaScript en el backend.
- El contexto de ejecución de CJR v8 es completamente diferente al contexto del navegador, dado que se ejecuta en el lado del servidor.

Otras de las cosas que debería tener en cuenta cuando trabaje con NodeJS son la programación asíncrona y la programación orientada a eventos, con la particularidad de que los eventos, en esta plataforma, se orientan a eventos que suceden del lado del servidor.

Además, NodeJS implementa los protocolos de comunicaciones en redes más habituales, de los usados en Internet, como puede ser el HTTP, DNS, TLS, SSL, etc. Otro aspecto sobre el que está basada NodeJS son los streams. Un stream es un objeto que encapsula un flujo de datos y provee de mecanismos para la escritura o lectura en él.

En nuestro caso, nos centraremos en el framework Express <http://expressjs.com/es/> un framework open source desarrollado por la comunidad de NodeJS, compuesto por un conjunto de módulos, que nos facilita el desarrollo de sitios web Express. Lo que hace es crear una estructura de trabajo para ofrecer las funcionalidades que más se suelen utilizar cuando empieza un proyecto con node; por ejemplo, la creación del servidor, el manejo de las rutas y la carga de todas las dependencias necesarias.

En el caso de aplicaciones desarrolladas con NodeJS, resulta común encontrarnos con que se necesita desarrollar el componente "servidor", típicamente utilizando el módulo *express* en entornos de desarrollo, además de la lógica propiamente dicha de la aplicación.

Sin embargo, la arquitectura también introduce limitaciones que deben tenerse en cuenta. Algunas de las técnicas de ataque mencionadas en este capítulo se encuentran directamente relacionadas con el uso incorrecto de NodeJS, y pueden evitarse fácilmente mediante la comprensión del entorno y la arquitectura.

Un aspecto interesante de NodeJS reside en que, al permitir múltiples peticiones de forma concurrente, esto hará que, en teoría, nos hallemos protegidos ante ataques de denegación de servicio, aunque luego veremos que no es del

todo cierto, ya que hay técnicas utilizando expresiones regulares con las cuales se puede llegar a realizar una denegación de servicios aprovechando una expresión regular mal configurada.

## 5.2 Modelo Event-Loop

NodeJS utiliza un modelo de funcionamiento asíncrono basado en **eventos** y en la ejecución de funciones de callback. Se trata de un framework que ha sido pensado para ejecutar operaciones sobre un único hilo de ejecución; por este motivo, nos encontramos con que el principal objetivo de NodeJS con su modelo single-thread consiste en mejorar el desempeño y escalabilidad de aplicaciones web. Ya no tenemos el modelo one thread per requests utilizado en las aplicaciones web tradicionales, las cuales se despliegan en servidores web como Apache o nginx.

Un modelo Event-Loop resulta ideal para aplicaciones en donde la velocidad de respuesta de las peticiones es más importante para las operaciones de entrada/salida. Nos encontramos ante un modelo completamente distinto, enfocado al desarrollo de aplicaciones escalables, asíncronas y con un bajo consumo de recursos debido a que ahora, sobre el proceso servidor, ya no se hace un fork para atender las peticiones de los clientes en procesos independientes, algo que resulta habitual en servidores web Apache.



Figura 5.2 Presentación de Modelo de eventos en NodeJS.

Dado que todo se ejecuta desde un único hilo de ejecución, la gestión inadecuada de excepciones puede interrumpir completamente el normal funcionamiento del servidor y, por este motivo, se recomienda tener rutinas de comprobación que se encarguen de verificar, en todo momento, el correcto funcionamiento del servidor y la disponibilidad del proceso en el sistema.

### 5.3 Gestión de paquetes

En NodeJS, el código se organiza por medio de módulos. Son como los paquetes o librerías de otros lenguajes como Java. Por su parte, NPM es el nombre del gestor de paquetes (*package manager*) que usamos en Node JS. Básicamente, constituye una forma de administrar módulos que se desea tener instalados, distribuir los paquetes y agregar dependencias a los programas.

NPM es el principal gestor de paquetes de NodeJS, que ha ido evolucionando hasta convertirse en el administrador de paquetes de JavaScript por defecto. NPM ha hecho la programación para Node.js modular: los desarrolladores crean pequeños módulos diseñados para una tarea específica y luego pueden compartir y reutilizar fácilmente este código en diferentes proyectos.

El gestor de paquetes NPM funciona de tal forma que, al descargarse un módulo, se agrega a un proyecto local, que es el que lo tendrá disponible para incluir. Aunque cabe decir que también existe la posibilidad de instalar los paquetes de manera global en nuestro sistema.

El sitio web npmjs <https://www.npmjs.com> es el repositorio oficial donde podemos compartir nuestros módulos y paquetes, y beneficiarnos de los módulos y paquetes que desarrollaron otros programadores. Para hacernos una idea de la cantidad de desarrolladores que están utilizando esta tecnología, podemos decir que hay más de 180 000 paquetes publicados en el sitio hasta la fecha. Existen paquetes con muy diferentes objetivos, por ejemplo, frameworks para sitios web, para acceder a una base de datos MySQL o a la base de datos mongoDB.

Existen distintos módulos que están disponibles de manera predeterminada en cualquier proyecto NodeJS y, por tanto, no necesitamos descargarnos ningún paquete adicional para utilizarlos. Esos toman el nombre de **módulos nativos** y ejemplos de ellos tenemos en “http”, para realizar peticiones; “fs”, para el acceso al sistema de archivos; “net”, para conexiones de red de más bajo nivel que “http”; “url”, que permite realizar operaciones sobre direcciones web; el módulo “util”, que es un conjunto de utilidades; “child\_process”, que te da

herramientas para ejecutar comandos y procesos del sistema, o “domain”, que le permite manejar errores.

Los paquetes o módulos no incluidos en forma nativa en NodeJS los tenemos que descargar a través del gestor de paquetes npm. Este es un poco distinto a otros gestores de paquetes que podemos conocer, como puede ser el de Linux, Fedora o Red Hat, por nombrar algunos, porque los instala localmente en los proyectos; es decir, al descargarse un módulo, se agrega a un proyecto local, que es el que lo tendrá disponible para incluir. Aunque cabe decir que también existe la posibilidad de instalar los paquetes de manera global en nuestro sistema.

Los paquetes que se instalan mediante npm pueden contener vulnerabilidades de seguridad críticas que podrían afectar también a su aplicación. La seguridad de su aplicación es solo tan fuerte como el “eslabón más débil” de sus dependencias. Estos paquetes podrían tener algún tipo de vulnerabilidad y encontrarse expuestos a un posible ataque. Los posibles ataques pueden ser:

- Crear y ejecutar secuencias de comandos en diferentes etapas durante la instalación o uso del paquete.
- Leer, escribir, actualizar y eliminar archivos en el sistema.
- Escribir y ejecutar archivos binarios.
- Obtener datos sensibles de forma remota.

Algunas de las herramientas nos pueden ayudar a analizar la seguridad de los paquetes de terceros que se utilizan como dependencias en nuestros proyectos. El servicio <https://www.npmjs.com/advisories> muestra las últimas vulnerabilidades descubiertas para determinadas versiones de paquetes y el nivel de criticidad.

Security advisories		
Advisory	Date of advisory	Status
Path Traversal restify-swagger-jade <small>Severity: Info</small>	Jul 3rd, 2019	<small>status: published</small>
Cross-Site Scripting jquery.json-viewer <small>Severity: Info</small>	Jul 3rd, 2019	<small>status: published</small>

Figura 5.3 Vulnerabilidades en algunos de los paquetes de npm.

Por ejemplo, en este caso, nos encontramos con error de Path Traversal en la librería `restify-swagger-jsdoc`, cuya solución consiste en actualizar a la nueva versión.

The screenshot shows the NPM security page for the package `restify-swagger-jsdoc`. At the top, there's a button labeled "View on GitHub". Below it, the title "Path Traversal" is displayed next to the package name "restify-swagger-jsdoc". There are two tabs at the bottom: "Advisory" (which is selected) and "Versions". On the right side, there's a vertical "Advisory timeline" section with two entries:

- Published**: Advisory Published Jul 16th, 2019
- Reported**: Reported by [xylesof](#) Jul 3rd, 2019

Below the timeline, there's an "Overview" section containing a detailed description of the vulnerability, followed by a "Remediation" section with instructions to upgrade to version 3.2.1 or later.

Figura 5.4 Vulnerabilidades en el paquete `restify-swagger-jsdoc`.

## 5.4 Programación asíncrona

La filosofía detrás de Node.js reside en hacer programas que no bloquen la línea de ejecución de código con respecto a entradas y salidas, de modo que los ciclos de procesamiento se queden disponibles cuando se está esperando a que se completen tales acciones. Esto se implementa a partir de callbacks, que son funciones que se indican con cada operación I/O y que se ejecutan cuando las entradas o salidas se han completado.

Estos callbacks son, más o menos, como los que podemos conocer de otros frameworks JavaScript, como jQuery; funciones que se ponen en ejecución cuando terminan de ejecutarse otras acciones; por ejemplo, con este código:

```
console.log("start");
fs.readFile("x.txt", function(error, archivo){
  console.log("fichero leido");
})
console.log("end");
```

Realmente JavaScript es síncrono y ejecuta las líneas de código una detrás de otra, pero, por la forma de ejecutarse el código, hace posible la programación asíncrona. La segunda instrucción que hace la lectura del archivo tardará un tiempo en ejecutarse y en ella indicamos además una función con un console.log ("fichero leido"); esa es la función callback que se ejecutará solamente cuando termine la lectura del archivo. A continuación, y antes de que se llegue a terminar la lectura del archivo, ejecuta la instrucción con el último console.log().

## 5.5 Problema del código piramidal

El uso intensivo de callbacks en la programación asíncrona produce el poco deseable efecto de código piramidal. Al utilizarse los callbacks, se meten unas funciones dentro de otras y se va entrando en niveles de profundidad que hacen un código menos sencillo de mantener.

La solución se basa en hacer un esfuerzo adicional por estructurar nuestro código que, básicamente, trata de modularizar el código de cada una de las funciones, para escribirlas aparte y, al indicar la función de callback, en vez de escribir el código, escribimos el nombre de la función que se ha definido aparte.

Al conseguir niveles de indentación menos profundos, estamos ordenando el código, con lo que será más sencillo de entender y será también más fácil encontrar posibles errores. Además, a la larga, conseguirá que sea más escalable y pueda extenderlo o mantenerlo en el futuro.

Algunos consejos a la hora de escribir código para que este sea de mayor calidad:

- Escriba código modularizado; por ejemplo, un fichero con más de 500 líneas de código probablemente haya que modularizarlo en varios ficheros.

- Use librerías como `async`, que ayuden al control de la ejecución de callbacks.
- Utilice promesas (`promise` en Node) y futuros (`future` en Node).

## 5.6 Módulo para administrar el sistema de archivos

El módulo de administración de archivos “`fs`” implementa la programación asíncrona para procesar su creación, lectura, modificación, borrado, etc. El módulo “`fs`” tiene muchas funciones, como crear, leer, borrar y renombrar archivos; crear directorios; borrar directorios; retornar información de archivos, etc. Para consultar estas funciones, podemos visitar el API de Node.js: <https://nodejs.org/api/fs.html>

En el siguiente ejemplo vemos el uso de una de las funciones de este módulo para crear un archivo:

```
var fs = require('fs');
fs.writeFile('./archivo1.txt','línea 1\nLínea 2',function(error){
if (error)
  console.log(error);
else
  console.log('El archivo fue creado');
});
console.log('última linea del programa');
```

Debido a la programación asíncrona, podemos ver que se muestra el mensaje ‘última línea del programa’ antes de mostrarnos que el archivo fue creado; es decir, que, cuando llamamos a la función `writeFile`, el script no se detiene en esta línea hasta que el archivo se crea, sino que continúa con las siguientes instrucciones.

En este script en particular no conlleva grandes ventajas utilizar la programación asíncrona, ya que, luego de llamar a la función `writeFile`, solo procedemos a mostrar un mensaje por la consola, pero, en otras circunstancias, podríamos estar ejecutando más actividades que no dependieran de la creación de dicho archivo. Llamamos a la función `writeFile` a través de la variable `fs`. Esta función posee tres parámetros:

- El primer parámetro es el nombre del archivo de texto que se vaya a crear. Indicamos el path donde debe crearse el fichero.
- El segundo parámetro representa la cadena que se desea almacenar en el archivo de texto (mediante los caracteres \n, generamos el salto de linea en el archivo de texto).
- Finalmente, el tercer parámetro es una función anónima que será llamada desde el interior de la función writeFile, cuando haya terminado de crear el archivo.
- La condición comprueba si en la variable error viene un objeto distinto de vacío. En caso de que se produzca un error, se muestra este por consola, y en caso de que no tengamos error, se ejecuta el bloque del else donde mostramos por consola información correspondiente al archivo que se ha creado.

Podemos crear un script para leer el contenido del archivo que creamos con el ejemplo anterior:

```
var fs = require('fs');
fs.readFile('./archivo1.txt',function(error,datos){
if (error) {
console.log(error);
}
else {
console.log(datos.toString());
}
});
console.log('última linea del programa');
```

Para **mostrar el contenido del fichero** en formato texto, llamamos al método `toString()`. Si no hacemos esto, en pantalla se mostrarán los valores numéricos de los caracteres. El empleo de funciones anónimas en JavaScript resulta muy común, pero podemos volver a codificar el problema anterior pasando el nombre de una función:

```
var fs = require('fs');
function leer(error,datos){
if (error) {
console.log(error);
}
else {
console.log(datos.toString());
}
}
fs.readFile('./archivo1.txt',leer);
console.log('última linea del programa');
```

## 5.7 Módulo http

Http es un módulo del core de Node.js e implementado en C para una mayor eficiencia en las conexiones web. Podemos utilizar este módulo para la implementación de un servidor web. Si analizamos el siguiente código, primero requerimos o importamos el módulo 'http' y guardamos una referencia en la variable http.

```
var http = require('http');
var servidor = http.createServer(function(request,response){
response.writeHead(200, {'Content-Type': 'text/html'});
response.write('<!doctype html><html><head></head>' +
'<body><h1>Sitio en desarrollo</h1></body></html>');
response.end();
});
servidor.listen(8888);
console.log('Servidor web iniciado');
```

El módulo 'http' dispone de una función llamada **createServer** que tiene por objetivo crear un servidor que implementa el protocolo HTTP. A la función **createServer** debemos enviarle una función anónima con dos parámetros que los hemos llamado "pedido" y "respuesta".

Los objetos "pedido" y "respuesta" los crea la misma función **createServer** y los pasa cuando se dispara el pedido de una página u otro recurso al servidor. De la misma forma que cuando hemos creado un fichero utilizando programación asincrónica, la función **createServer** se ejecuta en forma asincrónica, lo

que significa que no se detiene, sino que sigue con la ejecución de la siguiente función, que aparece en el código `servidor.listen(8888);`

La función `listen()`, que también es asíncrona, se queda esperando a recibir peticiones. Antes de poder solicitar una página desde el navegador, podemos ver en la consola el mensaje de “Servidor web iniciado”. El script, como podemos observar desde la consola, no ha finalizado, sino que está ejecutándose en un bucle infinito en la función `listen`, esperando peticiones de recursos.

Cuando hay una solicitud de recursos al servidor, se dispara la función anónima, y llegan dos objetos como parámetro:

```
var servidor = http.createServer(function(request,response){
  response.writeHead(200, {'Content-Type': 'text/html'});
  response.write('<!doctype html><html><head></head>' +
    '<body><h1>Sitio en desarrollo</h1></body></html>');
  response.end();
});
```

El primer parámetro contiene, entre otros datos, la `request` que realizamos e información del navegador que hace la petición. El parámetro `response` lo utilizamos para llamar a los métodos:

- **writeHead**: Sirve para indicar la cabecera de la petición HTTP (en este caso, indicamos con el código 200 que la petición fue Ok y, con el segundo parámetro, inicializamos la propiedad `Content-Type`, indicando que devolvemos contenido HTML).
- **write**: Mediante la función `write`, indicamos todos los datos propiamente dichos del recurso para devolver al cliente (en este caso, indicamos en la cabecera de la petición que se trata de HTML).
- **end**: Finalmente, llamando a la función `end`, indicamos que hay que devolver la respuesta con el contenido que haya en este momento.

Si detenemos el servidor que creamos desde la consola (presionando las teclas `Ctrl+C`, que aborta el programa) y solicitamos nuevamente datos al servidor, podremos ver que ahora el servidor no responde. Es importante entonces tener en cuenta que el programa `Node.js` esté ejecutándose, para poder pedirle recursos. Otra cosa importante estriba en que, cada vez que hagamos cambios en el código fuente de nuestra aplicación en `JavaScript`, debemos detener y volver a lanzar el programa, para que tenga en cuenta las modificaciones.

## 5.8 Utilización del Middleware Express

Express es una infraestructura web de direccionamiento y Middleware viene definida con una serie de funcionalidades donde, básicamente, una aplicación Express es una serie de llamadas a funciones de Middleware.

Un Middleware o lógica de intercambio de información entre aplicaciones es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones o paquetes de programas, redes, hardware o sistemas operativos. El Middleware simplifica el trabajo de los desarrolladores en la compleja tarea de generar las conexiones y sincronizaciones necesarias en los sistemas distribuidos. De esta forma, se provee de una solución que mejora la calidad de servicio al nivel de envío de mensajes entre aplicaciones, así como su seguridad.

Por ejemplo, los elementos Middleware permiten que se pueda acceder a los datos contenidos en una base de datos a través de otra, lo que ahorra tiempo a los desarrolladores. Las funciones de Middleware son funciones que tienen acceso al objeto de solicitud (*request*), al objeto de respuesta (*response*) y a la siguiente función de Middleware en el ciclo de solicitud/respuestas de la aplicación. Las funciones de Middleware pueden realizar las siguientes tareas:

- Ejecutar cualquier código.
- Realizar cambios en los objetos de petición y de respuesta.
- Finalizar el ciclo de solicitud/respuestas.
- Invocar la siguiente función de Middleware en la pila de llamadas.

Una aplicación Express puede utilizar los siguientes tipos de Middleware:

- Middleware de nivel de aplicación
- Middleware de nivel de direccionamiento
- Middleware de manejo de errores
- Middleware de terceros

### 5.8.1 Middleware de nivel de aplicación

Un ejemplo de una función de Middleware de nivel de aplicación es la función `use`, que vemos en el siguiente ejemplo. La función se ejecuta cada vez que la aplicación recibe una petición http.

<http://expressjs.com/es/guide/using-middleware.html#middleware.application>

```
var app = express();
app.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});
```

### 5.8.2 Middleware de nivel de direccionamiento

El Middleware de nivel de direccionamiento funciona de la misma manera que el Middleware de nivel de aplicación, excepto que está enlazado a una instancia de la clase `express.Router()`:

<http://expressjs.com/es/guide/using-middleware.html#middleware.router>

```
var router = express.Router();
```

### 5.8.3 Middleware de terceros

Utilizamos el Middleware de terceros para añadir funcionalidad a las aplicaciones Express. El siguiente ejemplo muestra la instalación con npm del paquete `cookie-parser` y, posteriormente, cargamos la función de Middleware de gestión de cookies:

<http://expressjs.com/es/guide/using-middleware.html#middleware.third-party>

```
$ npm install cookie-parser
var express = require('express');
var app = express();
var cookieParser = require('cookie-parser');
// cargamos el middleware
app.use(cookieParser());
```

Para más información, podemos consultar la documentación oficial:

<http://expressjs.com/es/guide/using-middleware.html>

## 5.9 Autenticación en NodeJS

### 5.9.1 Auth0

Auth0 es una plataforma de identidad universal con la cual podemos añadir autenticación a nuestro sitio web o aplicación móvil de una forma sencilla. Pueden obtener más información sobre esta herramienta en el sitio de Auth0: <https://auth0.com>.

En la dirección <https://auth0.com/docs/server-platforms/nodejs>, hay disponible un ejemplo del uso de Auth0 con una aplicación web en NodeJS que pueden descargar y seguir los pasos para verla funcionando, o pueden descargar el proyecto y añadirlo a alguna aplicación que ya tengan. Se puede encontrar un ejemplo de integración en el repositorio de GitHub <https://github.com/auth0-samples/auth0-nodejs-webapp-sample>

Entre las principales características que proporciona el framework, podemos destacar:

- Permite añadir autenticación con múltiples fuentes de autenticación, ya sean redes sociales como Google, Facebook, cuenta de Microsoft, LinkedIn, GitHub, Twitter u otros sistemas de identidad empresarial, como Windows Azure AD, Google Apps o Active Directory.
- Permite añadir autenticación a través de bases de datos mediante nombre de usuario/contraseña.
- Permite añadir soporte para vincular diferentes cuentas de usuario con la misma identidad de usuario.
- Ofrece soporte para generar *json web tokens* firmados para llamar a sus API y hacer que la identidad del usuario se realice de forma segura.
- Permite obtener datos de otras fuentes y añadirlos al perfil de usuario, a través de las reglas de JavaScript.

### 5.9.2 PassportJS

Passport <http://passportjs.org> es un Middleware de autenticación para Node.js. Passport se puede agregar en cualquier aplicación web basada en Express. Incluye un conjunto completo de estrategias de autenticación utilizando un nombre de usuario y contraseña, así como integración con redes sociales como Facebook y Twitter.

El módulo passport-auth0 <https://www.npmjs.com/package/passport-auth0> permite usar la autenticación que proporciona auth0. En el repositorio de GitHub <https://github.com/auth0/passport-auth0>, podemos encontrar más información al respecto.

## 5.10 OWASP top 10 en NodeJS

NodeJS es una tecnología para el mundo web; por este motivo, todas las amenazas descritas en el OWASP top 10 aplican, igualmente, a las aplicaciones basadas en Node.js y nos encontramos con los mismos problemas que afectan a las aplicaciones escritas en otros lenguajes PHP, JSP/JSF/J2EE: desde problemas típicos de inyección (SQLi, LDAPI o XSS) hasta problemas de fugas de información o ejecución remota de código. En el caso de Node.js, podemos encontrar las siguientes amenazas, incluidas en el OWASP top 10, y suelen ser bastante comunes en aplicaciones de este tipo. A continuación se enumeran algunas de ellas.

### 5.10.1 OWASP NodeGOAT

En esta sección explicaremos, a grandes rasgos, las principales vulnerabilidades que pueden afectar a una aplicación escrita en NodeJS por medio de una aplicación web vulnerable por diseño, el proyecto NodeGoat de OWASP, el cual se encuentra disponible en el siguiente repositorio de GitHub: <https://github.com/OWASP/NodeGoat>

Para su instalación, basta con descargar el proyecto del repositorio y, posteriormente, realizar la instalación de todos los módulos necesarios, para que la aplicación pueda arrancar. Dichos módulos se instalan rápidamente con el comando `npm install`. Evidentemente, es necesario tener Node.js instalado en el sistema, en el caso de sistemas basados en Debian. Resulta tan sencillo como ejecutar el comando `apt-get install nodejs`.

En la aplicación existen varias vulnerabilidades de inyección del tipo XSS. Una de ellas se encuentra en el perfil del usuario, en donde se puede editar información básica del usuario que se encuentra autenticado. Dado que algunos campos de entrada no gestionan adecuadamente las validaciones y no “escapan” caracteres que son potencialmente peligrosos, resulta fácil incluir un script que haga cualquier cosa, desde el típico “alert” hasta cargar un iframe que cargue código malicioso.

Además de la sección de perfil del usuario, ubicado en el extremo superior derecho de la aplicación web, en la opción “memos” es posible escribir un mensaje, que quedará almacenado en la base de datos y el cual admite, entre otras cosas, tags HTML, que serán renderizados directamente en el navegador web de cada uno de los usuarios que accedan a la página, lo que nuevamente da como resultado una vulnerabilidad del tipo XSS almacenado.

Otra vulnerabilidad que también resulta fácil de descubrir se encuentra en la sección “allocations”. La lógica de esta página indica que cada uno de los usuarios autenticados accede a sus propios “allocations”; sin embargo, la aplicación recibe por parámetro el identificador de cada “allocation” y permite realizar una búsqueda directa contra la base de datos.

En este caso, nos encontramos con una vulnerabilidad de **“Insecure direct object reference”**, definida en el OWASP top 10, ya que no se está validando si el usuario que realiza la petición cuenta con permiso para acceder al elemento solicitado. Aquí, simplemente basta con cambiar la dirección url “/allocations/2” por “/allocations/xxx”. Veremos entonces cómo cambia la página de perfil y podremos ver los datos de usuario con otro identificador.

The screenshot shows a web browser window with the following details:

- Address Bar:** https://nostalgiaethicalhacker.com/allocations/2
- Title Bar:** @RetireEasy Employee Retirement Savings Management
- Left Sidebar (Menu):**
  - Dashboard
  - Allocations
  - Assets
  - Profile
  - Learning Resources
  - Logout
- Main Content Area:**
  - Section: Filter Assets based on Stock Performance:**
    - Enter Threshold
    - (Using active threshold value, it will return all assets allocation above the specified stocks percentage number)
    - Submit**
  - Section: Asset Allocations for ZAP ZAP:**
    - Demand Stocks: 17 %
    - Funds: 18 %
    - Bonds: 45 %

Figura 5.5 Vulnerabilidad Insecure direct object reference en la ruta “allocations”.

En la sección que pone “Learning Resources”, existe un parámetro llamado “url”, el cual recibe como argumento una url que es utilizada por la aplicación web para realizar una redirección. Esta conforma otra vulnerabilidad fácil de

descubrir, dado que el parámetro puede ser modificado e incluir un dominio arbitrario; de este modo, se controla la navegación del usuario:

<http://nodegoat.herokuapp.com/learn?url=https://www.khanacademy.org/economics-finance-domain/core-finance/investment-vehicles-tutorial/ira-401ks/v/traditional-iras>

Hasta aquí se han visto las vulnerabilidades más sencillas en NodeGoat; sin embargo, hay otras más interesantes que podríamos investigar. Concretamente, a la hora de realizar una auditoría de código en una aplicación con esta tecnología, se deben buscar los siguientes patrones:

- Uso de la función “eval” recibiendo entradas por parte del usuario sin validar.
- Uso de la función “setInterval” recibiendo entradas por parte del usuario sin validar.
- Uso de la función “setTimeout” recibiendo entradas por parte del usuario sin validar.
- Creación de una función anónima partiendo de instrucciones recibidas por parte del usuario y sin validar.

Se trata de una aplicación donde encontramos la mayoría de las vulnerabilidades que vamos a comentar. La aplicación se encuentra disponible de forma pública en la Url <http://nodegoat.herokuapp.com/login>

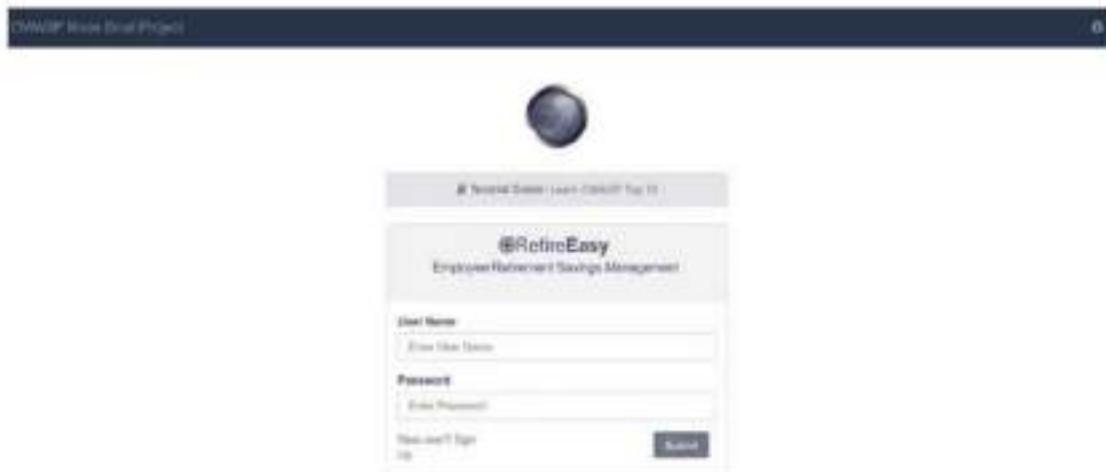


Figura 5.6 Página de login de NodeGoat.

### 5.10.2 Inyección de código

La inyección es un vector de ataque donde el atacante introduce código malicioso en la aplicación para engañar al programa para que lo ejecute. En general, el usuario envía datos específicos a la aplicación, que se pasan al intérprete para que los ejecute. Si la aplicación no valida correctamente los datos, se ejecutarán los comandos enviados.

Con la inyección de código, el atacante puede hacer que el servidor haga algo diferente a lo que se supone que debe hacer. Eso significa obtener información confidencial, realizar una denegación de servicio o incluso modificar el servicio en sí mismo.

The screenshot shows the OWASP Node.js Tutorial: Fixing OWASP Top 10 page. The main navigation bar has 'A1 INJECTION' selected. Below it, there's a sidebar with links to other sections: A2 Broken Authentication, A3 XSS, A4 Broken Object Reference, A5 Broken Access Control, A6 Session Hijacking, A7 Cross-Site Scripting (XSS), A8 Cross-Site Request Forgery, and A9 Security Misconfiguration. The main content area is titled 'A1 - Injection' and includes tabs for 'Exploitability: EASY', 'Likelihood: COMMON', 'Consequence: HIGH', and 'Technical Impact: 100%'. The 'Description' section states: 'Injection flaws occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or data manipulations.' The 'VAT: 1 Server Side JS injection' section provides a detailed description of how certain methods like eval(), setInterval(), setTimeout(), and clearTimeout() can be exploited if they process user-provided inputs. It also describes 'Attack Mechanics' for various applications using JavaScript's eval() function to parse incoming data without validation, such as setInterval([...], interval). The attack can take advantage of the first argument being a string to execute arbitrary code. The vulnerability can be critical if it allows attackers to send various types of commands.

Figura 5.7 Inyección de código en OWASP.

Existen varias subcategorías de ataques de inyección, dependiendo del entorno de ejecución de destino. Debido a que hay muchas formas de usar ataques de inyección, el proyecto OWASP clasifica estos ataques, especialmente la inyección de SQL, como el vector de ataque número uno para las aplicaciones web.

Las inyecciones de código se dirigen a las aplicaciones donde la funcionalidad se crea e interpreta durante el tiempo de ejecución en función de las aportaciones del usuario. Esto hace que encontrar posibles puntos de ataque resulte sencillo.

En NodeJS, un ejemplo directo de inyección de código puede ser la presencia de una llamada a la función `eval()`, utilizando como argumento una cadena de texto cuyo contenido es parcialmente controlado por el usuario.

Las vulnerabilidades de inyección ocurren cuando se envían datos no confiables a un intérprete como parte de un comando o consulta. Un atacante puede engañar al intérprete para que ejecute comandos no intencionados o acceda a los datos sin la debida autorización.

Las funciones de JavaScript como `eval()`, `setTimeout()` o `setInterval()` se usan para procesar entradas proporcionadas por el usuario. En este punto, un atacante podría aprovecharse de estas funciones para inyectar y ejecutar código JavaScript malicioso en el servidor.

Las aplicaciones web que utilizan la función `eval()` de JavaScript para analizar los datos de entrada sin realizar ningún tipo de validación normalmente se muestran vulnerables a este ataque. Un atacante puede inyectar código JavaScript arbitrario para ser ejecutado en el servidor. De manera similar, las funciones `setTimeout()` y `setInterval()` pueden tomar código en formato de cadena como un primer argumento que causa los mismos problemas que `eval()`.

#### 5.10.3 Función eval

Otra característica interesante de "eval" reside en que, además de validar valores numéricos, también se encarga de la validación y ejecución de instrucciones JavaScript. Esto significa que, conociendo la API de NodeJS, sería posible incluir cualquier tipo de instrucción que permita la ejecución arbitraria de código.

Si un atacante fuese capaz de manipular la respuesta de un servicio, podría inyectar código arbitrario mediante la llamada a `eval`, que ejecutaría el código en el contexto del navegador de la víctima. El ataque podría, en ese momento, extraer información de autenticación, cookies de sesión, manipular el dom para inducir al usuario a introducir datos o utilizar el navegador para realizar peticiones arbitrarias al servidor del dominio como si fuese el usuario.

```
function upload(response, postData) {
    console.log("request for 'upload' was called");
    var result = eval("(" + querystring.parse(postData).text + ")");
    response.writeHead(200, {"Content-Type": "text/html"});
    response.write("you sent: " + result);
    response.end();
}
```

Figura 5.8 Uso inseguro de la función eval.

La función “eval” lo que hace por debajo es ejecutar las instrucciones enviadas por parámetro de la función, lo que, en algunos casos, puede producir problemas de inyección de código si los valores enviados a “eval” provienen de una petición por parte del cliente y no se validan correctamente:

```
// Uso inseguro de eval() para analizar entradas  
var input = eval(request.body.input);
```

Este tipo de casos se pueden solucionar simplemente usando parseInt() en lugar de eval:

```
// Uso seguro de parseInt() para analizar entradas  
var input = parseInt(request.body.input);
```

#### 5.10.4 Ataque de denegación de servicio

Un ataque de denegación de servicio puede ejecutarse simplemente enviando los comandos a continuación de la función eval(): **while(1)**.

Esta entrada hará que el bucle de eventos del servidor de destino utilice el 100 % del tiempo del procesador y no pueda aceptar ninguna otra petición en el servidor hasta que se reinicie el proceso. Un ataque DoS alternativo podría ser enviando la instrucción process.exit() o matar el proceso en ejecución con el comando process.kill(process.pid).

#### 5.10.5 Uso de patrones y expresiones regulares

El uso de las expresiones regulares resulta bastante habitual para validar patrones y se utiliza con bastante frecuencia a la hora de comprobar los valores ingresados por un usuario en formularios web.

Las expresiones regulares ofrecen una manera de comprobar si los datos coinciden con un patrón específico. Cuando un usuario se registra por primera vez en una aplicación web, algunas de las primeras piezas de datos requeridos son un nombre de usuario, una contraseña y una dirección de correo electrónico. Si esta entrada proviene de un usuario malicioso, la entrada podría contener cadenas de ataque. Al validar la entrada del usuario para asegurarnos de

que cada dato contiene solo el conjunto válido de caracteres, podemos dificultar el ataque a esta aplicación web.

Analicemos la siguiente expresión regular para el nombre de usuario: `^a-zA-Z{3,16}$`.

Esta expresión regular solo permite letras minúsculas, números y el carácter de subrayado. El tamaño del nombre de usuario también se está limitando a 3-16 caracteres en este ejemplo.

Aquí hay un ejemplo de expresión regular para el campo de contraseña:

```
^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@#$%]).{10,20}$
```

Dicha expresión regular asegura que una contraseña se compone de 10 a 20 caracteres de longitud e incluye una letra mayúscula, una letra minúscula, un número y un carácter especial (uno o más usos de @, #, \$ o %). Aquí hay un ejemplo de expresión regular para una dirección de **correo electrónico**:

```
^[a-zA-Z0-9.!#$%&'^+/_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$
```

Se debe tener cuidado al crear expresiones regulares, ya que las expresiones mal diseñadas pueden resultar en condiciones potenciales de denegación de servicio, también conocidas como **Regex DoS**. Un análisis estático o una herramienta que permita evaluar expresiones regulares pueden ayudar a los equipos de desarrollo a detectar este tipo de casos.

Aunque se trata de elementos muy potentes, se caracterizan por ser de alto consumo en términos de recursos y tiempo de procesamiento. Puede que dichas expresiones funcionen correctamente, pero, debido a la forma en la que se encuentran declaradas, consuman más tiempo y recursos de lo que deberían. En el caso de NodeJS y, especialmente, en cualquier plataforma que siga el modelo Event-Loop, con un único proceso asociado al servidor, podría producirse un cuello de botella considerable, lo que, al final, termina por provocar una condición de denegación de servicio.

Se recomienda seguir un conjunto de buenas prácticas a la hora de crear y utilizar expresiones regulares, para evitar que node gaste demasiado tiempo y memoria evaluando datos que no cumplen la expresión regular:

- Comprobar toda la entrada utilizando ^ para el comienzo de la entrada y \$ para el final de la entrada.
- Evitar evaluar el mismo prefijo o sufijo en distintos caminos; por ejemplo,  $^((p)algo(s)) | ((p)otro(s))\$$  lo podríamos transformar en  $^((p)(algo | otro)(s))\$$ .
- Evitar las variaciones de variaciones; por ejemplo,  $(a | b+)^*$ .
- Evitar la búsqueda de determinados caracteres; por ejemplo,  $(?!.*[\%\{\}\|\^|=}]>\{<\$])$ .
- Realizar pruebas con entradas que concuerden y también con las que no concuerden y encontrar un balance en el número de pasos que se necesitan en ambos.
- No introducir, por parte del usuario, la expresión regular que se utilice en el servidor.

Se recomienda usar el módulo **safe-regex** <https://github.com/substack/safe-regex> para determinar si una expresión regular se muestra vulnerable, siempre teniendo en cuenta que este tipo de herramientas suelen ser propensas a falsos positivos. Si ejecutamos este módulo contra una expresión regular, nos devuelve true si es segura y false en caso contrario:

```
$ node .\npm\node_modules\safe-regex\example\safe.js (([a-z])+.)+[A-Z]([a-z])+
false

$ node .\npm\node_modules\safe-regex\example\safe.js ^[A-Z]([a-z])+
True
```

#### 5.10.6 Acceso al sistema de ficheros

Otro objetivo potencial de un atacante podría residir en leer el contenido de los archivos del servidor; por ejemplo, los siguientes dos comandos enumeran el contenido del directorio actual y del directorio raíz, respectivamente:

```
response.end(require('fs').readdirSync('.').toString())
response.end(require('fs').readdirSync('..').toString())
```

Una vez que se obtienen los nombres de los archivos, un atacante puede ejecutar el siguiente comando, para ver el contenido real de un archivo:

```
response.end(require('fs').readFileSync(filename))
```

Para evitar ataques de inyección a nivel de servidor, se recomienda seguir algunas buenas prácticas:

- Validar las entradas del usuario en el lado del servidor antes de procesarlas.
  - No usar la función eval() para analizar las entradas del usuario y evitar utilizar otros comandos con efectos similares, como setTimeOut() y setInterval().
  - Para analizar las entradas en formato JSON, en lugar de emplear eval(), podríamos recurrir a una alternativa más segura como JSON.parse().
  - Incluya “uso estricto” al principio de una función en JavaScript, que habilita el modo estricto dentro del alcance de la función de callback.

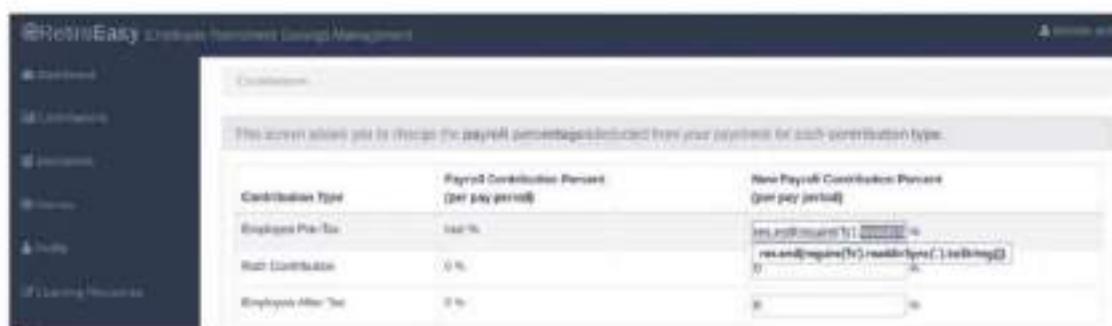


Figura 5.9 Vulnerabilidad que permite conocer los archivos.

Podriamos llegar a averiguar los **módulos de node que usa la aplicación**; para ello, podemos inyectar el comando:

```
res.end(require('fs').readdirSync('node_modules').toString())
```

Figura 5.10. Módulos que usa la aplicación alojados en node\_modules

Con una vulnerabilidad de este tipo, no solamente resulta posible producir una condición de denegación de servicio o crear una shell contra el sistema; también se puede, con muy pocas líneas de código, leer directorios y ficheros de forma arbitraria en el sistema de archivos.

Incluyendo una instrucción como `response.end(require('fs').readdirSync('.').toString())`, se podrían obtener los ficheros y directorios del directorio raíz de la aplicación web.

#### 5.10.7 Inyección de SQL

Las inyecciones de SQL y NoSQL permiten a un atacante injectar código en la consulta que ejecutaría la base de datos. Esta vulnerabilidad se introduce cuando los desarrolladores crean consultas de base de datos de forma dinámica que incluyen información proporcionada por el usuario. Aquí hay un ejemplo de ataque equivalente en ambos casos, donde el atacante consigue recuperar el registro del usuario administrador sin saber la contraseña.

Consideremos una declaración SQL de ejemplo utilizada para autenticar al usuario con nombre de usuario y contraseña:

```
SELECT * FROM accounts WHERE username = '$username' AND password = '$password'
```

Si esta declaración no está preparada o no se maneja adecuadamente cuando se construye, un atacante puede proporcionar el valor `admin --` en el campo de nombre de usuario, con el fin de acceder a la cuenta del usuario administrador sin pasar por la condición que verifica la contraseña. La consulta SQL resultante se vería así:

```
SELECT * FROM accounts WHERE username = 'admin' -- AND password = ''
```

Para evitar la inyección, se recomienda seguir unas buenas prácticas, entre las que podemos destacar:

- Se deberán validar todos los parámetros de entrada a la aplicación que vayan a ser empleados en la generación de sentencias SQL. Se habrá de utilizar como mecanismo de validación el basado en sentencias preparadas para cada uno de los parámetros de entrada.

- Se deberán implementar dichos controles de validación en la parte servidora de la aplicación, ya que NUNCA se ha de confiar en los datos provenientes de partes clientes.
- Se deberá aplicar el principio de mínimo privilegio a través de roles para el usuario de la BBDD que ejecute la sentencia SQL.

Por ejemplo, si estamos trabajando con MySQL, este sería un ejemplo donde estamos haciendo un buen uso de la sentencia SQL:

```
var username = '<nombre_usuario>';
var password= '<password_usuario>';
var mysql = require('mysql');
var pool = mysql.createPool(...);
pool.getConnection(function(err, connection) {
  connection.query('SELECT * FROM accounts WHERE username = ? and password =?', [username],[password],
  function(err, results) {
    // ...
  });
});
```

#### 5.10.8 Inyección de NoSQL

El equivalente de la consulta anterior para la base de datos NoSQL MongoDB es:

```
db.accounts.find({username: username, password: password});
```

En este caso, un atacante puede lograr los mismos resultados que la inyección de SQL al proporcionar el objeto de entrada JSON, como se muestra a continuación:

```
{
  "username": "admin",
  "password": { $gt: "" }
}
```

En MongoDB, `$gt` selecciona aquellos documentos donde el valor del campo se muestra mayor que el valor especificado. Por lo tanto, en la declaración anterior, se compara la contraseña en la base de datos con una cadena vacía, que devuelve verdadero.

La aplicación NodeGOAT resulta vulnerable a la inyección de NoSQL; por ejemplo, en la página Asignaciones, al realizar una búsqueda con una entrada maliciosa, `'1'; return 1 == '1'`, se recuperan asignaciones para todos los usuarios en la base de datos.

La inyección de JavaScript del lado del servidor es un ataque donde el código JavaScript se inyecta y se ejecuta en un componente del servidor. Específicamente, MongoDB se muestra vulnerable a este ataque cuando las consultas se ejecutan sin la validación adecuada.

El operador `$where` realiza la evaluación de expresiones de JavaScript en el servidor MongoDB. Si el usuario puede inyectar código directo en dichas consultas, entonces ocurriría un ataque de este tipo; por ejemplo:

```
db.allocationsCollection.find({ $where: "this.userId == " + parsedUserId + " && " + "this.stocks > " + "" + threshold + ""});
```

El código anterior devolverá aquellos documentos que tengan un campo `userId`, como se especifica en el parámetro `parsedUserId`, y un campo de acciones, según lo especificado por la variable `threshold`. El problema lo tenemos si estos parámetros no están validados o filtrados de forma correcta, por lo que serán vulnerables a la inyección de JavaScript del lado del servidor.

A continuación mostramos algunas medidas para evitar ataques de inyección de SQL/NoSQL o minimizar su impacto:

- **Sentencias preparadas:** Para llamadas a SQL, la mejor opción estriba en utilizar sentencias preparadas, en lugar de construir consultas dinámicas utilizando concatenación de cadenas.
- **Validación de las entradas:** Consiste en validar entradas para detectar inputs maliciosos. Para las bases de datos NoSQL, se recomienda también validar los tipos de entrada contra los tipos esperados.
- **Principio de mínimo privilegio:** Para minimizar el daño potencial de un ataque de inyección, se aconseja no asignar permisos de acceso de tipo administrador a las cuentas de usuario de la aplicación.

### 5.10.9 Inyección de logs

Las vulnerabilidades de inyección de log permiten a un atacante acceder y manipular los logs de una aplicación. Un atacante puede crear una solicitud que hará fallar la aplicación y esta acción se registrará en el log. Las vulnerabilidades pueden variar según la función de registro.

Consideremos un ejemplo en el que una aplicación registra un intento fallido de iniciar sesión en el sistema. Un ejemplo muy común para esto es el siguiente:

```
var userName = req.body.userName;
console.log('Error: attempt to login with invalid user: ', userName);
```

Un atacante podría crear una entrada maliciosa con el objetivo de realizar un ataque al sistema de registro de logs; por ejemplo, si una aplicación tiene una aplicación web de back-office que administra la visualización y el seguimiento de logs, entonces, un atacante puede realizar el registro de un log aprovechando una vulnerabilidad del tipo cross-site scripting (XSS). En este punto, la mejor práctica reside en validar la entrada del usuario; por ejemplo, podríamos utilizar el módulo `node-esapi` <https://www.npmjs.com/package/node-esapi> para la codificación de la entrada del usuario:

```
// Paso 1: Requerir un módulo que soporte codificación
var ESAPI = require('node-esapi');
// Paso 2: codificar la entrada del usuario que se registrará en el contexto correcto
console.log('Error: attempt to login with invalid user: %s', ESAPI.encoder().encodeForHT-
ML(userName));
console.log('Error: attempt to login with invalid user: %s', ESAPI.encoder().encodeForJa-
vaScript(userName));
console.log('Error: attempt to login with invalid user: %s', ESAPI.encoder().encodeForU-
RL(userName));
```

### 5.10.10 Gestión de la sesión y autenticación

En este caso, un atacante, que puede ser un atacante externo anónimo o un usuario con cuenta propia, usa fugas o vulnerabilidades en las funciones de autenticación o gestión de sesión para hacerse pasar por otros usuarios. Las funciones de la aplicación relacionadas con la autenticación y la administración de sesiones a menudo no se implementan correctamente, lo que

permite a los atacantes comprometer contraseñas, claves o tokens de sesión o explotar errores de implementación para asumir las identidades de otros usuarios.

La gestión de sesiones constituye una pieza crítica de la seguridad de la aplicación. Comporta un riesgo más amplio y requiere que los desarrolladores se encarguen de proteger la identificación de la sesión, el almacenamiento seguro de las credenciales del usuario, la duración de la sesión y la protección de los datos críticos de la sesión en tránsito. En este punto, podemos destacar, a grandes rasgos, los siguientes **escenarios de ataque** basados en el uso incorrecto de la sesión por parte del desarrollador:

- **Escenario 1:** Los tiempos de expiración de sesión de la aplicación no se configuran correctamente. En lugar de seleccionar “cerrar sesión”, el usuario, simplemente, cierra la pestaña del navegador. El atacante usa el mismo navegador unas horas después y ese navegador aún está autenticado.
- **Escenario 2:** El atacante actúa como un intermediario y adquiere la identificación de sesión del usuario a partir del análisis del tráfico de la red. Luego utiliza este identificador de sesión autenticado para conectarse a la aplicación sin necesidad de ingresar el nombre de usuario y la contraseña.
- **Escenario 3:** Un atacante externo obtiene acceso a la base de datos de contraseñas del sistema. Las contraseñas de los usuarios se encuentran en texto plano, lo que expone la información de cada usuario al atacante.

#### 5.10.11 Protegiendo credenciales de usuario

Para asegurar las credenciales del usuario, es importante gestionar el almacenamiento de contraseñas de una forma segura. Para ello, podemos utilizar el módulo bcrypt <https://github.com/kelektiv/node.bcrypt.js> y el método **hashSync**, pasando por parámetros el password que introduce el usuario y la semilla o salt que permite generar el hash del password.

La idea radica en poder guardar en una base de datos el hash y luego recuperarla para compararla con la password que introduce el usuario. Para manejar el almacenamiento de contraseñas de forma segura, podemos hacerlo generando primero el salt que se aplicaría sobre la password para obtener el hash:

```
// Generar hash a partir de la password
var salt = bcrypt.genSaltSync();
var passwordHash = bcrypt.hashSync(password, salt);

// Crear objeto usuario
var user = {
  userName: userName,
  firstName: firstName,
  lastName: lastName,
  password: passwordHash
};
```

Tenemos funciones para generar el *hash* a partir de la *password* en texto claro, y otra función que permite verificar si el *password* y el *hash* generado con dicha *password* coinciden. La llamada al método **bcrypt.hashSync** la podríamos utilizar en un proceso de registro de un usuario y la llamada al método **bcrypt.compareSync** la podríamos hacer en el proceso de login para verificar, de esta forma, que el usuario es correcto:

```
if (bcrypt.compareSync(password, user.password)) {
  callback(null, user);
} else {
  callback(invalidPasswordError, null);
}
```

Para comparar la contraseña cuando el usuario inicia sesión, la contraseña introducida por el usuario se convierte en hash y se compara con el hash que haya sido almacenado en la base de datos.

```
var bcrypt = require('bcrypt-nodejs');

var hashPassword = function(password, callback) {
  bcrypt.genSalt(10, function(err, salt) {
    if (err) return callback(err);

    bcrypt.hash(password, salt, null, function(err, hash) {
      if (err) return callback(err);

      callback(null, hash);
    });
  });
};
```

Figura 5.11 Uso de las funciones del módulo **bcrypt**.

### 5.10.12 Tiempo de espera de sesión y protección de cookies

La aplicación NodeGOAT no contiene ninguna validación para el tiempo de espera de sesión de usuario. La sesión permanece activa hasta que el usuario se desconecta explícitamente. Además, la aplicación no impide que se acceda a las cookies mediante código JavaScript, lo que hace que la aplicación resulte vulnerable a los ataques de cross-site scripting (XSS). Además, no se impide que las cookies se envíen en una conexión HTTP insegura.

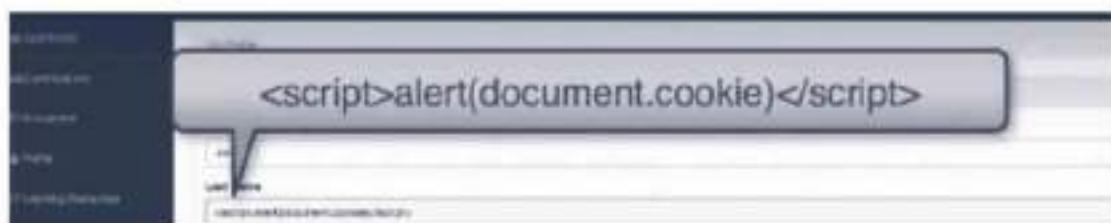


Figura 5.12 Acceso a las cookies del navegador.

Se deberán proteger las cookies de sesión de la aplicación para evitar que puedan ser accesibles y transmitidas a terceros sitios por medios no seguros. Para ello, se requiere activar el flag `Secure` y `HttpOnly` a los identificadores de sesión que manejan la aplicación. Si no se activa el flag `Secure` en la cookie, esta se enviará tanto en peticiones enviadas por medio seguro (HTTPS) como en peticiones no cifradas. Cuando no se activa el atributo `HttpOnly`, la cookie es susceptible de ser robada mediante código JavaScript. Se deberá controlar el tiempo de expiración de la cookie de sesión.

Para asegurar la aplicación en este aspecto, resulta importante habilitar la gestión de la sesión utilizando el Middleware de Express:

```
app.use(express.cookieParser());
```

Además, también se ha de establecer la cabecera `httpOnly`, para que las cookies solo puedan enviarse en conexiones seguras HTTPS configurando, igualmente, el flag `secure`:

```
app.use(express.session({
  secret: "s3Cur3",
  cookie: {
    httpOnly: true,
    secure: true
  }
}));
```

El atributo **httpOnly:true** indica al navegador que no permita el acceso a las cookies a través del objeto DOM `document.cookie` mediante JavaScript. Esta protección es obligatoria, con el fin de evitar el robo del identificador de sesión a través de ataques XSS.

El atributo de **secure:true** indica al navegador que solo envíe las cookies del usuario a través de una conexión HTTPS cifrada. Este mecanismo de protección de sesión resulta obligatorio, para evitar la obtención del identificador de sesión a través de un ataque MitM (*man in the middle*) y permite asegurar que un atacante no pueda capturar el identificador de sesión a partir del tráfico que genere el navegador al interactuar con la aplicación.

De esta forma, cuando el usuario trata de cerrar la sesión, destruye la sesión y la cookie de sesión:

```
request.session.destroy(function() {
  response.redirect("/");
});
```

Cuando utiliza cookies como identificador de sesión, también necesita un manejo seguro de cookies. Los atacantes intentarán robar cookies o, más específicamente, la información del token de sesión almacenada en esas cookies. Este ataque se denomina “secuestro de sesión”, porque se basa en el robo del token para acceder a la sesión autenticada de la víctima.

Primero se ha de configurar el servidor, para limitar su exposición y mitigar los vectores de ataque como el hombre en el medio (MITM) y las secuencias de comandos entre sitios (XSS).

Para evitar ataques de secuestro de sesión de MITM, debe utilizar HTTPS en todo el sitio y no solo para las páginas de inicio de sesión y registro. Si configura HTTPS, también configure la cookie como segura. Esto evitara que la cookie se envíe, a menos que sea parte de una solicitud HTTPS. Previene situaciones en las que el contenido inseguro del mismo dominio se manda a través de HTTP junto con el sessionID.

Podemos mitigar los vectores de ataque XSS impidiendo que JavaScript acceda al contenido de las cookies. Use la configuración httpOnly, que fue diseñada específicamente para ese propósito, de modo que solo las solicitudes del navegador tengan acceso a la cookie.

Además de httpOnly, también puede limitar la exposición de la cookie al establecer una restricción de dominio mínima; por ejemplo, si un sitio web con el dominio domain.com presenta una sección de inicio de sesión bien definida bajo el subdominio secure.domain.com, no hay razón para enviar una cookie por cada solicitud **".domain.com"**. A cuantos más lugares envíe su cookie, mayor será la superficie de ataque. De esta forma, podemos limitar la exposición de las cookies a su mínima superficie de exposición. En el siguiente ejemplo se analiza cómo evitar que alguien acceda a las cookies:

```
app.use(express.session({  
  cookie: {  
    domain: 'secure.domain.com'  
    secure: true,  
    path: '/',  
    httpOnly: true,  
    maxAge: null  
  }}));
```

En el ejemplo anterior, hemos definido el dominio para limitar la exposición de las cookies a un dominio en concreto y, mediante la propiedad secure, le indicamos que la cookie solo se envía si la conexión se hace mediante HTTPS.

Estos ajustes de configuración mitigan los ataques contra las sesiones activas, pero podría ser posible acceder a la información confidencial intercambiada desde la memoria caché del navegador, incluso después de que se haya cerrado la sesión. Esto resulta especialmente relevante cuando se consideran los puntos de acceso público.

Para mejorar la seguridad de la aplicación, las partes más sensibles del sitio web no deben almacenarse en caché, además del identificador de sesión. El encabezado Cache-control debe configurarse, al menos, en **Cache-Control: no-cache = "Set-Cookie, Set-Cookie2"**. Puede hacer esto usando Middleware:

```
// Establecer encabezado de control de caché para eliminar las cookies de caché
app.use(function (request, response, next) {
  response.header('Cache-Control', 'no-cache="Set-Cookie, Set-Cookie2"');
  next();
})
```

Lo anterior le indicará al navegador que no guarde esta información en la caché del navegador, para que no pueda acceder a ella más tarde.

También podríamos utilizar el módulo **express-session**, que se halla en el repositorio npm <https://www.npmjs.com/package/express-session> para indicar, de forma explícita, el tiempo de expiración de la sesión.

```
var express = require('express');
var session = require('express-session');

var app = express();

app.use(session({
  secret: 'our super secret session secret',
  cookie: { maxAge: 3600000 } // 2 hours in milliseconds
}));

app.listen(80);
```

Figura 5.13 Uso del módulo express-session para definir la expiración de la sesión.

También podríamos utilizar el módulo para configurar no enviar las cookies mediante HTTP, y solo se puedan enviar bajo un protocolo seguro como HTTPS. Para ello, utilizamos la propiedad **secure:true**, además de establecer la variable **trust proxy a 1**.

```
var express = require('express');
var session = require('express-session');

var app = express();

app.set('trust proxy', 1);

app.use(session({
  secret: 'our super secret session secret',
  cookie: {
    maxAge: 3600000,
    secure: true
  }
}));

app.listen(80);
```

Figura 5.14 Uso del módulo express-session para uso seguro de las cookies.

### 5.10.13 Secuestro de sesión (Session hijacking)

La aplicación NodeGOAT no regenera un nuevo identificador de sesión al iniciar la sesión del usuario, por lo que representa una vulnerabilidad de secuestro de sesión si un atacante puede, de alguna manera, robar la cookie con el identificador de sesión.

Al iniciar sesión, una práctica recomendada de seguridad con respecto a la administración de la sesión sería volver a generar la identificación de la sesión de modo que, si ya se creó una identificación para un usuario en un medio inseguro (es decir, un sitio web que no sea HTTPS), o si un atacante ha sido capaz de obtener el identificador de la cookie antes de que el usuario inicie sesión, el antiguo identificador de sesión ya no servirá, puesto que el usuario registrado con nuevos privilegios ahora dispone de un nuevo identificador de sesión.

Para asegurar la aplicación en este aspecto, se recomienda volver a generar un nuevo identificador de sesión después de iniciar sesión. La mejor práctica en este punto consiste en regenerarlos envolviendo el siguiente código como una función de callback para el método `request.session.regenerate()`:

```
request.session.regenerate(function() {
  request.session.userId = user._id;
  if (user.isAdmin) {
    return response.redirect("/benefits");
  } else {
    return response.redirect("/dashboard");
  }
})
```

#### 5.10.14 Módulo helmet

Cuando nuestro objetivo se basa en configurar correctamente las cabeceras HTTP relacionadas con la seguridad, hay algunas que se deben tener en cuenta. Estas cabeceras son:

- **Strict-Transport-Security** fuerza conexiones seguras (HTTP sobre SSL/TLS) con el servidor.
- **X-Frame-Options** proporciona protección clickjacking.
- **X-XSS-Protection** habilita el filtro de cross-site scripting (XSS), que tienen la mayoría de los navegadores.
- **X-Content-Type-Options**.
- **Content-Security-Policy** previene de una amplia gama de ataques, incluyendo secuencias de comandos en sitios cruzados y otras inyecciones de cross-site.

Helmet <https://helmetjs.github.io/> es un módulo que puede ayudar a proteger la aplicación de algunas vulnerabilidades web conocidas mediante el establecimiento de cabeceras HTTP de manera apropiada. Helmet es un Middleware diseñado para trabajar con Express y permite establecer las cabeceras HTTP de forma segura. Con solo dos líneas de configuración, protegemos la aplicación frente a varios ataques habilitando, de manera automática, las cabeceras:

```
var express = require('express');
var app = express();
var helmet = require('helmet');
app.use(helmet()); // Usa configuración por defecto
```

El código anterior, usando la configuración por defecto del módulo, realiza las siguientes comprobaciones y establece, de modo automático, las cabeceras:

- Elimina la cabecera **X-Powered-By**.
- Establece la cabecera **HSTS** para **HTTP Strict Transport Security**.
- Establece la cabecera **X-Download-Options** para IE8+, para prevenir la ejecución de descargas de forma automática.
- Establece la cabecera **X-Content-Type-Options**, para prevenir ataques por contenido MIME.
- Establece la cabecera **X-Frame-Options**, para prevenir ataques click-jacking.
- Establece la cabecera **X-XSS-Protection**, para prevenir ataques XSS.

En la configuración por defecto, helmet activa las cabeceras indicadas en la figura 5.15.

Module	Default?
contentSecurityPolicy for setting Content Security Policy	
dnsPrefetchControl controls browser DNS prefetching	✓
frameguard to prevent clickjacking	✓
hidePoweredBy to remove the X-Powered-By header	✓
hpkp for HTTP Public Key Pinning	
hsts for HTTP Strict Transport Security	✓
ieNoOpen sets X-Download-Options for IE8+	✓
noCache to disable client-side caching	
noSniff to keep clients from sniffing the MIME type	✓
referrerPolicy to hide the Referer header	

Figura 5.15 Cabeceras que activa por defecto el módulo helmet.

Helmet proporciona una colección de funcionalidades de Middleware que comprueban las cabeceras HTTP relacionadas con la seguridad:

- **csp** establece la cabecera Content-Security-Policy, para ayudar a prevenir ataques de cross-site scripting.
- **hidePoweredBy** elimina la cabecera X-Powered-By.
- **hpkp** previene ataques *man in the middle* con certificados falsificados.
- **hsts** establece la cabecera Strict-Transport-Security, que fuerza conexiones seguras con el servidor.
- **noCache** establece cabeceras Pragma Cache-Control para desactivar la caché del cliente.
- **frameguard** establece la cabecera X-Frame-, para proteger ataques del tipo de *clickjacking*.
- **xssFilter** establece la cabecera X-XSS-Protection, para habilitar el filtro de cross-site scripting (XSS).

Las cabeceras HTTP relacionadas con la seguridad se pueden añadir de forma automática usando el **módulo de helmet**:

```
// Deshabilita cabecera con versión de node que estamos utilizando
app.disable("x-powered-by");

// Evita que se abra la página en un iframe para protegerla del clickjacking
app.use(helmet.xframe());

// Evita que el navegador almacene en caché y almacene la página
app.use(helmet.noCache());

// Permitir la carga de recursos solo de dominios en lista blanca
app.use(helmet.csp());

// Permitir la comunicación solo en HTTPS
app.use(helmet.hsts());
```

Se recomienda utilizar la especificación 2.0 de CSP (Content Security Policy) y asignar a la cabecera X-Frame-Options el valor "DENY" para todas aquellas páginas que no se desea que figuren con marcos, o bien "SAMEORIGIN", para permitir únicamente marcos con origen único de la propia página que se está mostrando:

```
app.use(helmet.frameguard("SAMEORIGIN"));
app.use(helmet.frameguard("DENY"));
```

Para asegurarse de que el contenido de un recurso dado sea interpretado correctamente por el navegador, el servidor siempre debe enviar la cabecera **Content-Type** con el tipo de contenido y codificación correctos. El servidor también debería mandar la cabecera de seguridad **X-Content-Type-Options: nosniff**, con el fin de asegurarse de que el navegador no intente detectar un tipo de contenido diferente al que realmente se envía, lo cual podría producir una vulnerabilidad XSS. Además, el cliente debe enviar la cabecera **X-Frame-Options: deny**, para proteger contra los ataques del tipo *clickjacking* en los navegadores.

Podemos encontrar varios servicios que permiten validar la seguridad de las cabeceras, entre los que podemos destacar:

- <http://cyh.herokuapp.com/cyh>
- <https://securityheaders.io>

The screenshot shows a "Security Report Summary" page. At the top, there is a large letter "A" indicating a good grade. Below it, the URL is listed as "https://cyh.herokuapp.com/" and the IP Address as "192.33.241.112". The Report Time is "25 Apr 2019 12:27:49 UTC". Under Headers, several checkboxes are checked: "Strict-Transport-Security", "X-Frame-Options", "X-Content-Type-Options", and "X-XSS-Protection". A warning message below states: "Grade earned by A. Please see warnings below." The bottom section displays detailed header information for a request to "https://cyh.herokuapp.com/". It includes fields for Set-Cookie, X-Request-ID, X-HTTP-Transport-Security, X-Frame-Options, X-Content-Type-Options, and X-XSS-Protection. The X-Content-Type-Options field shows "nosniff" and the X-XSS-Protection field shows "1; mode=block".

Figura 5.16. Análisis de cabeceras de seguridad de un sitio web

### 5.10.15 Cross-site scripting (XSS)

La vulnerabilidad cross-site scripting (XSS) está relacionada con una pobre validación de datos de entrada a la aplicación que permite la inyección de código JavaScript, el cual, posteriormente, puede ejecutarse en la parte cliente de un

usuario y realizar acciones como obtener sesiones de usuario y otras acciones que pueden comprometer la confidencialidad e integridad de la aplicación.

Las vulnerabilidades XSS ocurren cuando una aplicación toma datos que no son de confianza y los envía a un navegador web sin la validación adecuada; es decir, en algún lugar del sitio web donde la entrada del usuario no se está validando correctamente, un atacante puede aprovechar el error para engañar a la aplicación, con el fin de que ejecute sus propios scripts.

XSS permite a los atacantes ejecutar scripts en el navegador de las víctimas, que pueden acceder a cualquier cookie, tokens de sesión u otra información confidencial almacenada por el navegador, o redirigir al usuario a sitios maliciosos. Existen dos tipos de vulnerabilidades XSS:

- **XSS reflejado:** El servidor responde en respuesta inmediata a una solicitud HTTP.
- **XSS almacenado:** Los datos maliciosos se almacenan en el servidor o en el navegador (por ejemplo, mediante el almacenamiento local/*localStorage* de HTML5) y, luego, se inyectan cuando se va a renderizar la página HTML. Dado que el XSS almacenado ya se encuentra en el servidor, cualquier usuario que visite la página correspondiente se verá afectado.

La vulnerabilidad XSS reflejada se relaciona con una pobre validación de datos de entrada a la aplicación que permite la inyección de código JavaScript, el cual, posteriormente, puede ejecutarse en la parte cliente de un usuario y realizar acciones como obtener sesiones de usuario y otras acciones que pueden comprometer la confidencialidad e integridad de la aplicación. Dado que el código inyectado no se almacena por la aplicación y requiere ser definido en el momento de su ejecución, la vulnerabilidad XSS se considera no persistente. La explotación de dicha vulnerabilidad suele llevarse a cabo a través de direcciones Url, las cuales incluyen el código que el usuario debe ejecutar y que será interpretado por el navegador como una petición válida.

Para prevenir este ataque, se pueden aplicar varias técnicas, entre las que podemos destacar:

- **Validación de entradas:** La validación de entradas del usuario tales como campos del formulario o la validación de parámetros en la Url son las primeras cosas que tendríamos que validar.
- **Codificación de salida:** Cuando un navegador está mostrando HTML y cualquier otro contenido asociado, como CSS y JavaScript, la codifica-

ción de salida sensible al contexto resulta crítica para mitigar el riesgo de XSS.

- **HTTPOnly cookie flag:** Este flag permite ayudar a minimizar el impacto de una vulnerabilidad XSS. Estableciendo esta cabecera, hacemos que no se pueda acceder a las cookies de sesión a través de JavaScript.
- **Implementar una política de seguridad de contenido (CSP):** CSP es un mecanismo del lado del navegador que permite crear listas blancas para los recursos del lado del cliente que usa la aplicación web; por ejemplo, JavaScript, CSS, imágenes, etc. CSP, a través de un encabezado HTTP, le indica al navegador que solo ejecute o genere recursos de esas fuentes; por ejemplo, el encabezado de CSP que se muestra a continuación permite el contenido solo desde el dominio propio mydomain.com y todos sus subdominios: **Content-Security-Policy: default-src 'self' \*.mydomain.com.**
- Aplicar codificación tanto en el lado del cliente como en el servidor para mitigar los ataques XSS basados en el DOM del navegador.
- Uso de cabeceras como **X-XSS-Protection:** no protege contra todo tipo de XSS, pero puede hacerlo contra XSS reflejado.
- Uso del módulo helmet para habilitar el filtro de XSS **app.use(helmet.xssFilter())**.
- Implementar la regla de protección ante XSS, consistente en codificar determinados caracteres de los datos de salida mediante entidades HTML, lo que evita que se produzcan ejecuciones de código malicioso.

#### 5.10.16 Referencias de objetos directos inseguros

Una referencia de objeto directo se produce cuando un desarrollador expone al exterior una referencia a un objeto de implementación interna de la aplicación, como un archivo, un directorio o una clave de base de datos. Sin una verificación de control de acceso, los atacantes pueden manipular tales referencias para acceder a datos no autorizados; por ejemplo, buscando rutas dentro de la aplicación que no hayan sido securizadas o carpetas del servidor a las que se puede acceder sin ningún tipo de control.

Las aplicaciones web normalmente utilizan el identificador de categorías o productos como parte de la Url para acceder a los detalles a través de la Url (/detalle/{id}). Un atacante podría manipular el valor del identificador para acceder a otras categorías o productos.

En este ejemplo vemos cómo el **parámetro userId es vulnerable** al recogerlo directamente de la Url:

```
var userId = request.params.userId;
allocationsDAO.getByUserId(userId, function(error, allocations) {
if (error) return next(error);
return response.render("allocations", allocations);
})
```

Una alternativa más segura consiste en recuperar variables que afecten a los datos del usuario que ha iniciado sesión utilizando **request.session.userId**, en lugar de recogerlo a partir de la Url con **req.params.userId**.

Entre las mejores prácticas para aplicar en este punto, podemos destacar:

- **Verificación de acceso:** Cada uso de una referencia de objeto directo de una fuente no confiable debe incluir una verificación de control de acceso, para garantizar que el usuario esté autorizado para el objeto solicitado.
- **Utilizar referencias de objetos indirectos por usuario o por sesión:** En lugar de exponer claves de base de datos reales como parte de los enlaces de acceso, podríamos usar referencias indirectas temporales por usuario; por ejemplo, en lugar de usar la clave de la base de datos del recurso para mostrar una lista desplegable al usuario, podría usar números aleatorios únicos, para indicar qué valor seleccionó el usuario. La aplicación, posteriormente, asignaría la referencia a la clave de base de datos en el servidor.
- **Pruebas y análisis de código:** Los evaluadores pueden manipular fácilmente los valores de los parámetros para detectar tales errores. Además, el análisis de código puede mostrar de forma rápida si la autorización se verifica correctamente.

#### 5.10.17 Mala configuración de seguridad

Una mala configuración de seguridad puede ocurrir en cualquier nivel de desarrollo de la aplicación, ya sea una mala configuración del servidor web, el servidor de aplicaciones o la base de datos.

Dicha vulnerabilidad permite que un atacante acceda a cuentas predeterminadas, páginas no utilizadas, vulnerabilidades no parcheadas, archivos y

directorios no protegidos, etc., con el fin de obtener acceso no autorizado del sistema. Tal vulnerabilidad abarca una amplia categoría de ataques, pero aquí hay algunas formas en que un atacante puede explotarla:

- Si el servidor de aplicaciones está configurado para ejecutarse como root, un atacante puede ejecutar scripts maliciosos o iniciar nuevos procesos en el servidor.
- Leer, escribir, borrar archivos en el sistema de archivos.
- Crear y ejecutar archivos binarios.
- Si el servidor está mal configurado de forma que se filtren detalles de la implementación interna a través de nombres de cookies o encabezados de respuesta HTTP, entonces el atacante puede usar esta información para encontrar otras vulnerabilidades.
- Si el tamaño del cuerpo de una petición no está limitado en las cabeceras, un atacante puede enviar una petición con una gran carga de datos de entrada, lo que podría hacer que el servidor se quedase sin memoria y provocar una denegación de servicio.

En esta lista, recogemos algunas **buenas prácticas y medidas de configuración específicas** para las aplicaciones basadas en NodeJS:

- Usar siempre la última versión estable de NodeJS y Express. Las últimas vulnerabilidades y actualizaciones de seguridad se pueden encontrar en:
  - <https://nodejs.org/en/feed/vulnerability.xml>
  - <http://expressjs.com/en/advanced/security-updates.html>
- Se recomienda no ejecutar la aplicación con privilegios de root. Puede parecer necesario ejecutar como usuario root para acceder a determinados puertos; sin embargo, esto puede lograrse, ya sea iniciando el servidor como root y, luego, bajar privilegios al usuario después de establecer la escucha en el puerto o usando un proxy.
- Revisar los valores predeterminados en las cabeceras de respuesta HTTP, para evitar la divulgación interna de la implementación.
- Establecer encabezados HTTP de seguridad específicos. Esto se puede lograr gracias a librerías como **helmet**:  
<https://www.npmjs.com/package/helmet>

- Bloquear las versiones de todos los paquetes npm utilizados, por ejemplo, utilizando shrinkwrap, para tener un control total sobre cuándo instalar una nueva versión del paquete.

### 5.10.18 Deshabilitar *fingerprinting*

En la configuración por defecto de Express, por cada petición que se realice, se envía, además, una cabecera HTTP que identifica que el sitio se ha desarrollado con Express; por ejemplo, en la búsqueda con Shodan, podemos ver información sobre los servidores que están usando Express como librería específica de NodeJS.



```

▼ Softphone ↗
159.88.198.200
Digital Ocean
Added on 2010-04-24 10:40:21 GMT
India, Bangalore
status
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Wed, 24 Apr 2019 10:40:20 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 584
Connection: keep-alive
X-Powered-By: Express
Accept-Ranges: bytes
Cache-Control: public, max-age=8
Last-Modified: Wed, 27 Mar 2019 10:15:41 GMT
ET...

```

Figura 5.17 Cabecera X-Powered-By:Express en un servidor nginx.

El encabezado HTTP predeterminado x-powered-by puede revelar los detalles de la implementación a un atacante. Esta **cabecera se puede deshabilitar** de una forma sencilla con la siguiente linea:

```
app.disable("x-powered-by");
```

Otra opción consiste en añadir cabeceras específicas que permitan falsear la información devuelta con otra versión del servidor:

```
app.use(helmet.hidePoweredBy({ setTo: 'PHP 4.2.0'}))

var options = {
  setHeaders: function(res, path, stat) {
    res.set('Server','cloudflare-nginx');
  }
}

app.use(express.static('public', options));
```

### **5.10.19 Exposición de datos sensibles**

Esta vulnerabilidad le permite a un atacante acceder a datos confidenciales, como credenciales de autenticación. La pérdida de dichos datos puede causar un impacto grave y dañar la reputación de la organización. Los datos confidenciales necesitan una protección adicional a nivel de cifrado, así como precauciones especiales cuando se intercambian con el navegador.

Si un sitio no usa SSL/TLS para todas las páginas autenticadas, un atacante podría monitorizar el tráfico de la red y obtener las cookies de sesión de navegación del usuario. El atacante podría, posteriormente, reproducir esta cookie y obtener la sesión del usuario, con lo que accedería a sus datos privados. Por ello, se recomienda acometer las siguientes prácticas:

- Utilizar el protocolo de red seguro HTTPS.
- Cifrar todos los datos confidenciales estáticos y dinámicos.
- Desactivar el autocompletado en los campos de los formularios que recopilan datos confidenciales y deshabilitar el almacenamiento en caché de las páginas que contienen datos confidenciales.

Las aplicaciones que manejen información confidencial, como pueden ser datos personales, datos bancarios, nombres de usuario o contraseñas, deben utilizar protocolos que cifren la comunicación entre el cliente y el servidor. De esta forma, en caso de que la comunicación sea interceptada, el tráfico nunca se obtendrá en texto claro.

Si se envían datos confidenciales a través de una aplicación web y el canal por el que se transmiten no se encuentra debidamente cifrado, un usuario que se halle en la misma red que el cliente o el servidor podría interceptar la comunicación y leer la información enviada y recibida. Resulta especialmente importante, como mínimo, forzar el uso de SSL en el formulario de autenticación de la aplicación.

### **5.10.20 Configurando SSL/TLS**

La comunicación segura y resistente a la manipulación entre un cliente y un servidor se produce a través de SSL (Secure Sockets Layer) y el protocolo TLS (Transport Layer Security). La unión de estos protocolos es lo que se conoce hoy día como **HTTPS**. Una conexión SSL/TLS requiere un protocolo de enlace entre el cliente y el servidor, durante el cual el cliente (normalmente un navegador)

le permite al servidor saber qué tipo de funciones de seguridad admite. El servidor elige una función y la envía a través de un certificado SSL, que incluye su clave pública. El cliente confirma el certificado y genera un número aleatorio usando la clave del servidor, y lo envía de vuelta al servidor.

El servidor, posteriormente, usa su clave privada para descifrar el número que, a su vez, se emplea para habilitar la comunicación segura. Con el fin de que todo esto funcione, deberá generar tanto la clave pública como la privada, así como el certificado. Para fines de desarrollo, puede hacer uso de un certificado autofirmado; sin embargo, para una aplicación en producción, el certificado debería estar firmado, al menos, por una **autoridad certificadora de confianza (CA)**.

TLS utiliza lo que se conoce como “criptografía de clave pública”, donde todos los elementos de la comunicación disponen de una clave pública que comparten con todos y una clave privada que no comparten. El emisor cifra el mensaje con su clave privada y su clave pública (disponible públicamente para cualquier persona). El receptor descifra con su clave privada y la clave pública del emisor.

Para configurar un servidor HTTPS en NodeJS, solo necesitamos certificados SSL válidos generados por una autoridad de certificación (CA). La herramienta OpenSSL constituye el estándar utilizado para generar los certificados con las claves pública y privada. **El siguiente comando genera la clave privada de 1024 bits y encriptada con Triple-DES:**

```
openssl genrsa -des3 -out private.key 1024
```

Lo siguiente sería realizar la solicitud de firma de certificado (CSR):

```
openssl req -new -key private.key -out cert.csr
```

Por último, generamos el certificado autofirmado. El siguiente comando crea un certificado que funcionaría durante 365 días:

```
openssl x509 -req -days 365 -in cert.csr -signkey private.key -out certificate.crt
```

Si necesita solicitar un certificado de una CA de confianza, puede empezar a trabajar con Let's Encrypt, una CA gratuita y de código abierto. Puede consultar el servicio en <https://letsencrypt.org>.

Una vez que tenga un certificado, puede usar el módulo HTTPS incorporado de NodeJS con Express. Es muy similar al módulo HTTP, con la diferencia de que tendrá que proporcionar su clave privada y certificado generados con openssl.

A nivel de cifrado, se puede establecer un servidor HTTPS seguro utilizando el **módulo https**. Esto necesitaría una clave privada y un certificado. A continuación mostramos ejemplos de código fuente:

```
// Cargar claves para establecer una conexión segura HTTPS
var fs = require("fs");
var https = require("https");
var path = require("path");

//Define un objeto que contiene su clave privada y su certificado
var httpsOptions = {
  key: fs.readFileSync(path.resolve(__dirname, "./certificates/private.key")),
  cert: fs.readFileSync(path.resolve(__dirname, "./certificates/certificate.crt"))
};
```

Con el método **createServer**, podemos arrancar un servidor seguro HTTPS:

```
https.createServer(httpsOptions, app).listen(config.port, function() {
  console.log("Express https server listening on port " + config.port);
});
```

El ejemplo anterior se puede simplificar utilizando el framework **Express**:

```
var fs = require('fs');
var https = require('https');
var express = require('express');
var app = express();
app.get('/', function (request, response, next) {
  response.send('hello world');
});

//Define un objeto que contiene su clave privada y su certificado

var options = {
  key: fs.readFileSync(path.resolve(__dirname, "./certificates/private.key")),
  cert: fs.readFileSync(path.resolve(__dirname, "./certificates/certificate.crt"))
};
https.createServer(options, app).listen(443);
```

Se debe tener en cuenta que NodeJS no redirige automáticamente de HTTP a HTTPS. De esta forma, necesitamos configurar nuestro controlador de peticiones HTTP para que realice la redirección:

```
var httpApp = express();
httpApp.get('*', function (request, response){
    response.redirect('https://' + request.headers.host + request.url);
});
httpApp.listen(443);
```

Resulta importante señalar que, para manejar las conexiones HTTPS, deberíamos usar servidores específicos, como **Apache** o **Nginx**. Además, el software de servidor dedicado es mucho mejor en la creación y el manejo de conexiones SSL, con soporte para funciones HTTP avanzadas como HTTP2.

#### 5.10.21 Forzar peticiones HTTPS

Dentro del repositorio de paquetes npm, podemos encontrar módulos que permiten forzar el uso de peticiones HTTPS, como el de **express-enforces-ssl**. Básicamente, si la petición funciona sobre HTTPS, continúa funcionando de la misma forma. Si no lo es, redirige a la versión HTTPS. Para usar este módulo, habría que realizar un par de configuraciones:

1. Habilitar la configuración de “proxy de confianza”. La mayoría de las veces, cuando implementa sus aplicaciones, la aplicación cliente no se conecta directamente al servidor, sino que pasa por una serie de proxies; por ejemplo, si está usando Heroku <https://www.heroku.com> como plataforma de despliegue, los servidores Heroku actúan a modo de proxy y se ubican entre su servidor y el cliente que realiza la petición. Para informar a Express sobre esto, debe habilitar la configuración de “proxy de confianza”.
2. Cargar el módulo y llamar al Middleware.

```
var enforceSSL = require("express-enforces-ssl");
app.enable("trust proxy");
app.use(enforceSSL());
```

En el repositorio de GitHub <https://github.com/hengkiardo/express-enforces-ssl> encontramos el módulo que realiza esta función.

```
$ npm install express-enforces-ssl --save
```

Afterwards, require the module and use the `HTTPS()` method:

```
var express = require('express');
var http = require('http');
var express_enforces_ssl = require('express-enforces-ssl');

var app = express();

app.enable('trust proxy');

app.use(express_enforces_ssl());

/*
  Routes Here
*/

http.createServer(app).listen(app.get('port'), function() {
  console.log('Express server listening on port ' + app.get('port'));
});
```

Figura 5.18 Uso del módulo `express-enforces-ssl`.

Para mejorar la experiencia de navegación al usar HTTPS, los navegadores admiten una cabecera llamada **HTTP Strict Transport Security (HSTS)**. Se trata de una cabecera que les dice a los navegadores que permanezcan en HTTPS por un periodo de tiempo. Si desea mantener a sus usuarios en HTTPS durante un año (aproximadamente 31 536 000 segundos), lo puede hacer estableciendo la siguiente cabecera:

**Strict-Transport-Security: max-age=31536000**

HTTP Strict Transport Security (HSTS) es una cabecera que le pide al agente de usuario que se esté utilizando en este momento que cambie todos los enlaces HTTP inseguros a HTTPS y rechace cualquier conexión TLS/SSL en la que el usuario no confie. Si no es práctico obligar a todos los usuarios a usar HSTS, los desarrolladores si deberían, al menos, ofrecer a los usuarios la opción de habilitarlo si desean utilizarlo.

Para configurar esta cabecera, podemos usar el módulo Helmet:  
<https://github.com/helmetjs/helmet>, un módulo para configurar las cabeceras

de seguridad HTTP útiles en sus aplicaciones basadas en Express. Con el módulo helmet, puede utilizar el Middleware de la siguiente forma:

```
var helmet = require("helmet");
var ms = require("ms");
app.use(helmet.hsts({
  maxAge: ms("1 year"),
  includeSubdomains: true
}));
```

### 5.10.22 Falta de control de acceso

La mayoría de las aplicaciones web verifican los permisos de acceso antes de hacer visible esa funcionalidad en la interfaz de usuario. Sin embargo, las aplicaciones deben realizar las mismas comprobaciones de control de acceso en el servidor cuando se accede a cada función.

Si no se verifican las peticiones junto con los permisos de acceso en el servidor, los atacantes pueden falsificar las peticiones para acceder a la funcionalidad sobre la cual no estén autorizados. En una aplicación que fuera vulnerable, el acceso a una ruta específica no validaría que el usuario que intenta acceder dispone de la autorización adecuada:

```
app.get("/detalle", usuarioAutenticado, detalleHandler.verDetalle);
app.post("/detalle", usuarioAutenticado, detalleHandler.actualizarDetalle);
```

Esto se puede solucionar agregando una función Middleware que permita comprobar si el usuario cuenta con permisos de administrador:

```
app.get("/detalle", usuarioAutenticado, usuarioAdmin, detalleHandler.verDetalle);
app.post("/detalle", usuarioAutenticado, usuarioAdmin, detalleHandler.actualizarDetalle);
```

Para implementar **usuarioAdmin**, podríamos comprobar si el indicador `isAdmin` está establecido para el usuario que inició sesión en la base de datos:

```
this.usuarioAdminMiddleware = function(request, response, next) {
    if (request.session.userId) {
        userDao.getUserById(request.session.userId, function(err, user) {
            if(user && user.isAdmin) {
                next();
            } else {
                return response.redirect("/login");
            }
        });
    } else {
        console.log("redirecting to login");
        return response.redirect("/login");
    }
};

var sessionHandler = require("./session");
// Middleware para comprobar si el usuario tiene permisos de administrador.
var usuarioAdmin = sessionHandler.usuarioAdminMiddleware;
```

### 5.10.23 Redirecciones no validadas

Las aplicaciones web, con frecuencia, redirigen y envían a los usuarios a otras páginas y sitios web y utilizan datos que no son de confianza para determinar las páginas de destino. Sin la validación adecuada, los atacantes pueden redirigir a las víctimas a sitios de phishing o malware, o usar reenvíos para acceder a páginas no autorizadas.

Un atacante puede usar enlaces redirigidos no validados como medio para redirigir al usuario a contenidos maliciosos y trucos, para que las víctimas hagan clic en ellos. El atacante puede explotarlo para pasar por alto los controles de seguridad y hacer que se considere confiable.

Por ejemplo, el enlace “Consultar detalle” (`/obtenerDetalle?url=`) en la aplicación puede redirigir a otro sitio web sin validar la Url:

```
// Manejar redirección para enlace de obtener detalle
app.get("/obtenerDetalle", function (request, response, next) {
    return response.redirect(request.query.url);
});
```

Un atacante podría cambiar el parámetro Url para apuntar a un sitio web malicioso. Resulta más probable que las víctimas hagan clic en él, ya que la parte

inicial del enlace apunta a un sitio que parece confiable. El uso seguro de redirecciones y reenvíos se puede implementar de varias maneras:

- Evitar, siempre que se pueda, usar redirecciones y reenvíos.
- No incluir los parámetros del usuario en la generación de la url.
- Si no hay más remedio que incluir los parámetros de usuario, es importante asegurarse de que el valor proporcionado sea válido y esté autorizado para el usuario que está haciendo la petición.

#### 5.10.24 Denegación de servicio mediante expresiones regulares

Uno de los principales problemas de NodeJS estriba en que es vulnerable a ataques DoS. Al ejecutar un solo subprocesso, puede usar tanta CPU como sea necesario para completar una tarea a la vez. Esa es la razón por la que la mayoría de las tareas de la CPU son generalmente cortas, para dar a otras tareas la oportunidad de ejecutarse. Cuando el subprocesso necesita hacer un uso intensivo de la CPU para realizar un cálculo, podría llegar a agotar todos los recursos de la CPU, y acabar bloqueando todos los demás eventos en espera.

La denegación de servicio mediante expresión regular (ReDoS) constituye un ataque de denegación de servicio que explota el hecho de que la mayoría de las implementaciones de expresiones regulares pueden alcanzar situaciones que hagan que se colapse el servidor ante una entrada anormal. Un atacante puede hacer que la aplicación entre en un estado donde se queda colgada por un tiempo muy largo.

Cuando la entrada de datos se ejecuta en el contexto de la comprobación de una expresión regular ante una cadena de entrada, se pueden explotar patrones vulnerables para ejecutar cálculos largos para que coincidan con una cadena dada. Para NodeJS, esto resulta crítico, debido a la arquitectura de bucle de eventos de un solo subprocesso, lo que significa que el proceso principal de NodeJS no puede atender otras solicitudes.

Para prevenir este tipo de ataques, se recomienda implementar las siguientes prácticas:

- Usar el paquete **express-validator** <https://express-validator.github.io> de NodeJS para validar el formato de datos esperado, en lugar de escribir sus propias expresiones regulares.
- Como último recurso para escribir sus propios patrones de expresiones regulares, puede utilizar el paquete **safe-regex** <https://www.npmjs.com/package/safe-regex> de Node.js, que permite detectar si una

expresión regular se muestra propensa a producir un ataque de denegación de servicio.

Para probar el módulo safe-regex, podríamos declarar el siguiente código en un fichero js y, posteriormente, ejecutarlo con node:

```
var safe = require('safe-regex');
var regex = process.argv.slice(2).join(' ');
console.log(safe(regex));
```

Al ejecutarlo, podríamos pasarle por parámetro la expresión regular que queremos validar. Si el resultado es true, significa que la expresión resulta segura; si devuelve false, significa que la expresión no resulta segura desde el punto de vista de poder llegar a realizar una denegación de servicio. Las siguientes expresiones son no seguras:

```
$ node safe.js '{x+y}*'  
false  
$ node safe.js '{a+}{10}'  
False
```

#### 5.10.25 Validar datos de entrada con validator

Podemos garantizar que los datos de entrada sean válidos con el módulo validator, disponible en el repositorio de npm <https://www.npmjs.com/package/validator>:

```
npm install validator
```

Por ejemplo, podemos comprobar que los datos de entrada tengan un formato concreto, como verificar que el texto introducido sea un correo electrónico sin necesidad de utilizar expresiones regulares:

```
var validator = require('validator');
var email = 'user@domain.com';

try {
  validator.isEmail(email);
} catch (err) {
  console.log(err.message);
}
```

En el repositorio de GitHub <https://github.com/validatorjs/validator.js>, encontramos todos los validadores disponibles; por ejemplo, tenemos métodos para validar código postal, una url o si una cadena aparece en mayúsculas.

	check if the string is a postal code.
isPostalCode(str, locale)	(locale is one of: ['AD', 'AT', 'AU', 'BE', 'BG', 'CA', 'CH', 'CZ', 'DE', 'DK', 'FI', 'EE', 'ES', 'PT', 'FR', 'GB', 'GR', 'HU', 'ID', 'IL', 'IN', 'IS', 'IT', 'JP', 'KR', 'LT', 'LV', 'MT', 'NL', 'NO', 'NZ', 'PL', 'PR', 'PT', 'RO', 'RU', 'SA', 'SE', 'SI', 'TN', 'TR', 'UA', 'ES', 'ZA', 'ZH'] OR 'any'. If 'any' is used, function will check if any of the locales match. Locale list is <code>validator.isPostalCodeLocales</code> ).
isSurrogatePair(str)	check if the string contains any surrogate pairs chars.
	check if the string is an URL.
isURL(str [, options])	options is an object which defaults to { protocols: ['http', 'https', 'ftp'], require_tld: true, require_protocol: false, require_host: true, require_valid_protocol: true, allow_underscores: false, host_whitelist: false, host_blacklist: false, allow_trailing_dot: false, allow_protocol_relative_urls: false, disallow_port: false }.
isUUID(str [, version])	check if the string is a UUID (version 3, 4 or 5).
isUppercase(str)	check if the string is uppercase.
isVariableWidth(str)	check if the string contains a mixture of full and half-width chars.

Figura 5.19 Validadores disponibles en el módulo validators.

#### 5.10.26 Validar datos de entrada con express-validator

También podríamos utilizar el módulo **express-validator**, disponible en el repositorio npm <https://www.npmjs.com/package/express-validator>. Puede acceder a los métodos de verificación y otros métodos proporcionados directamente en el objeto request:

```
var expressValidator = require('express-validator');
app.use(expressValidator);

app.get('/somepage', function (request, response) {
  request.check('zip', 'Please enter zip code').isInt(6);
  request.sanitize('newdata').xss();
});
```

Con el módulo **express-validator**, también podemos escapar todas las entradas del usuario en las peticiones que se realicen.

```
var express      = require('express');
var bodyParser = require('body-parser');
var validator   = require('express-validator');

var app = express();

app.use(bodyParser.urlencoded());

app.use(validator());
app.use(function(req, res, next) {
    for (var item in req.body) {
        req.sanitize(item).escape();
    }
    next();
});

app.listen(80);
```

Figura 5.20 Uso del módulo express-validator.

También podríamos utilizar el módulo para validar si los campos de un formulario son requeridos a través del objeto request con los métodos `request.assert()` y `request.validationErrors()`:

```
var express = require('express');
var bodyParser = require('body-parser');
var validator = require('express-validator');
var app = express();

// middleware
app.use(bodyParser.urlencoded({ extended: false }));
app.use(validator());
app.post('/login', function(request, response){
    request.assert('password', 'Password is required').notEmpty();
    request.assert('email', 'A valid email is required').notEmpty().isEmail();
    var errors = request.validationErrors();
    if (errors)
        response.render('index', {errors: errors});
    else
        response.render('login', {email: request.email});
});
```

Password is required

A valid email is required

user	Password	Login
------	----------	-------

Figura 5.21 Validaciones de password y e-mail con express-validator.

#### 5.10.27 Configuración de cabeceras HTTP

Como propietario de un sitio web o desarrollador web, puede controlar qué cabeceras HTTP puede enviar su servidor. El propósito de esta sección reside en dar a conocer las diferentes cabeceras HTTP de respuesta que un servidor web puede incluir en una solicitud y el impacto que tienen en la seguridad para el navegador web. Los desarrolladores web pueden implementar las siguientes cabeceras, para que la experiencia del usuario sea más segura:

- **X-Content-Options.** La cabecera X-Content-Options permite indicar al navegador web que se debe seguir el MIME (indicado mediante el encabezado Content-Type) para el contenido solicitado. La solicitud que incluye la cabecera X-Content-Options le dice al navegador web que no debería hacer una búsqueda MIME en el documento. En este caso, estamos previniendo al usuario contra un ataque XSS.
- **X-XSS-Protection.** Esta cabecera le dice al navegador web que la protección XSS incorporada debe estar habilitada. En la mayoría de los navegadores web modernos, el filtro XSS se encuentra habilitado de forma predeterminada. El valor recomendado de esta cabecera es 1; mode = block, lo que significa que el filtro XSS está habilitado y se bloquearán aquellas solicitudes que puedan originar un ataque XSS.



Figura 5.22 Protección XSS del navegador.

Si el servidor establece el valor de esta cabecera en 0, esto deshabilitará la protección contra XSS. Como resultado, vemos que se representó el contenido de JavaScript reflejado.

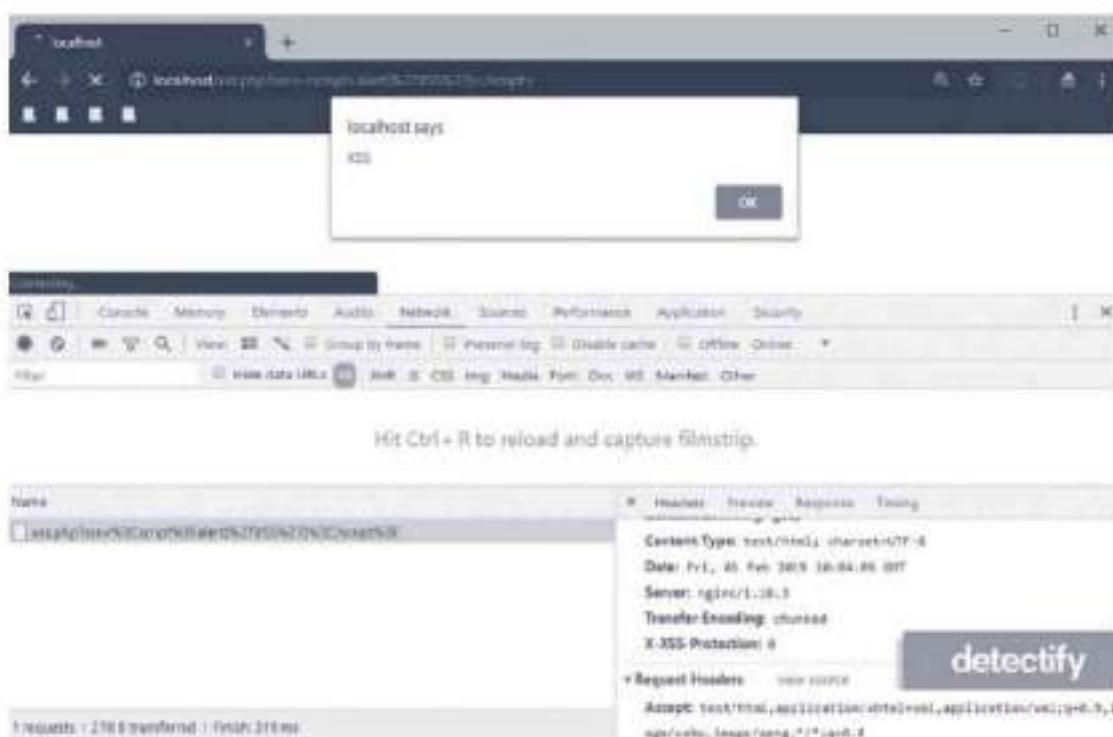


Figura 5.23 Vulnerabilidad XSS.

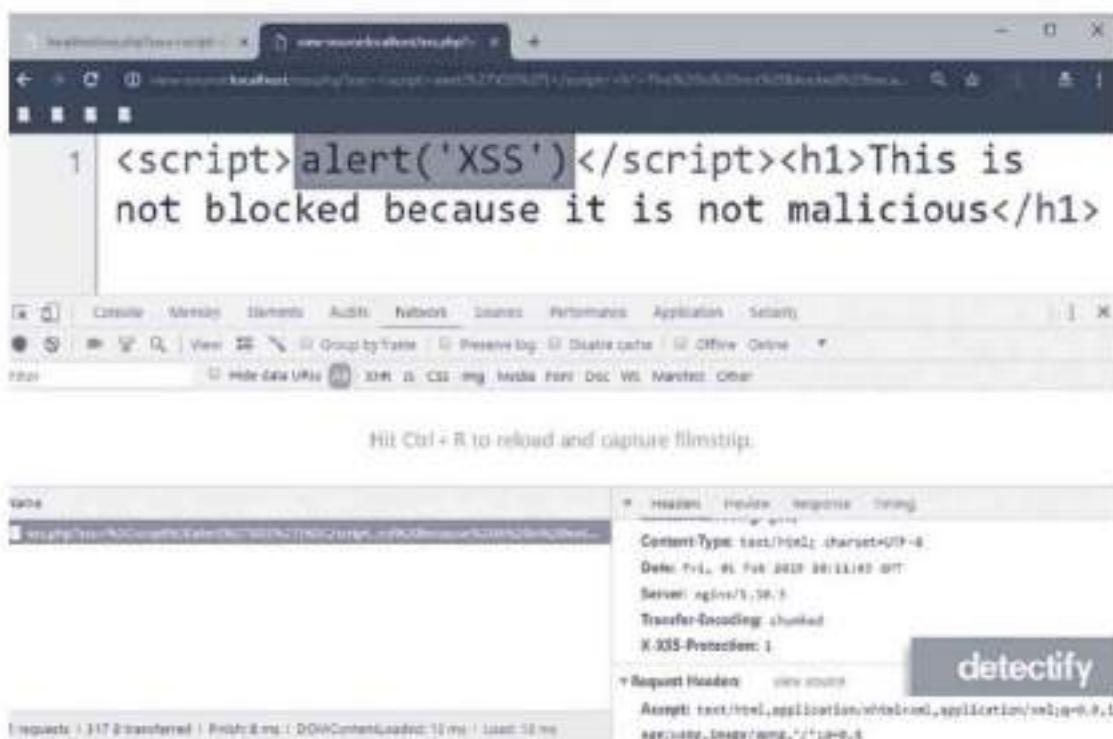


Figura 5.24 Bloqueo de contenido malicioso.

- **Set-Cookie.** La cabecera Set-Cookie establece una cookie con algunos valores específicos. La cabecera posee algunos atributos que se deben tener en cuenta si la aplicación web utiliza cookies HTTP para la autenticación.
  - **HttpOnly:** El atributo HttpOnly le dice al navegador web que solo se debe poder acceder a la cookie a través del encabezado de solicitud HTTP; esto significa que no se puede acceder a la cookie a través de JavaScript. Este atributo es muy importante de incluir porque, de lo contrario, se puede explotar una vulnerabilidad XSS para leer la cookie y enviarla al atacante, quien puede hacerse cargo de la sesión por completo.
  - **Secure:** Este atributo le dice al navegador web que la cookie solo debe enviarse a través de una conexión segura (normalmente HTTPS). Esto podría proteger a un usuario si un atacante está escuchando en la red para robar cookies.
  - **SameSite:** El atributo SameSite proporciona una protección extra contra los ataques CSRF. Si una cookie utiliza el atributo SameSite, el navegador web se asegurará de que la solicitud realizada con la cookie provenga del origen donde estaba la cookie.
  - **\_Host-and \_Secure:** Se usan como prefijos para nombrar las cookies. La razón por la que es un prefijo y no un atributo estriba en que, si modifica el nombre de la cookie, tendrá que cambiar el servidor para aceptar el nuevo nombre de la cookie.

#### 5.10.28 Política de seguridad de contenido (CSP)

Content-Security-Policy es una cabecera que permite definir cómo diferentes recursos pueden manejarse por el navegador web. Si se configura correctamente, puede limitar la superficie de ataque en gran medida. Requiere una mayor comprensión de la aplicación web para utilizar una política que sea estricta y evitar bloquear los recursos que no debería. Google dispone de una herramienta llamada CSP Evaluator (<https://csp-evaluator.withgoogle.com>), la cual permite evaluar un CSP para determinar si se puede considerar seguro o no.

CSP Evaluator facilita a los desarrolladores y expertos en seguridad para comprobar si una política de seguridad de contenido (CSP) es segura y permite mitigar los ataques de scripts entre sitios. Las comprobaciones que realiza esta

herramienta están destinadas a ayudar a los desarrolladores a fortalecer las políticas y mejorar la seguridad de sus aplicaciones.

#### 5.10.29 Cross-site request forgery (CSRF)

La falsificación de solicitudes entre sitios constituye una de las vulnerabilidades en la lista de los 10 principales de OWASP:

<https://www.cvedetails.com/vulnerability-list/year-2019/opsrf-1/csrf.html>

Se trata de un ataque utilizado para realizar solicitudes en nombre del usuario, donde el atacante aprovecha el hecho de que el usuario ya aparece autenticado.

Un ataque CSRF obliga al navegador de una víctima que ha iniciado sesión a enviar una solicitud HTTP falsificada, incluida la cookie de sesión de la víctima y cualquier otra información de autenticación incluida automáticamente, a una aplicación web vulnerable. Esto permite al atacante forzar al navegador de la víctima a generar solicitudes que la aplicación vulnerable procesa como solicitudes legítimas de la víctima.

Como los navegadores envían automáticamente credenciales como cookies de sesión con solicitudes HTTP al servidor desde donde se reciben las cookies, los atacantes pueden crear páginas web maliciosas que generan solicitudes falsificadas que no se pueden distinguir de las legítimas.

Para protegernos en este caso, lo que más se utiliza es generar un *token* cada vez que enviamos información sensible al servidor. Cada petición se valida comparando el valor presentado con el valor del *token* esperado. Si los valores coinciden, la solicitud resulta válida.

Para mitigar este tipo de ataques en la plataforma de NodeJS, podemos encontrar el módulo `csurf` <https://www.npmjs.com/package/csrf>. De forma predeterminada, este Middleware genera un token llamado “`_csrf`”, que debe añadirse a las peticiones que realizan envío de datos al servidor (PUT, POST o DELETE); normalmente, se envía dentro de un campo oculto del formulario de envío.

Cuando se envía el formulario, el Middleware comprueba la existencia del token y lo valida, para hacer coincidir el token generado para el par de solicitud de respuesta. Si los tokens no coinciden, rechaza la solicitud. Esta simple, pero efectiva, medida hace que sea bastante difícil para un atacante explotar una vulnerabilidad de este tipo.

Una vez que hayamos añadido el módulo y configurado el Middleware, podemos generar un token que pasaremos a la vista:

```
var csrf = require("csurf");
app.use(csrf());
app.get("/", function(request, response) {
  response.render("myview", {
    csrfToken: request.csrfToken()
  });
});
```

En la vista podemos utilizar la variable `csrfToken` que ha generado nuestro controlador en un campo oculto, con el nombre `_csrf`:

```
<form method="post" action="/submit">
  <input name="_csrf" value="<% csrfToken %>" type="hidden">
</form>
```

Una vez que haya añadido el token CSRF a sus formularios, el Middleware `csurf` se encargará del resto. En el siguiente ejemplo se configura el Middleware de CSRF con Express, en lugar de utilizar el módulo `csurf`, una vez se ha inicializado la sesión del usuario. Posteriormente crea un Middleware personalizado para generar un token nuevo usando `request.csrfToken()` y lo expone a la vista mediante `response.locals`:

```
app.use(express.csrf());
app.use(function(request, response, next) {
  response.locals.csrfToken = request.csrfToken();
  next();
});
```

Una de las maneras de implementar la validación consiste en el uso de Middleware personalizado, para pasar el token CSRF a todas las plantillas utilizando `response.locals`. Este Middleware personalizado se tiene que poner antes de la definición de las rutas.

En la parte de la vista, este *token* se puede incluir en un **campo de formulario oculto** de forma que, cada vez que se carga el formulario, se genera un nuevo token:

```
<input type="hidden" name="_csrf" value="{{ csrfToken }}>
```

Cuando una aplicación web realiza una petición, el formulario debe incluir un parámetro de entrada oculto “`_csrf`”. Un token CSRF debe ser único por sesión de usuario, de forma que el servidor rechaza la acción solicitada si el token CSRF se cambia y falla la validación.



```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <form method="post" action="/login">
      <input type="hidden" name="_csrf" value="mHGrhVvu:h2p047q2P%3egIuv911X0%1NN5g" />
      <input type="text" name="email" placeholder="user@domain.com">
      <input type="password" name="password" placeholder="Password">
      <button type="submit">Login</button>
    </form>
  </p></p>
</body>
</html>
```

Figura 5.25 Uso del campo oculto para guardar el token autogenerado.

En general, los desarrolladores solo necesitan generar este *token* una vez para la sesión actual. Después de la generación inicial de este *token*, el valor se almacena en la sesión y se usa para cada solicitud hasta que la sesión caduque. Cuando el usuario emite una solicitud, el componente del lado del servidor debe verificar la existencia y la validez del token en la solicitud en comparación con el token encontrado en la sesión del usuario. Si no se encontró el *token* dentro de la petición, o si el valor proporcionado no coincide con el valor dentro de la sesión del usuario, la petición no debería realizarse y el evento debe registrarse como un posible ataque CSRF.

Para mejorar aún más la seguridad de este diseño propuesto, podríamos considerar la posibilidad de aleatorizar el nombre del parámetro del token CSRF y el valor para cada petición. La implementación de este enfoque da como resultado la generación de tokens por petición en lugar de tokens por sesión. Este enfoque resulta más seguro que los tokens por sesión, ya que se reduce la ventana temporal durante la cual un posible atacante puede acceder a los tokens.

#### 5.10.30 Ejecutar código JavaScript de forma aislada

El módulo `vm` <https://nodejs.org/api/vm.html> proporciona una forma segura de ejecutar código JavaScript de manera aislada. Este módulo ofrece acceso a una nueva máquina virtual V8 en la que puede ejecutar el código JavaScript que necesite ejecutar independiente del flujo principal.

Cuando se usa `script.runInThisContext()` o `vm.runInThisContext()`, el código se ejecuta dentro del contexto global actual del motor de V8. El código pasado a este contexto de VM tendrá su propio alcance y se ejecutará de forma aislada del resto.

Podemos ejecutar un servidor web utilizando el módulo `http`, donde el código que se pasa al contexto debe llamar a `require('http')` por sí mismo o bien se debe pasar una referencia al módulo `http`:

```
const vm = require('vm');

const code = `
({require} => {
  const http = require('http');

  http.createServer((request, response) => {
    response.writeHead(200, { 'Content-Type': 'text/plain' });
    response.end('Hello World\\n');
  }).listen(8124);

  console.log('Server running at http://127.0.0.1:8124/');
});

vm.runInThisContext(code)(require);
```

Cuando se llama al método `vm.createContext()`, el objeto de sandbox que se pasa se asocia internamente con una nueva instancia de un contexto V8. Este contexto V8 proporciona el código ejecutado utilizando los métodos del módulo `vm` con un entorno global aislado dentro del cual puede operar.

#### 5.10.31 Uso de componentes con vulnerabilidades conocidas

Los componentes, como librerías y otros módulos de software, casi siempre se ejecutan con privilegios de administrador. Si se explota un componente vulnerable, tal ataque puede facilitar la pérdida de datos o la toma de control del servidor. Las aplicaciones que utilizan componentes con vulnerabilidades conocidas permiten una variedad de posibles ataques.

Podríamos realizar un análisis de las librerías utilizadas por la aplicación con el objetivo de detectar posibles vulnerabilidades que afecten a librerías y dependencias. Para ello, se puede tomar como referencia la base de datos de

vulnerabilidades CVE (Common Vulnerabilities and Exposures) utilizando los servicios de búsqueda de la NVD (National Vulnerability Database).

El uso de paquetes npm inseguros puede provocar esta vulnerabilidad. Algunos proyectos que podemos encontrar dentro del ecosistema de NodeJS ayudan a probar y alertar sobre dependencias inseguras. Entre estos proyectos, podemos destacar:

- Podemos consultar las **últimas actualizaciones de seguridad** en el sitio web oficial del proyecto:  
<https://expressjs.com/en/advanced/security-updates.html>
- El proyecto **Node Security** <https://nodesecurity.io/advisories> constituye un buen recurso para conocer las vulnerabilidades relacionadas con la plataforma.
- **Snyk.io** <https://snyk.io> es otra herramienta y plataforma CLI para escanear y detectar paquetes vulnerables. Presenta la capacidad para conectarse con los repositorios de GitHub y buscar proyectos que usen NodeJS.
- **npm-check** <https://www.npmjs.com/package/npm-check> permite comprobar si existen en su proyecto dependencias desactualizadas o no utilizadas.
- **David DM** <https://david-dm.org> le ofrece una descripción general de las dependencias de su proyecto, la versión que utiliza y la última versión disponible. Además, si nos instalamos el módulo, podemos ver para cada paquete cuál es la última versión que tenemos y cuál es la última disponible; asimismo, nos muestra el comando que se debe ejecutar para actualizar los paquetes.
- **Krakenjs** <http://krakenjs.com> La seguridad se proporciona mediante el módulo **Lusca**, que actúa como Middleware a nivel de seguridad para Express y sigue las mejores prácticas de OWASP.

```
app.use(lusca.csrf());
app.use(lusca.csp({ /* ... */}));
app.use(lusca.xframe('SAMEORIGIN'));
app.use(lusca.p3p('ABCDEF'));
app.use(lusca.hsts({ maxAge: 31536000 }));
app.use(lusca.xssProtection(true));
app.use(lusca.nosniff());
```

Figura 5.26 Métodos del módulo Lusca para añadir cabeceras de seguridad.

El módulo Lusca permite, entre otras cosas:

- **Habilitar cabeceras CSRF.**
- **Habilitar cabeceras X-FRAME-OPTIONS**, para ayudar a prevenir clickjacking.
- **Habilitar cabeceras X-XSS-Protection**, para ayudar a prevenir ataques de cross-site scripting (XSS).

Los paquetes npm son parte esencial de nuestra aplicación en NodeJS. Tales paquetes pueden contener, de manera accidental o malintencionada, un código inseguro. A través de paquetes inseguros, un atacante puede ejecutar acciones como:

- Crear y ejecutar scripts en diferentes etapas durante la instalación o el uso del paquete.
- Leer, escribir, actualizar o borrar archivos en el sistema.
- Escribir y ejecutar archivos binarios.
- Recopilar datos confidenciales de la aplicación y enviarlos de forma remota a otro servidor externo.

Estas son algunas medidas que podemos tomar para protegernos contra paquetes maliciosos de npm:

- No ejecutar la aplicación con privilegios de root.
- Usar paquetes que incluyan análisis de código estático. Podríamos ejecutar herramientas como JSHint/JSLint para obtener qué porcentaje del código cumple las reglas relacionadas con la calidad del código.
- Usar paquetes que contengan pruebas unitarias completas y pruebas de revisión para las funciones que utiliza nuestra aplicación.
- Revisar el código para cualquier archivo inesperado o acceso a la base de datos.
- Investigar cómo de popular es un paquete que queremos incluir, ver comentarios y si el autor ha escrito otros módulos.
- Tener siempre actualizadas las librerías de las que depende el proyecto, debido a que la mayoría de los proyectos que gestionan el desarrollo de estas librerías no proporcionan parches de seguridad para sus vulnerabilidades, sino que, simplemente, arreglan el problema en las siguientes versiones. Por eso, la actualización a las versiones más recientes resulta crítica.

- Si no se puede actualizar a versiones más nuevas por problemas de compatibilidad, se debería considerar la adición de envoltorios de seguridad alrededor de los componentes, para deshabilitar la funcionalidad vulnerable o asegurar los aspectos débiles o vulnerables del componente.

#### 5.10.32 NodeJsScan

NodeJsScan (<https://github.com/ajinabraham/NodeJsScan>) es una herramienta de análisis estático que puede detectar posibles problemas de seguridad, código inseguro y bibliotecas obsoletas. Básicamente, se trata de un script en Python que, al ejecutar, devuelve un informe con todo lo que ha encontrado que puede originar problemas de seguridad.

Lo que hace es comprobar una serie de reglas de seguridad que están definidas en un fichero que podemos encontrar en el repositorio del proyecto en GitHub: <https://github.com/ajinabraham/NodeJsScan/blob/master/core/rules.xml>

```
<!-- all string comparison rules go here -->
<rule name="Express BodyParser tempfile creation issue">
  <signature>bodyParser()</signature>
  <description>POST Request to express body parser "bodyparser()" can create temporary files and consume space.</description>
  <tags>nodejs/tags</tags>
</rule>
<rule name="Handlebars unescaped String">
  <signature>handlebars.safestring</signature>
  <description>Handlebars safestring will not escape the data passed through it. Untrusted user input passing through safestring can
  <tags>nodejs/tags</tags>
</rule>
<!-- All Regex rules go here -->
<rule name="Server-side injection[SQL] - eval()">
  <signature>(eval|C|.(0,40000))(req|resp|query|req|body|req|param)</signature>
  <description>User controlled data in eval() can result in server side injection (SSI) or Remote Code Execution (RCE).</description>
  <tags>sql/rce</tags>
</rule>
<rule name="Server-side rejection[SSI] - settimeout()">
  <signature>(settimeout|((.|0,40000))(req|.|req|query|req|body|req|param))</signature>
  <description>User controlled data in 'settimeout()' can result in Server Side Injection (SSI) or Remote Code Execution (RCE).</description>
  <tags>sql/rce</tags>
</rule>
```

Figura 5.27 Reglas de seguridad definidas en NodeJsScan.



Figura 5.28 Interfaz de usuario de NodeJsScan.

Por ejemplo, permite detectar vulnerabilidades del tipo XSS, donde no estamos validando correctamente las entradas del usuario.

The screenshot shows a security scan result for a file named 'hpp.js' located at the path 'b5036571f35a061e54ec30ee6fe205a35cd3bcd5c767d120558061189689cc0f/Node.Js-Security-Course/hpp.js'. The issue is identified as 'XSS - Reflected Cross Site Scripting'. The code snippet below contains a vulnerability:

```
var express = require('express');
var app = express();
app.get('/', function(req, res) {
    res.send('ids ' + req.query.id);
    console.log("GET / id=" + req.query.id);
});
app.listen(3000);
```

Figura 5.29 Vulnerabilidad detectada XSS.

Podemos desplegar la aplicación en nuestra máquina local a través de la imagen de Docker, que podemos encontrar en el repositorio DockerHub: <https://hub.docker.com/r/opensecurity/nodejsscan/>

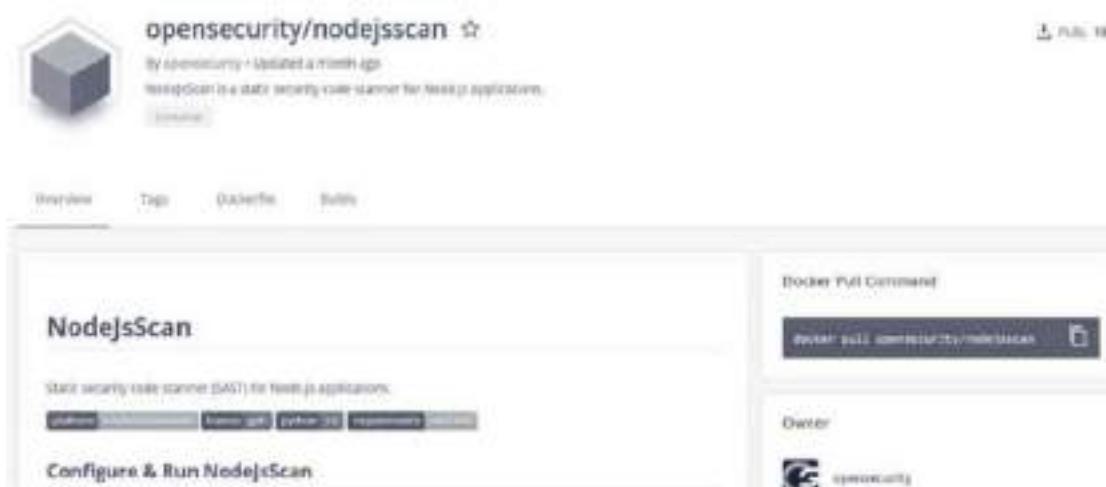


Figura 5.30 Imagen en DockerHub.

Además, podemos ver los resultados del análisis de un fichero en concreto en formato json.

```
[INFO] Running Static Code Analysis on ./Node.js-Security-Course/deserialization.js
{
  "description": "User controlled data in 'serialize()' or 'deserialize()' function can result in Object Inject Remote Code Injection",
  "filename": "deserialization.js",
  "line": 31,
  "lines": "app.use(cookieParser())\napp.get('/', function(req, res) {\n  if (req.cookies.profile) {\n    var str = new Buffer(req.cookies.profile, 'base64').toString();\n    var obj = serialize.deserialize(str);\n    if (obj.username) {\n      res.send('Hello ' + escape(obj.username));\n    } else {\n      res.send('User profile not found');\n    }\n  } else {\n    res.send('User profile not found');\n  }\n});\n",
  "path": "./Node.js-Security-Course/deserialization.js",
  "sha2": "04f398ff3deed27aeb5995fa27abc7722899a33518c24644af97789d8777cbe5b",
  "tag": "fixed",
  "title": "Deserialization Remote Code Injection"
}
```

Figura 5.31 Resultados del análisis de un fichero JavaScript.

## Capítulo 6. Seguridad en proyectos Python

Python es un lenguaje que permite escalar fácilmente de proyectos iniciales a aplicaciones complejas para procesar datos y servir páginas web dinámicas. Pero, a medida que aumenta la complejidad de sus aplicaciones, puede ser fácil presentar problemas de seguridad y vulnerabilidades. En este capítulo analizamos los principales problemas que podemos encontrar en las funciones de Python, cómo evitarlos, y las herramientas y los servicios que nos pueden ayudar a identificar vulnerabilidades en el código fuente.

Python tiene un sistema de tipos con una tipificación fuerte, lo que significa que cada cambio de tipo requiere una conversión explícita. Una cadena que contiene solo dígitos no se convierte automáticamente en un número, como puede suceder en otros lenguajes como Perl o JavaScript.

Desde el punto de vista de la seguridad, se puede decir que existen otros lenguajes más seguros que tienen tipado estático, como C, C++, pero la seguridad es mayor si la comparamos con lenguajes de tipado débil, como JavaScript o PHP.

Que Python tenga tipado dinámico significa que se puede cambiar el tipo de una variable, y fuerte significa que no se pueden combinar tipos; por ejemplo, si intenta sumar un número y una cadena, devuelve un error:

```
>>> x = 5
>>> y = '5'
>>> print(x+y)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Mientras que, en JavaScript, la misma operación de sumar se realiza de forma correcta, ya que realiza una conversión explícita de la variable x de entero a cadena y lo que hace es realizar una concatenación de cadenas:

```
var x = 5
var y = '5'
console.log(x + y) //Devuelve "55"
```

Esa constituye la principal diferencia entre tipado débil y tipado fuerte. Los tipos débiles, automáticamente, intentan convertir de un tipo a otro, depen-

diendo del contexto (por ejemplo, JavaScript), mientras que el tipado fuerte no permite convertir implícitamente.

El tipado en Python es dinámico, lo que significa que a las variables se les asignará el tipo de dato en tiempo de ejecución. Gracias al tipado dinámico podemos, en el mismo bloque de código, asignar diferentes tipos de datos a la misma variable.

Con Python, también resulta posible indicarle un tipado estático, de forma que puede usar herramientas como **mypy**:

<https://mypy.readthedocs.io/en/latest/index.html>, para hacer análisis estático en Python.

El objetivo de mypy reside en analizar el código en busca de anotaciones basadas en tipado estático y le dice si el código presenta errores desde el punto de vista de la comprobación de tipos. Para que mypy detecte dicha clase de errores, se necesita añadir anotaciones de tipo; por ejemplo, podemos definir una función que acepte un string por parámetro y devuelva como resultado la concatenación con otra cadena:

```
def concatenar(nombre: str) -> str:  
    return 'Hola' + nombre
```

En este punto, mypy puede usar las sugerencias de tipo proporcionadas para detectar usos incorrectos de la función; por ejemplo, las siguientes llamadas darán error, ya que los argumentos se componen de tipos no válidos:

```
concatenar(1)  
# Argument 1 to "concatenar" has incompatible type "int"; expected "str"  
concatenar(b'User')  
# Argument 1 to "concatenar" has incompatible type "bytes"; expected "str"
```

La ventaja de usar tales anotaciones de tipado estático radica en que se puede detectar de una forma rápida cualquier llamada a una función con argumentos de tipo erróneo.

La codificación segura es la práctica del desarrollo de software que protege a los programas contra las vulnerabilidades de seguridad y lo hace resistente a los ataques maliciosos, desde el diseño del programa hasta su implementación y puesta en producción. Básicamente, se trata de escribir código intrínsecamen-

te seguro en lugar de pensar en la seguridad como una capa que se añade más adelante. La codificación segura incluye todas estas ramas de *testing*:

- Análisis de las arquitecturas involucradas
- Revisión de detalles de implementación
- Verificación de la lógica del código
- Pruebas operacionales unitarias y de caja blanca
- Pruebas funcionales de caja negra

A lo largo de este capítulo, encontrará algunos scripts en Python, cuyo código fuente se encuentra disponible en el repositorio de GitHub, organizados por carpetas en función de la sección: [https://github.com/jmortega/testing\\_python\\_security](https://github.com/jmortega/testing_python_security)

■ command injection	testing python security	8 months ago
■ crypto	Testing python security	2 hours ago
■ flask	Testing python security	2 hours ago
■ imágenes	testing python security	6 months ago
■ pickle	Testing python security	2 hours ago
■ requests	Testing python security	2 hours ago
■ sql injection	testing python security	8 months ago
■ subprocess	Testing python security	2 hours ago
■ ss	testing python security	8 months ago

Figura 6.1 Repositorio de GitHub scripts en Python.

Este capítulo también se puede complementar con conferencias y presentaciones impartidas por el autor y disponibles en el espacio de *speaker deck*.

<https://speakerdeck.com/jmortega/testing-python-security>

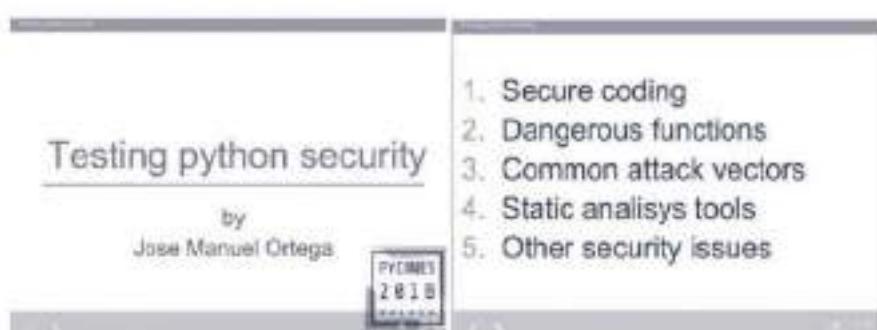


Figura 6.2 Presentación realizada en una conferencia de Python.

## 6.1 Componentes inseguros en Python

En la figura 6.3 se presenta una lista de los componentes de Python inseguros más conocidos, donde podemos destacar funciones como eval, pickle y los módulos os, subprocess o yaml.

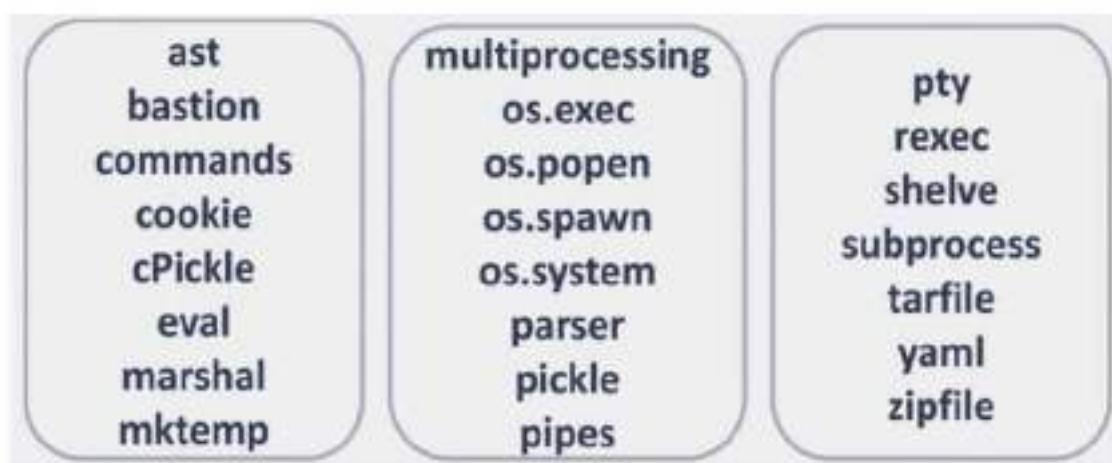


Figura 6.3 Funciones inseguras en Python.

Algunos módulos de Python, como el de **subprocess**, si se usan de forma insegura, pueden suponer un riesgo para la aplicación. La documentación de Python, normalmente, incluye una nota sobre los principales riesgos si, por ejemplo, usamos el parámetro shell=True al intentar ejecutar un comando con la función **subprocess.call()**:

- <https://docs.python.org/3.5/library/subprocess.html#security-considerations>

**Warning:** Executing shell commands that incorporate unsanitized input from an untrusted source makes a program vulnerable to shell injection, a serious security flaw which can result in arbitrary command execution. For this reason, the use of `shell=True` is **strongly discouraged** in cases where the command string is constructed from external input:

```
>>> from subprocess import call
>>> filename = input("What file would you like to display?\n")
What file would you like to display?
name_exists: no -rf / *
>>> call(["cat " + filename, shell=True]) # Oh-oh. This will end badly..
```

`subprocess` disables all shell based features, but does not suffer from this vulnerability; see the Note in the `Popen` constructor documentation for helpful hints in getting `subprocess` to work.

When using `subprocess`, `subprocesses()` can be used to properly escape whitespace and shell metacharacters in strings that are going to be used to construct shell commands.

Figura 6.4 Uso inseguro del módulo subprocess.

Entre los principales problemas relacionados con la seguridad, podemos destacar:

- Uso de funciones de Python peligrosas, como `eval()`, sin la correcta validación de las cadenas de entrada.
- Acceso al sistema de archivos.
- Los fragmentos de código SQL y JavaScript integrados en el código fuente o las plantillas, especialmente si estamos utilizando Django o Flask. En este punto, se recomienda seguir las mejores prácticas usando mecanismos de persistencia como **django ORM**:  
<https://docs.djangoproject.com/en/2.2/topics/db/>, para persistir en las plantillas de base de datos y evitar la inyección de SQL y XSS.
- Claves API hardcodeadas en el código fuente.
- Llamadas HTTP a servicios web internos o externos.

## 6.2 Validación incorrecta de entrada/salida

La validación y el saneamiento de entradas y salidas representa uno de los problemas más críticos y frecuentes que podemos encontrar hoy día y que originan más del 70 % de vulnerabilidades de seguridad, donde los atacantes pueden hacer que un programa acepte información maliciosa, como datos de código o comandos del sistema que, cuando se ejecutan, pueden comprometer un sistema. En el ejemplo de la figura 6.5, el parámetro `arg` se pasa a una función considerada como insegura sin realizar ningún tipo de validación.

```
def main():
    for arg in sys.argv[1:]:
        os.system(arg)
```

Figura 6.5 Validación incorrecta de parámetros de entrada.

Una aplicación que apunta a mitigar este tipo de ataque debe tener filtros para verificar y eliminar contenido que sea malicioso y solo aceptar datos que sean razonables y seguros para la aplicación.

### 6.3 Función eval()

Python dispone de la función eval(), que permite evaluar una cadena de código Python. Se trata de la función más peligrosa, si se acepta cualquier tipo de cadena de entrada que no se esté validando correctamente.

Considere una situación donde está utilizando el módulo os (operating system), que proporciona una forma rápida de usar funcionalidades del sistema operativo, como leer o escribir un archivo. Si, además, permitimos a los usuarios ingresar un valor mediante la función input() y tratamos de evaluar la cadena de entrada usando eval(input()), el usuario podría introducir un comando cuyo objetivo sea cambiar el archivo o incluso eliminar todos los archivos del sistema de archivos usando el comando **os.system ('rm -rf \*)**, en el caso de conseguir los permisos adecuados.

Si está utilizando la llamada eval(input()), resulta una buena práctica verificar qué variables y métodos puede utilizar el usuario. Puede ver qué variables y métodos se hallan disponibles usando el método dir().

```
>>> print(eval('dir()'))
['__annotations__', '__builtins__', '__doc__', '__loader__',
 '__name__', '__package__', '__spec__', 'os']
>>> -
```

Figura 6.6 Obtener variables y métodos disponibles por defecto.

Afortunadamente, la función eval dispone de una serie de argumentos opcionales para restringir lo que puede ejecutar:

```
eval(expression[, globals[, locals]])
```

La función eval() acepta como segundo argumento la variable **globals**, que son los valores globales para usar durante la evaluación. Si no proporciona un diccionario global, eval utiliza los globales actuales. Además, si se ofrece un diccionario vacío {} como segundo parámetro, entonces no habrá definido ningún global.

De esta forma, puede hacer que la evaluación de una expresión sea segura al ejecutarla sin elementos globales. El siguiente comando genera un error al intentar ejecutar el comando **os.system('clear')** y pasar un diccionario vacío en el parámetro **globals**:

```
eval("os.system('clear')", {})

>>> eval("os.system('clear')", {})
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "<string>", line 1, in <module>
NameError: name 'os' is not defined
```

Figura 6.7 Ejecución de un comando de sistema sin variables globales.

Si queremos que el comando anterior funcione, lo podemos hacer añadiendo el import correspondiente del módulo os:

```
eval("__import__('os').system('clear')", {})
```

El siguiente paso para hacer que la función eval() resulte más segura consiste en deshabilitar el uso de los imports. Para ello, se han de deshabilitar las funciones builtins a nivel global que vienen por defecto. Podemos indicar explícitamente que las funciones builtins no se encuentren disponibles, para lo que hay que ese nombre como un diccionario vacío en el parámetro globals:

```
eval("__import__('os').system('clear')", {'__builtins__': {}})
```

Al deshabilitar los builtins, hacemos que no podamos utilizar el import y que devuelva un error.

```
>>> eval("__import__('os').system('clear')", {'__builtins__': {}})
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "<string>", line 1, in <module>
NameError: name '__import__' is not defined
>>>
```

Figura 6.8 Ejecución de un comando de sistema eliminando builtins.

La mayoría de las veces, todos los métodos y las variables disponibles utilizados en la expresión eval pueden no ser necesarios o incluso pueden presentar un

agujero de seguridad. Es posible que quiera restringir el uso de estos métodos y variables para eval(). Puede hacerlo pasando parámetros globales y locales opcionales al método eval(). Si se omiten ambos parámetros, la expresión se ejecuta en el ámbito actual. Si se omite el diccionario local, se utiliza de forma predeterminada el diccionario global; es decir, las variables globales se usarán para variables globales y locales.

En el ejemplo de la figura 6.9, estamos pasando un diccionario vacío como parámetro global. Si se pasa un diccionario vacío como globales, solo los `_builtins_` aparecen disponibles para usarse como expresión en el primer parámetro de la función eval(). Aunque hemos importado el módulo os (operating system), la expresión no puede acceder a ninguna de las funciones proporcionadas por el módulo os, ya que el import se ha realizado fuera del contexto de la función eval.

```
>>> print(eval('dir()',{}))
['__builtins__']
>>> import os
>>> eval("os.system('clear')",{})
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "<string>", line 1, in <module>
NameError: name 'os' is not defined
```

Figura 6.9 Ejecución de un comando con el módulo importado en un contexto diferente.

Podemos hacer que las funciones y variables necesarias se hallen disponibles para su uso al pasar el diccionario local. En este programa, la expresión (primer parámetro de la evaluación) puede hacer uso del método sqrt() y la variable 'a'. Todos los demás métodos y variables no estarán disponibles.

```
>>> from math import *
>>> a = 4
>>> print(eval('sqrt(a)', {'__builtins__': None},
{'a': a, 'sqrt': sqrt}))
2.0
```

Figura 6.10 Ejecución de la función eval con las funciones necesarias.

En este caso, al intentar usar el método pow(), nos devolvería que el método no está definido, ya que no se encuentra definido dentro de los métodos permitidos.

```
>>> print(eval('pow(a)', {'__builtins__':None}, {'a':a, 'sqrt':sqrt}))  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
  File "<string>", line 1, in <module>  
NameError: name 'pow' is not defined
```

Figura 6.11 Ejecución de la función eval con funciones sin definir.

De esta forma, estamos mejorando el uso de la función eval(), al restringir su uso a aquello que definamos en los diccionarios global y local. Las conclusiones finales sobre el uso de la función eval son que no se recomienda para evaluar valores de entrada, pero, si tiene que usarla, es importante hacerlo a partir de fuentes de entrada validadas y confiables.

También podríamos hacer uso de expresiones regulares para evitar entradas controladas por el usuario no neutralizadas en la evaluación de código dinámico:

```
import re  
python_code = input()  
  
pattern =re.compile("entrada_valida")  
if pattern.fullmatch(python_code):  
    eval(python_code)
```

Como alternativa a la función eval, tenemos la función `literal_eval()`, perteneciente al módulo `ast` [https://docs.python.org/3/library/ast.html#ast.literal\\_eval](https://docs.python.org/3/library/ast.html#ast.literal_eval), que permite evaluar, de forma segura, una expresión o una cadena de Python. La cadena proporcionada solo puede contener las siguientes estructuras de datos de Python: strings, bytes, numbers, tuples, lists, dicts, sets o booleans.

## 6.4 Serialización y deserialización de datos con pickle

El módulo `pickle`, de la biblioteca estándar de Python, ofrece herramientas para convertir objetos de Python en flujos de bytes, y viceversa. El módulo `pickle` es específico de Python y, gracias a las capacidades de introspección de Python,

admite una gran cantidad de tipos de datos. El tratamiento de datos con pickle a partir de una fuente de datos no confiable puede ser considerado como algo peligroso, si decide usar dicho módulo en su aplicación. En la documentación del módulo podemos ver el mensaje que muestra en este aspecto <https://docs.python.org/3.7/library/pickle.html>

## 12.1. pickle — Python object serialization

**Source code:** Lib/pickle.py

The `pickle` module implements binary protocols for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as “serialization”, “marshalling,” [1] or “flattening”; however, to avoid confusion, the terms used here are “pickling” and “unpickling”.

**Warning:** The `pickle` module is not secure against erroneous or maliciously constructed data. Never unpickle data received from an untrusted or unauthenticated source.

Figura 6.12 Uso inseguro del componente pickle.

El módulo pickle implementa protocolos binarios para serializar y deserializar una estructura de objetos Python, de forma que le permite almacenar el estado de los objetos para, posteriormente, restaurarlo. Pickle resulta útil para almacenar algo que no necesita una base de datos o para datos que son inherentemente temporales.

En pickle, para serializar una jerarquía de objetos, llamamos a la función `pickle.dumps()`. De manera similar, para deserializar un flujo de datos, debe llamar a la función `pickle.loads()`. La vulnerabilidad se produce cuando la entrada suministrada por el usuario no se encuentra correctamente validada antes de pasarla a las funciones `yaml.load()`, `pickle.load()` o `pickle.loads()`.

Estas entradas no seguras pueden provenir de archivos de configuración o proporcionarse a través de las API REST; por ejemplo, normalmente usamos YAML para los archivos de configuración, pero los archivos YAML también pueden contener código Python incorporado. Esto puede proporcionar a un atacante un método para ejecutar código:

```
import yaml
import pickle
conf_str = ""
!!python/object:_main_.AttackerObj
key: 'value'
"
conf = yaml.load(conf_str)
```

En cuanto a la seguridad, pickle presenta los mismos problemas que las funciones exec y eval. Faculta a los usuarios para crear información que ejecuta código arbitrario. Como Python, permite la serialización de objetos; los atacantes podrían pasar cadenas serializadas *ad hoc* a una llamada vulnerable, lo que resultaría en una inyección arbitraria de código en el ámbito de la aplicación.

Algunos de los problemas que nos encontramos al trabajar con pickle son estos:

- Inyección de código y corrupción de datos.
- No hay controles sobre la integridad de los datos/objetos.
- No hay control sobre el tamaño de los datos.
- Código evaluado sin controles de seguridad.

La solución para evitar tales problemas pasa por usar métodos alternativos más seguros cuando necesitemos serializar/deserializar datos, como usar los módulos JSON/YAML o el uso de métodos seguros como `yaml.safe_load()`. En estos ejemplos, estamos usando dicho método para serializar, de forma segura, un fichero y una cadena.

En tales ejemplos, estamos usando el método `safe_load()`, para serializar de forma segura un fichero y una cadena:

```
import yaml
user_input = input()
with open(user_input) as file:
    contents = yaml.safe_load(file)
```

```
import yaml
conf_str = ""
- key: 'value'
- key: 'value'
"
conf = yaml.safe_load(conf_str)
```

Otro de los problemas que tenemos con pickle reside en que podemos **sobrescribir el método reduce**. Se recurre a ese método cuando se realiza la deserialización de un objeto pickle y podríamos ejecutar cualquier código malicioso en dicha función. En este caso, al ejecutar la deserialización, se ejecuta el método reduce, que listaría los ficheros del directorio actual.

El siguiente código lo podemos encontrar en el repositorio de GitHub en la ruta:

[https://github.com/jmortega/testing\\_python\\_security/blob/master/pickle/pickle\\_vulnerable.py](https://github.com/jmortega/testing_python_security/blob/master/pickle/pickle_vulnerable.py)

```
import os
import pickle

class Vulnerable(object):
    def __reduce__(self):
        # Note: this will only list files in your directory.
        return (os.system, ('ls',))

def serialize_exploit():
    shellcode = pickle.dumps(Vulnerable())
    return shellcode

def insecure_deserialize(exploit_code):
    pickle.loads(exploit_code)

if __name__ == '__main__':
    shellcode = serialize_exploit()
    print('Obtaining files...')
    insecure_deserialize(shellcode)
```

Si su aplicación realmente depende del uso del módulo pickle, puede usar otras estrategias de mitigación, como una jaula chroot o una sandbox, para evitar los efectos negativos de la ejecución de código malicioso que no haya sido validado correctamente. El siguiente código empaqueta una clase **ShellSystemJail**, que implementa una jaula chroot, la cual evita la ejecución de código en la carpeta raíz a la hora de deserializar y llamar al método reduce.

El siguiente código lo podemos encontrar en el repositorio de GitHub en la ruta:

[https://github.com/jmortega/testing\\_python\\_security/blob/master/pickle/pickle\\_safe\\_chroot\\_jail.py](https://github.com/jmortega/testing_python_security/blob/master/pickle/pickle_safe_chroot_jail.py)

```

import os
import pickle
from contextlib import contextmanager

class ShellSystemJail(object):
    def __reduce__(self):
        # this will list contents of root / folder
        return (os.system, ('ls -al /',))

@contextmanager
def system_jail():
    """ A simple chroot jail """
    os.chroot('safe_root/')
    yield
    os.chroot('/')

def serialize():
    with system_jail():
        shellcode = pickle.dumps(ShellSystemJail())
    return shellcode

def deserialize(exploit_code):
    with system_jail():
        pickle.loads(exploit_code)

if __name__ == '__main__':
    shellcode = serialize()
    deserialize(shellcode)

```

## 6.5 Ataques de inyección de entrada

Una inyección es un uso no controlado de la información del usuario que llega a un intérprete. Esto podría ser un intérprete de SQL, la línea de comandos o el intérprete de Python. Si existe una vulnerabilidad de inyección, el usuario puede enviar comandos arbitrarios al servidor y, posiblemente, acceder o modificar los datos sin autorización. A continuación se presentan dos subgrupos de ataques de inyección, inyecciones de SQL e inyecciones de comandos:

- La **inyección de comandos** se puede producir cuando estamos llamando a un proceso utilizando las funciones `popen`, `subprocess` u `os.system` y pasando argumentos de las variables de entrada. Al usar

tales funciones, existe la posibilidad de que alguien use las variables de entrada para ejecutar código malicioso.

- La **inyección de SQL** es donde escribe consultas SQL directamente, en lugar de usar un ORM y mezclar los literales de cadena con variables. Lo más importante en este punto reside en familiarizarse con todas las formas complejas de inyección de SQL.

### 6.5.1 Inyección de comandos

La inyección de comandos permite a un atacante injectar código malicioso en una shell del sistema. Incluso las aplicaciones web utilizan programas de línea de comandos para mayor comodidad y funcionalidad. Tales procesos se ejecutan normalmente dentro de una shell; por ejemplo, si desea mostrar todos los detalles de un archivo cuyo nombre es dado por el usuario, una posible implementación sería la siguiente:

```
os.system("ls -l {}".format(filename))
```

Las vulnerabilidades de ejecución remota de comandos permiten a los atacantes ejecutar código arbitrario en sus servidores. Las inyecciones de comandos son muy similares a las inyecciones de SQL, pero, en lugar de injectar comandos en un sistema de base de datos, este ataque hace posible injectar comandos en la shell del servidor. La capacidad de ejecutar comandos en un servidor abre posibilidades para leer archivos de contraseñas, eliminar archivos del sistema y otras operaciones peligrosas:

```
>>> filename = 'somefile; rm -rf ~'  
>>> command = 'ls -l {}'.format(filename)  
>>> print(command)  
>>> ls -l somefile; rm -rf ~
```

¿Qué sucede si alguien proporciona un archivo con un nombre como <nombre\_fichero>; rm -rf /? Si la máquina host ejecuta el proceso de Python como un usuario privilegiado, se podrían llegar a eliminar todos los archivos de la máquina. Tener acceso al sistema de archivos resulta, obviamente, peligroso, ya que un usuario malintencionado podría ejecutar comandos arbitrarios en el servidor.

En este punto, la función más peligrosa que podríamos usar es la de `subprocess.call(command, shell = True)`, donde el parámetro `command` puede ser el input de entrada del usuario: a partir de un cuadro de texto en un formulario, se pasa directamente a la llamada. Al introducir una cadena que comienza con un separador de comando, se puede interactuar directamente con la shell del sistema. **El problema con subprocess.call reside en que el comando no se está escapando antes de ser ejecutado por la shell.**

El siguiente código lo podemos encontrar en el repositorio de GitHub en la ruta: [https://github.com/jmortega/testing\\_python\\_security/blob/master/command%20injection/vulnerable\\_command\\_injection.py](https://github.com/jmortega/testing_python_security/blob/master/command%20injection/vulnerable_command_injection.py)

```
@app.route('/menu',methods=['POST'])
def menu():
    param = request.form [ ' suggestion ' ]
    command = ' echo ' + param + '>> ' + ' menu.txt '
    subprocess.call(command,shell = True)
    with open('menu.txt','r') as f:
        menu = f.read()
    return render_template('command_injection.html', menu = menu)
```

La mejor práctica en este punto consiste en ejecutar la llamada con el parámetro `shell = False`:

```
@app.route('/menu',methods=['POST'])
def menu():
    param = request.form [ ' suggestion ' ]
    command = ' echo ' + param + '>> ' + ' menu.txt '
    subprocess.call(command,shell = False)
    with open('menu.txt','r') as f:
        menu = f.read()
    return render_template('command_injection.html', menu = menu)
```

Python ofrece otras alternativas para sanitizar o escapar la entrada. En Python, si necesita escapar de la entrada, puede usar el módulo `shlex`, que está integrado en la biblioteca estándar y posee una función de utilidad para escapar de los comandos de shell usando el método `shlex.quote()` <https://docs.python.org/3/library/shlex.html#shlex.quote>

Esta función devuelve una versión **shell escaped** de la cadena de entrada. El valor devuelto es una cadena que puede usarse de forma segura como un token en una línea de comandos de shell.

```
>>> from shlex import quote
>>> command = 'ls -l {}'.format(quote(filename))
>>> print(command)
ls -l 'somefile; rm -rf ~'
>>> remote_command = 'ssh home {}'.format(quote(command))
>>> print(remote_command)
ssh home 'ls -l 'somefile; rm -rf ~'`
```

Figura 6.13 Uso del módulo **shlex** para escapar parámetros de entrada.

También podríamos construir nuestra propia clase, que permita ejecutar comandos complejos de forma segura utilizando el método de **subprocess.Popen()**. La función **exec\_cmd()** lee, de forma recursiva, el resultado del proceso actual y lo escribe en el descriptor del archivo de entrada del siguiente proceso. Un error en cualquiera de los procesos llamados da como resultado que la función genere una excepción **CalledProcessError**.

El siguiente código lo podemos encontrar en el repositorio de GitHub en la ruta: [https://github.com/jmortega/testing\\_python\\_security/blob/master/command%20injection/PyExecCmd.py](https://github.com/jmortega/testing_python_security/blob/master/command%20injection/PyExecCmd.py)

```
import shlex
import subprocess

class PyExecCmd(object):
    """
    Helper class to run a complex command through Python subprocess
    """

    def __init__(self):
        return

    def exec_cmd(self, cmdstr, *args, **kwargs):
        """ Safely execute the command passed as the string using
        Popen invocation without shell=True. The command may contain
        multiple piped commands. Returns the <stdout> and <stderr> of
        executing the command.

        Args:
            @param cmdstr: type string
        Returns:
            tuple
        """
        p = subprocess.Popen(cmdstr, shell=True, stdout=subprocess.PIPE,
                            stderr=subprocess.PIPE, *args, **kwargs)
        return p.stdout.read(), p.stderr.read()
```

```

    allcmds = cmdstr.split(' ')
    numcmds = len(allcmds)

    popen_objs = []
    for i in range(numcmds):
        scmd = shlex.split(allcmds[i])
        stdin = None if i == 0 else popen_objs[i-1].stdout
        stderr = subprocess.STDOUT if i < (numcmds - 1) else subprocess.PIPE

        thiscmd_p = subprocess.Popen(scmd, stdin=stdin,
                                     stdout=subprocess.PIPE,
                                     stderr=stderr, *args, **kwargs)
        if i != 0: popen_objs[i-1].stdout.close()
        popen_objs.append(thiscmd_p)

    # Collect output from the final command
    (cmdout, cmderr) = popen_objs[-1].communicate()

    # Set return codes
    for i in range(len(popen_objs) - 1):
        popen_objs[i].wait()

    # Now check if any command failed
    for i in range(numcmds):
        if popen_objs[i].returncode:
            raise subprocess.CalledProcessError(popen_objs[i].returncode,
                                                allcmds[i])

    # All commands succeeded
    return (cmdout, cmderr)

```

En el siguiente ejemplo, contamos con dos funciones: una que ejecuta un comando de forma insegura y la otra que ejecuta la misma funcionalidad, pero de manera segura.

El siguiente código lo podemos encontrar en el repositorio de GitHub en la ruta: [https://github.com/jmortega/testing\\_python\\_security/blob/master/subprocess/count\\_lines\\_request.py](https://github.com/jmortega/testing_python_security/blob/master/subprocess/count_lines_request.py)

```
import subprocess

#La función es insegura porque usa shell = True, lo que permite la inyección de shell.
def count_lines_unsafe(website):
    return subprocess.check_output('curl %s | wc -l' % website, shell=True)

#En este caso la función es segura ya que en lugar de llamar a un solo proceso de
#shell que ejecuta cada uno de nuestros programas, los ejecutamos por separado y
#conectamos la salida estándar del comando curl. Indicamos stdout = #subprocess.PIPE,
#para indicar el subprocesso al cual enviar la salida de archivo #correspondiente.

def count_lines_safe(website):
    args = ['curl', website]
    args2 = ['wc', '-l']
    process_curl = subprocess.Popen(args, stdout=subprocess.PIPE,
                                    shell=False)
    process_wc = subprocess.Popen(args2, stdin=process_curl.stdout,
                                 stdout=subprocess.PIPE, shell=False)
    process_curl.stdout.close()
    return process_wc.communicate()[0]

print(count_lines_unsafe('www.google.com & touch file'))
```

El siguiente script usa el módulo subprocess, para ejecutar el comando ping sobre un servidor cuya IP se pasa por parámetro.

El siguiente código lo podemos encontrar en el repositorio de GitHub en la ruta: [https://github.com/jmortega/testing\\_python\\_security/blob/master/subprocess/ping\\_server.py](https://github.com/jmortega/testing_python_security/blob/master/subprocess/ping_server.py)

```
def ping(server):
    return subprocess.check_output('ping -c 1 %s' % server, shell=True)

>>> ping('8.8.8.8')
64 bytes from 8.8.8.8: icmp_seq=1 ttl=58 time=5.82 ms
```

El principal problema de esta llamada radica en que se produce con el **parámetro server**, que es controlado por el usuario, y se podría utilizar para ejecutar comandos arbitrarios; por ejemplo, la eliminación de archivos:

```
>>> ping('8.8.8.8; rm -rf /')
64 bytes from 8.8.8.8: icmp_seq=1 ttl=58 time=6.32 ms
rm: cannot remove '/bin/dbus-daemon': Permission denied
rm: cannot remove '/bin/dbus-uuidgen': Permission denied
rm: cannot remove '/bin/dbus-cleanup-sockets': Permission denied
rm: cannot remove '/bin/cgroups-mount': Permission denied
rm: cannot remove '/bin/cgroups-umount': Permission denied
```

Esta función se puede reescribir de forma segura. En lugar de pasar una cadena a un subprocesso, nuestra función pasa una lista de cadenas. El programa ping obtiene cada argumento por separado (incluso si el argumento tiene un espacio en él), por lo que la shell no procesa otros comandos que proporciona el usuario después de que finaliza el comando ping:

```
def ping(server):
    args = ['ping', '-c', '1', server]
    return subprocess.check_output(args, shell=False)
```

Si probamos esto con la misma entrada que antes, el comando ping interpreta el valor del parámetro server correctamente como un solo argumento y devuelve el mensaje de error host desconocido, ya que el comando añadido (; rm -rf) invalida realizar el ping de forma correcta:

```
>>> ping('8.8.8.8; rm -rf /')
ping: unknown host 8.8.8.8; rm -rf /
```

### 6.5.2 Inyección de SQL

Las inyecciones de SQL se producen cuando una aplicación crea una consulta parcial a partir de la entrada del usuario sin validar dicha entrada. El usuario podría realizar un escape de la consulta y añadir una consulta arbitraria. En este ejemplo, se recoge como parámetro de entrada (param) del usuario el valor que será utilizado para filtrar la consulta:

```
@app.route('/filtering')
def filtering():
    param = request.args.get('param', 'not set')
    Session = sessionmaker(bind = db.engine)
    session = Session()
    result = session.query(User).filter(" username ={} ".format(param))
    for value in result:
        print(value.username , value.email)
    return ' Result is displayed in console.'
```

El problema en este ejemplo estriba en que la cadena de entrada no está siendo validada, el usuario puede ingresar cualquier cadena y un valor de entrada como '`or 1 = 1`' devolverá a todos los usuarios en la base de datos.

Las vulnerabilidades de inyección de SQL facultan a los atacantes para modificar la estructura de las consultas de SQL, de manera que permitan la extracción o manipulación de datos existentes. Si ejecuta este ejemplo contra la base de datos de usuarios, la consulta original se modificará, con lo que se añadirá la posibilidad de eliminar la tabla de usuarios:

```
import sqlite3

def get_user_by_name(name, cursor):
    cursor.execute("SELECT * FROM users WHERE name = '%s'" % name)

if __name__ == '__main__':
    conn = sqlite3.connect('example.db')
    cursor = conn.cursor()
    malicious_name = "User'; DROP TABLE users; --"
    get_user_by_name(malicious_name, conn)
```

La inyección de SQL constituye la segunda vulnerabilidad más común de las aplicaciones web, después de XSS. El ataque implica ingresar código SQL malicioso en una consulta que se ejecuta en la base de datos. Podría resultar en el robo de datos, al descargar contenido de la base de datos, o la destrucción de datos. El siguiente fragmento de código obtiene una dirección de correo a partir del nombre de usuario:

```
name = request.GET['user']
sql = "SELECT email FROM users WHERE username = '{}'".format(name)
```

A primera vista, puede parecer que devolverá solo la dirección de correo correspondiente al nombre de usuario que se ha pasado por el parámetro GET. Sin embargo, si un usuario malintencionado añade la cadena SQL 'OR '1'='1', el código SQL que se ejecuta sería el siguiente:

```
SELECT email FROM users WHERE username = '' OR '1'='1';
```

Dado que esta cláusula WHERE siempre será cierta, se devolverán los correos de todos los usuarios de la base de datos. También se podrían ejecutar consultas más peligrosas, como las siguientes:

```
SELECT email FROM users WHERE username = ""; DELETE FROM users WHERE '1'='1';
```

Una de las soluciones para evitar una inyección de SQL consiste en utilizar alguno de los sistemas ORM que ofrece Python <https://docs.djangoproject.com/en/2.2/topics/db/>. El ejemplo anterior se podría implementar de la siguiente manera:

```
User.objects.get(username=name).email
```

Para prevenir los ataques de inyección de SQL, podemos seguir el siguiente conjunto de buenas prácticas:

- Evitar la concatenación de entradas que no sea de confianza.
- Evitar el código SQL formado por entradas de usuario no validadas previamente.

- Usar consultas parametrizadas.
- A la hora de construir la consulta, la mejor práctica es usar el carácter ? (en lugar de %s) para pasar los valores. De esta forma, se pueden escapar los caracteres especiales de forma automática.

```
def get_user_by_name(name, cursor):
    cursor.execute("SELECT * FROM users WHERE name = (?)", name)
```

Los resultados de un ataque de inyección de SQL exitoso pueden incluir la filtración de información confidencial, como contraseñas de usuario, modificación o eliminación de datos, y la obtención de privilegios, lo que permitiría a un atacante ejecutar comandos arbitrarios en el servidor de la base de datos.

La inyección de SQL normalmente se puede mitigar mediante el uso de una combinación de sentencias preparadas, procedimientos almacenados y el escape de la entrada proporcionada por el usuario.

En este ejemplo estamos usando **sqlalchemy** <https://www.sqlalchemy.org>, que proporciona un mecanismo de sustitución de parámetros que permite reemplazar, de forma segura, la variable ': name' por el valor proporcionado en myvar:

```
import sqlalchemy

connection = engine.connect()
myvar = 'user'
myvar = 'user or 1=1'
query = "select username from users where username = :name"
result = connection.execute(query, name = myvar)
for row in result:
    print "username:", row['username']
connection.close()
```

## 6.6 Acceso seguro al sistema de archivos y ficheros temporales

Desde el punto de vista de la seguridad en el acceso al sistema de archivos, resulta importante evitar que las entradas controladas por el usuario no neutralizadas formen parte de una ruta (archivo o directorio) utilizada en las opera-

ciones de E/S. En este ejemplo vemos cómo las rutas del sistema de archivos no están controladas directamente por entradas controladas por el usuario, sino que pasan previamente por una función que diga si ese path que ha introducido el usuario resulta válido o no. Para ello, podemos comprobar si el path se encuentra dentro de una lista de paths válidos:

```
import requests
import os

valid_paths=['valid_path1','valid_path2']

def getSanitizedPath(path):
    if path in valid_paths:
        return path
    else:
        return ""

def getReport(request):
    reportName = request.field('reportName')
    sanitizedField = getSanitizedPath(reportName)
    file = os.open("usr/local/reports/" + sanitizedField)

    content = requests.get('url_report').text
    getReport(content)
```

Para crear archivos temporales en Python, podríamos utilizar la función **mktemp()** <https://docs.python.org/3/library/tempfile.html#tempfile.mktemp>. El problema de usar dicha función radica en que no es segura, ya que un proceso diferente podría crear un archivo con el mismo nombre en el lapso de tiempo entre la llamada a `mktemp()` y el intento posterior de crear el archivo por el primer proceso, lo que llegaría a exponer los datos temporales al segundo proceso y proporcionar una oportunidad para que un atacante interfiriera en el archivo.

La recomendación para crear ficheros temporales es usar el módulo **tempfile** y el método **mkstemp()** <https://docs.python.org/3/library/tempfile.html#tempfile.mkstemp>.

## 6.7 Inyección de XSS

La inyección de secuencias de comandos entre sitios (XSS) permite a un atacante ejecutar secuencias de comandos maliciosas, normalmente mediante JavaScript, en el sitio web accesible por los usuarios. Para ello, se aprovecha de fallos de seguridad en algunos componentes web, como campos de formulario o redireccionamiento de las url.

Un ejemplo podría ser una sección de comentarios de un sitio web donde un usuario envía un comentario que contiene algo de JavaScript. Cuando otros usuarios ven este comentario, su navegador ejecuta este JavaScript, que puede realizar acciones como acceder a las cookies de los usuarios o redirigir a un sitio malicioso.

En el siguiente ejemplo estamos utilizando Flask como framework para ejecutar nuestro servidor web, que atiende peticiones a través del navegador. En la línea `param = request.args.get('param', 'not set')`, se toma un parámetro de entrada y se devuelve en la respuesta que se muestra al usuario a través del método `make_response()`, que proporciona el framework. En este ejemplo, un usuario malintencionado podría inyectar JavaScript arbitrario que todos los usuarios podrían ejecutar en sus navegadores.

El siguiente código lo podemos encontrar en el repositorio de GitHub en la ruta: [https://github.com/jmortega/testing\\_python\\_security/tree/master/xss](https://github.com/jmortega/testing_python_security/tree/master/xss)

```
from flask import Flask , request , make_response
app = Flask(__name__)

@app.route ('/XSS_param',methods =['GET'])
def XSS():
    param = request.args.get('param','not set')
    html = open('templates/XSS_param.html ').read()
    resp = make_response(html.replace('{{ param}}',param))
    return resp

if __name__ == '__main__':
    app.run(debug = True)
```

Para evitar que nuestra aplicación se vuelva vulnerable a este tipo de ataque, es necesario escapar todas aquellas entradas que impliquen enviar datos de entrada por parte del usuario, ya sea a través de un formulario o a través de la url.

Si estamos trabajando con Flask, una forma sencilla de evitar esta vulnerabilidad consiste en usar el **motor de plantillas** que proporciona el framework. En este caso, el motor de plantillas, a través de la **función escape**, se encargaría de escapar y validar los datos de entrada.

```
from flask import Flask , request , make_response  
  
# using escape function  
from flask import escape  
  
app = Flask(__name__)  
  
@app.route ('/xss_param',methods =['GET' ])  
def XSS():  
    param = escape(request.args.get('param','not set'))  
    html = open('templates/XSS_param.html ').read()  
    resp = make_response(html.replace('{{ param}}',param))  
    return resp  
  
if __name__ == '__main__':  
    app.run(debug = True)
```

Figura 6.14 Usando la función escape de Flask para escapar los datos en entrad.

Cualquier contenido HTML que se genere directamente dentro de un controlador de peticiones debería usar la función de escape adecuada. También podríamos utilizar el método **render\_template\_string** para renderizar nuestro modelo en la vista. En la vista podríamos usar el **filtro de escape | e**, para pintar los datos de nuestro modelo.

El siguiente código lo podemos encontrar en el repositorio de GitHub en la ruta: [https://github.com/jmortega/testing\\_python\\_security/tree/master/xss](https://github.com/jmortega/testing_python_security/tree/master/xss)

```
from flask import Flask  
from flask import request, render_template_string, render_template  
  
app = Flask(__name__)  
  
TEMPLATE = ""  
<html>  
<head><title> Hello {{ person.name | e }} </title></head>  
<body> Hello {{ person.name | e }}</body>  
</html>  
""
```

```
@app.route('/render_template')
def render_template():
    person = {'name':'world', 'secret':
        'jo5gmvllgcZSYZGenWnGcoI8JnwWZd2UZYo=='}
    if request.args.get('name'):
        person['name'] = request.args.get('name')
    return render_template_string(TEMPLATE, person=person)

if __name__ == "__main__":
    app.run(debug=True)
```

## 6.8 Inyección de SSTI

La inyección de plantillas del lado del servidor (SSTI) es un ataque que utiliza las plantillas del lado del servidor que usan frameworks web como Flask. El ataque se aprovecha de puntos débiles en la forma en que la entrada del usuario se incrusta en las plantillas. Los ataques de SSTI se pueden usar para descubrir los aspectos internos de una aplicación web y ejecutar comandos de shell.

En el siguiente ejemplo vemos cómo resulta vulnerable a este ataque debido a que, en el template, estamos concatenando un texto con la información del nombre de usuario.

El siguiente código lo podemos encontrar en el repositorio de GitHub en la ruta: [https://github.com/jmortega/testing\\_python\\_security/tree/master/flask](https://github.com/jmortega/testing_python_security/tree/master/flask)

```
from flask import Flask
from flask import request, render_template_string, render_template

app = Flask(__name__)

@app.route('/hello-flask')
def hello_flask():
    user = {'name':'user', 'secret':'mysecret'}
    if request.args.get('name'):
        user['name'] = request.args.get('name')
    template = '<h2>Hello %s</h2>' % user['name']
    return render_template_string(template, user=user)

if __name__ == "__main__":
    app.run(debug=True)
```

Si ejecutamos nuestro servidor en Flask e intentamos acceder a la url <http://127.0.0.1:5000/hello-flask?name=user%20{{user.secret}}>, somos capaces de obtener el campo secret del objeto user. Al pasar por parámetro {{user.secret}}, el motor de plantillas de Flask permite evaluar el valor de la clave secreta de ese usuario.



Figura 6.15 Accediendo al servidor de Flask a través del EndPoint "hello-flask."

También podríamos construir una url de la forma `http://localhost:5000/hello-flask?name=% for item in user %<p>{{item, user[item] }}</p>% endfor %`, para obtener los datos del objeto user.

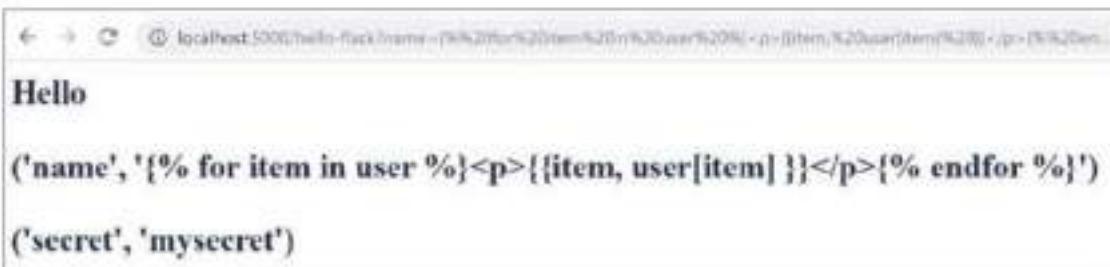


Figura 6.16 Accediendo al servidor de Flask a través del EndPoint "hello-flask".

También podríamos obtener la configuración del servidor de Flask pasando como parámetro `http://127.0.0.1:5000/hello-flask?name={{config}}`

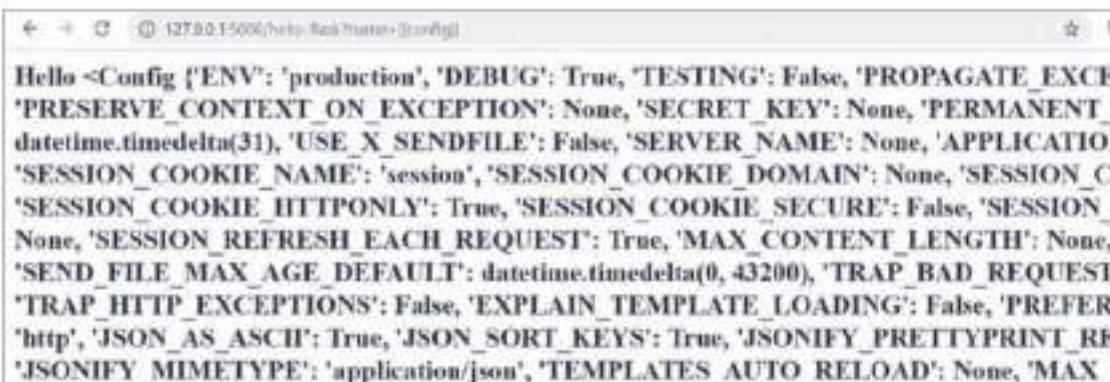


Figura 6.17 Accediendo al servidor de Flask para obtener la configuración.

Para solucionar este problema, bastaría con cambiar la forma en que generamos el template con la información del usuario; en este caso, usamos la forma correcta de visualizar un dato dentro de la plantilla `{{user.name}}`.

El siguiente código lo podemos encontrar en el repositorio de GitHub en la ruta: [https://github.com/jmortega/testing\\_python\\_security/tree/master/flask](https://github.com/jmortega/testing_python_security/tree/master/flask)

```
from flask import Flask
from flask import request, render_template_string, render_template
app = Flask(__name__)
@app.route('/hello-flask')
def hello_flask():
    user = {'name': "user", 'secret': 'mysecret'}
    if request.args.get('name'):
        user['name'] = request.args.get('name')
    template = '<h2>Hello {{user.name}}</h2>'
    return render_template_string(template, user=user)
if __name__ == "__main__":
    app.run(debug=True)
```

Ahora, si ejecutamos nuestro servidor de Flask e intentamos acceder a la url <http://127.0.0.1:5000/hello-flask?name=user%20{{user.secret}}>, ya no muestra la información secreta del objeto user.



Figura 6.18 Accediendo al servidor de Flask a través delEndPoint "hello-flask".

Ahora, si lanzamos la misma url que hemos visto anteriormente, [http://localhost:5000/hello-flask?name=%20for%20item%20in%20user%20%}3Cp%3E{{item%20user\[item\]}%3C/p%3E%20endfor%20%}](http://localhost:5000/hello-flask?name=%20for%20item%20in%20user%20%}3Cp%3E{{item%20user[item]}%3C/p%3E%20endfor%20%}), tampoco podemos acceder al valor de secret.

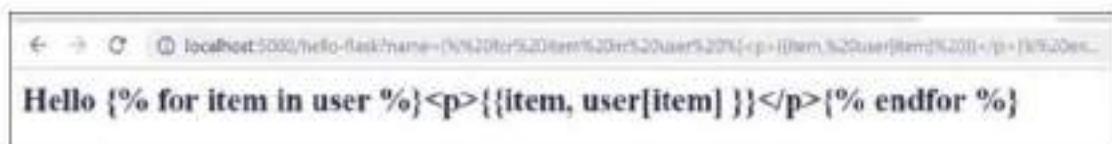


Figura 6.19 Accediendo al servidor de Flask a través delEndPoint "hello-flask".

Lo mismo con la configuración <http://127.0.0.1:5000/hello-flask?name={{config}}>



Figura 6.20 Accediendo al servidor de Flask para obtener la configuración.

De esta forma, aseguramos el acceso a la información de configuración del servidor y de la aplicación.

## 6.9 Servicios para comprobar la seguridad de proyectos Python

### 6.9.1 Pyup

Se trata de un servicio <https://pyup.io> que permite analizar las dependencias y librerías que está usando nuestro proyecto. Para ello, es capaz de analizar los repositorios tanto públicos como privados de GitHub e internamente. Lo que hace es analizar el fichero **requirements.txt** dentro del proyecto y ver si, para cada librería que utiliza, está empleando la última versión o, por el contrario, se necesita actualizarla.



Figura 6.21 Configuración de actualización de las dependencias de nuestro proyecto.

### requirements.txt

altgraph	=0.16.1	0.16.1	<a href="#">upgrade</a>	<input checked="" type="checkbox"/> Python3	<input checked="" type="checkbox"/> Permissive	
amqp	=0.4.1	2.6.2	<a href="#">upgrade</a>	<input checked="" type="checkbox"/> Python3	<input checked="" type="checkbox"/> Permissive	<a href="#">MIT</a>
androguard	=0.3.5	3.3.5	<a href="#">upgrade</a>	<input checked="" type="checkbox"/> Python3	<input checked="" type="checkbox"/> Permissive	
appdirs	=1.4.3	1.4.3	<a href="#">upgrade</a>	<input checked="" type="checkbox"/> Python3	<input checked="" type="checkbox"/> Permissive	
asn1crypto	=0.24.0	0.24.0	<a href="#">upgrade</a>	<input checked="" type="checkbox"/> Python3	<input checked="" type="checkbox"/> Permissive	
async-timeout	=3.0.1	3.0.1	<a href="#">upgrade</a>	<input checked="" type="checkbox"/> Python3	<input checked="" type="checkbox"/> Permissive	
asyncio	=3.4.3	3.4.3	<a href="#">upgrade</a>	<input checked="" type="checkbox"/> Python3	<input checked="" type="checkbox"/> Permissive	
attrs	=18.1.0	19.1.0	<a href="#">upgrade</a>	<input checked="" type="checkbox"/> Python3	<input checked="" type="checkbox"/> Permissive	<a href="#">MIT</a>
Automat	=0.7.0	0.7.0	<a href="#">upgrade</a>	<input checked="" type="checkbox"/> Python3	<input checked="" type="checkbox"/> Permissive	

Figura 6.22 Dependencias de nuestro proyecto a partir del fichero requirements.txt.

Además, si alguna librería que esté usando en su proyecto tiene alguna issue, le proporciona el enlace para ver la información en GitHub.

1 urllib3 vulnerability found in requirements.txt 13 minutes ago

**Remediation**

Upgrade urllib3 to version 1.24.2 or later. For example:

```
urllib3>~1.24.2
```

*Always verify the validity and compatibility of suggestions with your codebase.*

**Details**

CVE-2019-11324 [\[2\]](#) high severity

Vulnerable versions: < 1.24.2  
Patched version: 1.24.2

The urllib3 library before 1.24.2 for Python mishandles certain cases where the desired set of CA certificates is different from the OS store of CA certificates, which results in SSL connections succeeding in situations where a verification failure is the correct outcome. This is related to use of the ssl\_context, ca\_certs, or ca\_certs\_dir argument.

Figura 6.23 Vulnerabilidad detectada en la librería urllib3, que se soluciona actualizando la versión.

## 6.9.2 LGTM en proyectos Python

LGTM <https://lgtm.com/help/lgtm/about-lgtm> es una herramienta que nos permite analizar los repositorios de GitHub para la ejecución del análisis estático de código y el análisis de vulnerabilidades. A continuación vamos a analizar algunas de las reglas de seguridad relacionadas con Python que LGTM tiene definidas en su base de datos:

<https://lgtm.com/search?q=language%3Apython%20security&t=rules>

The screenshot shows the LGTM interface with a search bar at the top containing 'language:python security'. On the left is a sidebar with links: 'Projects (129)', 'People (0)', 'Queries (14)', and 'Help (0)'. The main area displays a list of security rules under the heading 'Queries (14)'. Each rule has a title, a detailed description, and a 'View' link.

- Incomplete URL substring sanitization** ([py/incomplete-url-substring-sanitization](#))  
Security checks on the substrings of unparsed URLs are often vulnerable to bypassing.
- Use of a broken or weak cryptography algorithm** ([py/weak-cryptography-algorithm](#))  
Using broken or weak cryptographic algorithms can compromise security.
- Incomplete regular expression attack hostnames** ([py/incomplete-regex-hostnames](#))  
Matching a URL on hosts containing a regular expression that contains no constraints to account for the specific regular expression.
- Request without certificate validation** ([py/request-without-certificate-validation](#))  
HTTP requests without certificate validation can allow man-in-the-middle attacks.
- Request from browser** ([py/request-from-browser](#))

Figura 6.24 Reglas de seguridad para proyectos Python.

## 6.9.3 Sanitización de las url

La sanitización de las url que pueden no ser confiables es una técnica importante para prevenir ataques como falsificaciones de solicitudes y redirecciones maliciosas. Por lo general, esto se hace comprobando que el dominio de una url se halle en un conjunto de dominios permitidos. Podemos encontrar un ejemplo de este caso en la url <https://lgtm.com/rules/1507386916281/>

This screenshot shows the details for the 'Incomplete URL substring sanitization' rule. It includes the rule's title, description, and various metadata such as query pack, ID, language, severity, and tags.

**Security checks on the substrings of an unparsed URL are often vulnerable to bypassing.**

**Query pack:** core.lgtm/python-queries  
**Query ID:** py/incomplete-url-substring-sanitization  
**Language:** Python  
**Severity:** warning  
**Tags:** correctness, security, external/cwe/cwe-20  
**Displayed by default?** Yes. Alerts for this query are visible by default, but can be hidden on a per-project basis.

Figura 6.25 Regla de seguridad para sanitización de las url.

#### 6.9.4 Uso de un algoritmo criptográfico roto o débil

El uso de algoritmos criptográficos rotos o débiles puede hacer que los datos resulten vulnerables y serán descifrados por un atacante. Se sabe que algunos algoritmos criptográficos, como DES o SHA-1 para la generación de hashes y que son proporcionados por las librerías de criptografía como `pycrypto` <https://pypi.org/project/pycrypto>, son débiles y se encuentran criptográficamente rotos.

La principal recomendación reside en utilizar como algoritmos criptográficos, al menos, AES-128 o RSA-2048, para el cifrado, y SHA-2 o SHA-3, para la generación de hashes. Se recomienda utilizar el paquete de criptografía <https://cryptography.io/en/latest> utilizando algoritmos de clave asimétrica como RSA, el algoritmo de firma digital (DSA) o criptografía de curva elíptica (ECC). Con la tecnología actual, los tamaños de clave de 2048 bits para RSA y DSA, o 224 bits para ECC se consideran irrompibles. Podemos encontrar un ejemplo de este caso en la url <https://lgtm.com/rules/1506299418482>

Using broken or weak cryptographic algorithms can compromise security.

**Query pack:** com\_lgtm/python-queries

**Query ID:** py/weak-cryptographic-algorithm

**Language:** Python

**Severity:** warning

**Tags:** security, external/cws/cwe-327

**Displayed by default?** Yes. Alerts for this query are visible by default, but can be hidden on a per-project basis.

Figura 6.26 Regla de seguridad para el uso de criptografía.

#### 6.9.5 Peticiones con requests sin validación de certificado

Las funciones en el módulo de `requests` <https://3.python-requests.org> proporcionan comprobación de certificados al conectarnos a un sitio seguro de forma predeterminada y, solo cuando se desactiva explícitamente su uso mediante `verify = False`, no se realiza tal comprobación. Realizar una petición https con la librería `requests` sin una correcta validación del certificado puede dar lugar a ataques *man in the middle*. Por esta razón, no resulta aconsejable deshabilitar la verificación que proporciona TLS. Podemos encontrar un ejemplo de este caso en la url <https://lgtm.com/rules/1506755127042>.

Making a request without certificate validation can allow man-in-the-middle attacks.

**Query pack:** core.lib/python-queries

**Query ID:** py/request-without-cert-validation

**Language:** Python

**Severity:** error

**Tags:** security, external, security-2015

**Displayed by default?** No. Alerts for this query are hidden by default, but can be made visible on a per-project basis. Learn how.

Figura 6.27 Regla de seguridad para la validación del certificado.

#### 6.9.6 Uso de la versión insegura SSL/TLS

El uso de una versión SSL/TLS insegura puede dejar la conexión vulnerable a los ataques; por ejemplo, resulta conocido que todas las versiones de SSL y TLS 1.0 son vulnerables a determinados ataques de suplantación. En este punto, se recomienda utilizar TLS 1.1 o superior.

El siguiente fragmento de código muestra diversas maneras de configurar una conexión usando SSL o TLS. Se ha de tener en cuenta que el método `ssl.wrap_socket()` está deprecado en las últimas versiones de Python y ha quedado en desuso a partir de Python 3.7. Una alternativa consiste en usar `ssl.SSLContext`, que se admite a partir de las versiones Python 2.7.9 y 3.2 <https://docs.python.org/3/library/ssl.html#ssl.SSLContext>

También podríamos utilizar el módulo <https://pyopenssl.org/en/stable/api/ssl.html>, usando una versión al menos TLS1.1:

```
import ssl
import socket
# método deprecado
ssl.wrap_socket(socket.socket(), ssl_version=ssl.PROTOCOL_SSLv2)
# SSLContext
context = ssl.SSLContext(ssl_version=ssl.PROTOCOL_SSLv3)
# pyOpenSSL
from pyOpenSSL import SSL
context = SSL.Context(SSL.TLSv1_METHOD)
```

#### 6.9.7 Deserialización de entrada no confiable

Deserializar datos no confiables utilizando módulos de deserialización como pickle, que permite la construcción de objetos serializables arbitrarios, resulta fácilmente explotable y, en muchos casos, permite que un atacante ejecute código arbitrario. La deserialización automática de los campos de un objeto

puede hacer que un atacante cree una combinación anidada de objetos en los que el código de inicialización ejecutado puede tener efectos imprevistos, como la ejecución de código arbitrario. Entre las principales librerías de serialización de objetos en Python, podemos destacar:

- **pickle**: <https://docs.python.org/3/library/pickle.html>
- **marshal**: <https://docs.python.org/3/library/marshal.html#module-marshal>
- **yaml**: <https://pyyaml.org/wiki/PyYAMLDocumentation>

Podemos encontrar un ejemplo de este caso en la url <https://lgtm.com/rules/1506218107765>



Figura 6.28 Regla de seguridad para la deserialización de los datos.

### 6.9.8 Vulnerabilidades de XSS

Escribir directamente la entrada del usuario en un sitio web sin validar de manera correcta la entrada permite una vulnerabilidad de XSS. En este punto, la principal recomendación consiste en escapar la entrada antes de escribir la entrada del usuario en la página.

La librería estándar de Python proporciona una serie de funciones de escape, entre las que podemos destacar la que encontramos en la librería estándar **html.escape()**: <https://docs.python.org/3/library/html.html#html.escape>. La mayoría de los frameworks web, como Django o Flask, disponen también de sus propias funciones de escape; por ejemplo, **flask.escape()**. Podemos encontrar un ejemplo de este caso en la url <https://lgtm.com/rules/1506064236628>.

Writing user input directly to a web page allows for a cross-site scripting vulnerability.

**Query pack:** core\_lgtm/python-queries

**Query ID:** py/reflection-xss

**Language:** Python

**Severity:** Error

**Tags:** security, external/cwe/cve-079, external/cwe/nwe-116

**Displayed by default?** Yes. Alerts for this query are visible by default, but can be hidden on a per-project basis. Learn how.

Figura 6.29 Regla de seguridad para vulnerabilidades de XSS.

### 6.9.9 Exposición de información a través de una excepción

Los desarrolladores de software, en ocasiones, dejan trazas relacionadas con los mensajes de error que pueden darse en una aplicación como una ayuda de depuración. En particular, los stacktraces pueden ofrecer al desarrollador más información sobre la secuencia de eventos que origina un determinado error.

El problema con esta información es que, en ocasiones, puede exponer los detalles de la implementación, que resultan útiles para un atacante con el fin de encontrar otras vulnerabilidades *a posteriori*; por ejemplo, la secuencia de nombres de clase puede revelar la estructura de la aplicación, así como obtener información sobre cualquier componente interno. Además, los mensajes de error pueden incluir información, como los nombres de los archivos del lado del servidor y el código SQL en el que se basa la aplicación, lo que permite a un atacante ejecutar un ataque de inyección.

En este punto, la recomendación es enviar al usuario un mensaje de error lo más genérico posible para evitar mostrar toda la traza de error. Podemos encontrar un ejemplo de este caso en la url <https://lgtm.com/rules/1506064236628>.

Leaking information about an exception, such as messages and stack traces, to an external user can expose implementation details that are useful to an attacker for developing a subsequent exploit.

**Query pack:** core\_lgtm/python-queries

**Query ID:** py/stack-trace-exposure

**Language:** Python

**Severity:** Error

**Tags:** security, external/cwe/nwe-209, external/cwe/nwe-097

**Displayed by default?** Yes. Alerts for this query are visible by default, but can be hidden on a per-project basis. Learn how.

Figura 6.30 Regla de seguridad para exposición de la información.

### 6.9.10 Conexión con hosts remotos mediante SSH utilizando Paramiko

En el protocolo Secure Shell (SSH), las claves de host se utilizan para verificar la identidad de los hosts remotos. Aceptar claves de host desconocidas puede originar ataques *man in the middle*.

En particular, si estamos utilizando **Paramiko** (<http://www.paramiko.org>) como librería para establecer una conexión de SSH con un servidor remoto, resulta importante establecer la política **RejectPolicy**, ya que permite lanzar una excepción cuando encuentra una clave de host desconocida.

En el siguiente ejemplo se muestran dos formas de abrir una conexión de SSH a un servidor dominio.com. En la primera función donde se establece la política en **AutoAddPolicy**, la conexión puede verse comprometida si da un error en la verificación de la clave del host. En la segunda función se establece la política **RejectPolicy** y se lanzará una excepción si falla la verificación de la clave de host:

```
from paramiko.client import SSHClient, AutoAddPolicy, RejectPolicy

def unsafe_connect():
    client = SSHClient()
    client.set_missing_host_key_policy(AutoAddPolicy)
    client.connect("dominio.com")
    # ... interaction with server
    client.close()

def safe_connect():
    client = SSHClient()
    client.set_missing_host_key_policy(RejectPolicy)
    client.connect("dominio.com")
    # ... interaction with server
    client.close()
```

## 6.10 Análisis estático de código Python

En general, se puede decir que el código del núcleo de Python resulta seguro, pero los módulos de terceros y la forma en que ha desarrollado una aplicación puede no serlo. Para ello, podemos usar diferentes herramientas para encontrar riesgos de seguridad en una aplicación desarrollada en Python.

### 6.10.1 Python Taint

Una herramienta de análisis estático de código abierto <https://github.com/python-security/pyt> detecta bugs de seguridad relacionados con la inyección de comandos, cross-site scripting, inyección de SQL y ataques transversales de directorios en aplicaciones web de Python.

```
usage: python -m pyt [-h] [-v] [-a ADAPTOR] [-pr PROJECT_ROOT]
                     [-b BASELINE_JSON_FILE] [-t TRIGGER_WORD_FILE]
                     [-m BLACKBOX_MAPPING_FILE] [-l] [-o OUTPUT_FILE]
                     [--ignore-nosec] [-r] [-x EXCLUDED_PATHS]
                     [--dont-prepend-root] [--no-local-imports] [-u] [-j] [-s]
                     targets [targets ...]

required arguments:
  targets           source file(s) or directory(s) to be scanned

optional arguments:
  -v, --verbose    Increase Logging verbosity. Can be repeated e.g. -vvv
  -a ADAPTOR, --adaptor ADAPTOR
                    Choose a web framework adaptor: Flask(Default),
                    Django, Every or Pylons
  -pr PROJECT_ROOT, --project-root PROJECT_ROOT
                    Add project root, only important when the entry file
                    is not at the root of the project.
  -b BASELINE_JSON_FILE, --baseline BASELINE_JSON_FILE
                    Path of a baseline report to compare against (only
                    JSON-formatted files are accepted)
  -t TRIGGER_WORD_FILE, --trigger-word-file TRIGGER_WORD_FILE
                    Input file with a list of sources and sinks
  -m BLACKBOX_MAPPING_FILE, --blackbox-mapping-file BLACKBOX_MAPPING_FILE
```

Figura 6.31 Opciones de ejecución de Python Taint.

Esta herramienta ayuda a la detección de errores de seguridad en las aplicaciones web de Python y se centra en el desarrollo de aplicaciones web con Flask. La idea con esta herramienta reside en determinar cuándo una entrada de un usuario puede alcanzar un lugar en el código donde puede ser peligrosa. Una técnica para determinar esto consiste en el análisis estático, donde la fuente de un programa se analiza estáticamente.

```
pyt examples/vulnerable_code/XSS_reassign.py
1 vulnerability found:
Vulnerability 1:
File: examples/vulnerable_code/XSS_reassign.py
> User input at line 6, source "request.args.get(":
    -call_1 = ret_request.args.get('param', 'not set')
Reassigned in:
    File: examples/vulnerable_code/XSS_reassign.py
    > Line 6: param = -call_1
    File: examples/vulnerable_code/XSS_reassign.py
    > Line 8: param = param + ''
File: examples/vulnerable_code/XSS_reassign.py
> reaches line 11, sink "replace(":
    -call_4 = ret_html.replace('{{ param }}', param)
```

Figura 6.32 Detección de una vulnerabilidad de XSS.

Para detectar tales vulnerabilidades, la herramienta lo que hace es conectar información sobre las entradas del usuario y el código que va analizando para detectar cuándo una llamada u operación puede ser peligrosa. La idea es analizar el código fuente para detectar posibles casos de inyección de SQL o XSS; por ejemplo, podemos detectar una inyección de SQL cuando la entrada del usuario llega a una consulta de la base de datos o ataques transversales que se producen cuando la entrada del usuario llega a un fragmento de código que realiza el acceso al sistema de archivos.

### 6.10.2 Bandit

Bandit <https://github.com/PyCQA/bandit> es una iniciativa de la empresa **OpenStack** para encontrar riesgos de seguridad en los scripts y aplicaciones desarrollados en Python. El uso de Bandit puede personalizarse, de forma que sería posible realizar pruebas para encontrar un tipo específico de vulnerabilidad; por ejemplo, podría detectar un fallo relacionado con la ejecución e inyección de comandos a través de la shell si ejecuta el comando sobre nuestro directorio de trabajo:

```
$ bandit work_directory/*.py -p ShellInjection
```

Bandit usa el **módulo ast** de la biblioteca estándar de Python para analizar el código de Python. El módulo ast solo puede analizar el código de Python que es válido en la versión del intérprete desde el cual se importa. De esta forma, si se intenta usar el módulo ast desde un intérprete Python 3.5, el código debería estar escrito para 3.5 para poder analizar el código.

```
usage: bandit [-h] [-r] [-a {file,vuln}] [-n CONTEXT_LINES] [-c CONFIG_FILE]
              [-p PROFILE] [-t TESTS] [-s SKIPS] [-i] [-i]
              [-F {csv,custom,html,json,screen,txt,xml,yaml}]
              [--msg-template MSG_TEMPLATE] [-o {OUTPUT_FILE}] [-v] [-d]
              [--ignore-NOSC] [-x EXCLUDED_PATHS] [-b BASELINE]
              [-ini INI_PATH] [---version]
              [targets ...]

bandit - a Python source code security analyzer

positional arguments:
  targets            source file(s) or directory(s) to be tested

optional arguments:
  -h, --help          show this help message and exit
  -r, --recursive    find and process files in subdirectories
  -a {file,vuln}, --aggregate {file,vuln}
                     aggregate output by vulnerability (default) or by
                     filename
  -n CONTEXT_LINES, --number CONTEXT_LINES
                     maximum number of code lines to output for each issue
  -c CONFIG_FILE, --configfile CONFIG_FILE
                     optional config file to use for selecting plugins and
                     overriding defaults
  -p PROFILE, --profile PROFILE
                     profile to use (defaults to executing all tests)
  -t TESTS, --tests TESTS
                     comma-separated list of test ids to run
```

Figura 6.33 Opciones de ejecución de Bandit.

Bandit soporta muchos tipos de pruebas diferentes para detectar diversos problemas de seguridad en el código de Python. Estas pruebas se crean como complementos y se pueden desarrollar otras nuevas para ampliar la funcionalidad ofrecida por defecto.

## Plugin ID Groupings

ID	Description
B1xx	misc tests
B2xx	application/framework misconfiguration
B3xx	blacklists (calls)
B4xx	blacklists (imports)
B5xx	cryptography
B6xx	injection
B7xx	XSS

Figura 6.34 Plugins disponibles en Bandit.

Por ejemplo, el plugin **B602: subprocess\_popen\_with\_shell\_equals\_true** busca el uso de la llamada **subprocess.Popen**, que emplea como argumento en la llamada **shell=True**. Este tipo de llamada no resulta recomendable, ya que se muestra vulnerable a varios ataques de inyección de shell.

```
shell_injection:  
    # Start a process using the subprocess module, or one of its  
    # wrappers.  
    subprocess: [subprocess.Popen, subprocess.call,  
                subprocess.check_call, subprocess.check_output  
                execute_with_timeout]
```

Figura 6.35 Plugin que detecta la llamada a la función subprocess.popen.

El plugin **shell\_injection** explora aquellos métodos y llamadas que se encuentran en la sección de subprocess y tienen activado el parámetro **shell=True**.

```
more info: https://bandit.readthedocs.io/en/latest/plugins/b602_subprocess_popen_with_shell_equals_true.html
11:     pop('/bin/gcc --version', shell=True)
12:     Popen('/bin/gcc --version', shell=True)

--vs-- issue: [B604:any_other_function_with_shell_equals_true] Function call with shell=True parameter identified, possible security issue.
severity: Medium confidence: Low
location: ./subprocess_shell.py:12
more info: https://bandit.readthedocs.io/en/latest/plugins/b604_any_other_function_with_shell_equals_true.html
11:     pop('/bin/gcc --version', shell=True)
12:     Popen('/bin/gcc --version', shell=True)
13:
```

Figura 6.36 Plugin que detecta la llamada a la función subprocess.Popen.

Bandit dispone de un complemento que analiza los métodos que podemos encontrar en la sección de shell.

```
shell_injection:
    shell:
        - os.system
        - os.popen
        - os.popen2
        - os.popen3
        - os.popen4
        - popen2.popen2
        - popen2.popen3
        - popen2.popen4
        - popen2.Popen3
        - popen2.Popen4
        - commands.getoutput
        - commands.getstatusoutput
```

Figura 6.37 Plugin que detecta una posible inyección de shell.

Un ataque de inyección de SQL consiste en la inserción o “inyección” de una consulta de SQL a través de los datos de entrada de una aplicación. El plugin **B608: Test for SQL Injection** busca cadenas que se parezcan a las sentencias de SQL que estén involucradas en alguna en un ataque de inyección de SQL; por ejemplo:

```
SELECT %s FROM derp;" % var
"SELECT thing FROM " + tab
"SELECT " + val + " FROM " + tab + ...
"SELECT {} FROM derp;".format(var)
```

```

>> Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector through string-based query construction.
SEVERITY: medium confidence: low
LOCATION: \sql_statements.py:4
More Info: https://bandit.readthedocs.io/en/latest/plugins/b608_hardcoded_sql_expressions.html
# bad
query = "SELECT * FROM foo WHERE id = '%s'" % identifier
query = "INSERT INTO foo VALUES ('a', 'b', '%s')" % value
query = "DELETE FROM foo WHERE id = '%s'" % identifier

>> Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector through string-based query construction.
SEVERITY: medium confidence: low
LOCATION: \sql_statements.py:5
More Info: https://bandit.readthedocs.io/en/latest/plugins/b608_hardcoded_sql_expressions.html
query = "SELECT * FROM foo WHERE id = '%s'" % identifier
query = "INSERT INTO foo VALUES ('a', 'b', '%s')" % value
query = "DELETE FROM foo WHERE id = '%s'" % identifier

>> Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector through string-based query construction.
SEVERITY: medium confidence: low
LOCATION: \sql_statements.py:6
More Info: https://bandit.readthedocs.io/en/latest/plugins/b608_hardcoded_sql_expressions.html
query = "INSERT INTO foo VALUES ('a', 'b', '%s')" % value
query = "DELETE FROM foo WHERE id = '%s'" % identifier
query = "UPDATE foo SET value = %d WHERE id = '%s'" % identifier

```

Figura 6.38 Plugin que detecta una posible inyección de SQL.

Además, dispone de una lista de comprobaciones que realiza para detectar aquellas funciones que no se estén usando de forma segura.

The following tests were discovered and loaded:

```

B101    assert_used
B102    exec_used
B103    set_bad_file_permissions
B104    hardcoded_bind_all_interfaces
B105    hardcoded_password_string
B106    hardcoded_password_funcarg
B107    hardcoded_password_default
B108    hardcoded_tmp_directory
B110    try_except_pass
B112    try_except_continue
B201    flask_debug_true
B301    pickle
B302    marshal
B303    md5
B304    ciphers
B305    cipher_modes
B306    mktemp_q
B307    eval
B308    mark_safe
B309    httpsconnection
B310    urllib_urlopen

```

Figura 6.39 Comprobaciones que realiza en el uso de funciones inseguras.

Por ejemplo, si encuentra que se está usando el módulo pickle, las funciones que comprobaría serían las enumeradas en la figura 6.40.

ID	Name	Calls	Severity
B301	pickle	<ul style="list-style-type: none"><li>• pickle.loads</li><li>• pickle.load</li><li>• pickle.Unpickler</li><li>• cPickle.loads</li><li>• cPickle.load</li><li>• cPickle.Unpickler</li><li>• dill.loads</li><li>• dill.load</li><li>• dill.Unpickler</li></ul>	Medium

Figura 6.40 Comprobaciones que realiza relacionadas con el módulo pickle.

#### 6.10.3 Hawkeye

Hawkeye Scanner CLI <https://github.com/hawkeyesec/scanner-cli> es una herramienta que permite evaluar la seguridad de los proyectos y detectar vulnerabilidades y librerías desactualizadas. La herramienta asume que la estructura de su proyecto es tal que se sigue el estándar para cada tipo de proyecto:

- Los proyectos NodeJS tienen un **package.json** en el nivel raíz del proyecto.
- Los proyectos de Ruby tendrán un **Gemfile** en el nivel raíz del proyecto.
- Los proyectos de Python tendrán un **requirements.txt** en el nivel raíz del proyecto.
- Los proyectos PHP tendrán un **composer.lock** en el nivel raíz del proyecto.
- Los proyectos Java tendrán una carpeta de compilación (gradle) o destino (maven) e incluirán archivos .java y .jar.

Podemos instalar y ejecutar Hawkeye en un proyecto a través del gestor de paquetes npm:

```
npm install --save-dev @hawkeyesec/scanner-cli  
npx hawkeye scan
```

Con el comando **npx hawkeye modules**, podemos ver los módulos que incorpora para cada tipo de proyecto. En el caso de Python, los módulos disponibles son los que siguen:

- **python-bandit**: Busca problemas de seguridad comunes en el código Python con Bandit.
- **python-piprot**: Analiza las dependencias de Python en busca de paquetes obsoletos con Piprot.
- **python-safety**: Comprueba las dependencias de Python para detectar vulnerabilidades de seguridad conocidas.

```
[info] Enabled: python-bandit  
[info]          Scans for common security issues in Python code with bandit.  
[info] Enabled: python-piprot  
[info]          Scans python dependencies for out of date packages  
[info] Enabled: python-safety  
[info]          Checks python dependencies for known security vulnerabilities with the safety tool.
```

Figura 6.41 Módulos del proyecto para Python.

```
Usage: hawkeye-scan [options]  
  
Options:  
-a, --all                         Scan all files, regardless if a git repo is found. Defaults to track  
files in git repositories.  
-t, --target [/path/to/project]      The location to scan. Defaults to $PWD.  
-f, --fail-on [low|medium|high|critical] Set the level at which hawkeye returns non-zero status codes. Default  
to low.  
-m, --module [module name]          Run specific module. Defaults to all applicable modules.  
-e, --exclude [pattern]             Specify one or more exclusion patterns (eg. test/*). Can be specified  
multiple times.  
-j, --json [/path/to/file.json]      Write findings to file.  
-s, --sum [https://metacritic-music-connector] Write findings to sumologic.  
-H, --http [https://your-ulta.com/api/results] Write findings to a given url.  
--show-code                        Shows the code the module uses for reporting, useful for ignoring cer-  
tain false positives.  
-g, --staged                        Scan only git-staged files.  
-h, --help                          output usage information
```

Figura 6.42 Opciones de ejecución de hawkeye-scan.

Podemos realizar un escaneo con el módulo **python-bandit**, con el fin de detectar vulnerabilidades del nivel high:

```
npx hawkeye scan -a -f high --target . --module python-bandit
```

```
[info] Checking python-bandit for applicability
[node:20984] ExperimentalWarning: The http2 module is an experimental API.
[info] Running module python-bandit
[info] Scan complete, 15 issues found
[info] Writing to: writer-console
module      level offender                                description
mitigation

-----
python-bandit: high  .\crypto\crypto_example.py lines 1,2      blacklist B413
    The pyCrypto library and its module DTS are no longer actively maintained and have been deprecated. Consider using pycryptodome library. Review the file and fix the issue.
python-bandit: high  .\crypto\crypto_example.py lines 3      blacklist B384
    use of insecure cipher Crypto.Cipher.DES.new. Replace with a known secure cipher such as AES. Review the file and fix the issue.
python-bandit: high  .\crypto\crypto_example.py lines 9      blacklist B384
    Use of insecure cipher Crypto.Cipher.Blowfish.new. Replace with a known secure cipher such as AES. Review the file and fix the issue.
```

Figura 6.43 Ejecución de hawkeye-scan con el módulo python-bandit.

#### 6.10.4 DLint

El objetivo del análisis estático estriba en buscar en el código e identificar problemas potenciales. Esta es una forma efectiva de encontrar problemas en el código a un bajo coste, en comparación con el análisis dinámico, que implica la ejecución del código. Sin embargo, ejecutar un análisis estático efectivo requiere superar algunos desafíos.

Primero, tenemos que averiguar qué buscar; por ejemplo, casi nunca desearía pasar una **entrada no confiable** a una llamada a `os.system`. DLint dispone de reglas específicas para asegurarnos de que los diferentes módulos se utilizan de forma segura.

Después de que sepamos qué buscar, necesitamos una manera de hacerlo. La forma más simple de análisis estático sería buscar a través del código línea por línea, para cadenas específicas. Sin embargo, podemos llevar esto un paso más allá y analizar el árbol de sintaxis abstracta, o AST, del código para realizar consultas más informadas y complejas.

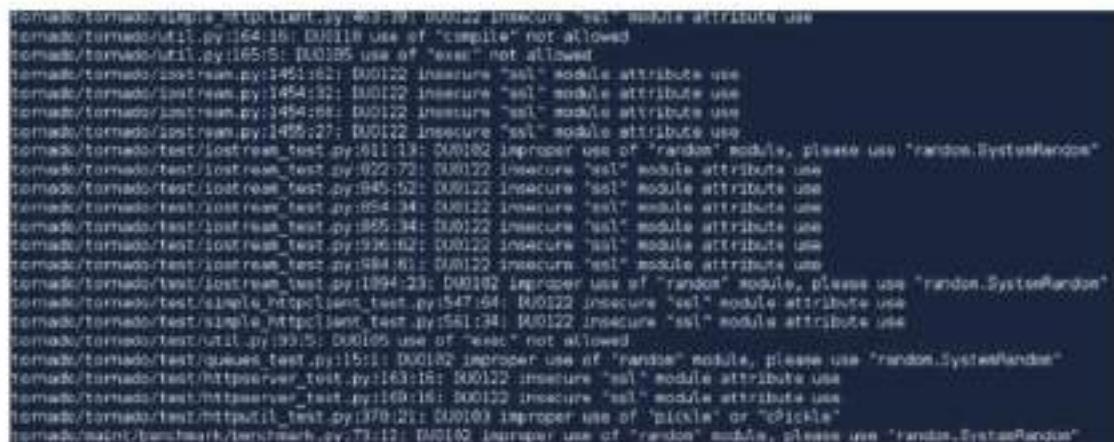
Una vez que tengamos la capacidad de realizar análisis, debemos determinar cuándo ejecutar las comprobaciones. Creemos en proporcionar herramientas que puedan ejecutarse tanto a nivel local como en el código que se está desarrollando para proporcionar una respuesta rápida, así como en nuestra línea de desarrollo continuo, antes de que el código se fusione en nuestro código base.

Dlint <https://github.com/duo-labs/dlint> comprende un conjunto de reglas <https://github.com/duo-labs/dlint/tree/master/dlint/linters> que definen lo que que-

remos buscar y un indicador capaz de evaluar esas reglas sobre nuestro código base. Para esta versión inicial, Dlint contiene un conjunto de reglas que verifican las mejores prácticas comunes cuando se trata de escribir Python seguro.

Para evaluar tales reglas sobre nuestro código base, Dlint aprovecha Flake8 <http://flake8.pycqa.org/en/latest>

Este enfoque le permite a Flake8 hacer el trabajo pesado de analizar el AST de Python, lo que nos posibilita concentrarnos en escribir un conjunto de reglas sólido y proporcionar excelentes recomendaciones.



```
tornado/tornado/base.py:101:16: D0010 use of "compile" not allowed
tornado/tornado/util.py:164:36: D0010 use of "compile" not allowed
tornado/tornado/test/test.py:165:5: D0010 use of "exec" not allowed
tornado/tornado/test/testream.py:1451:62: D0012 insecure "set" module attribute use
tornado/tornado/test/testream.py:1454:32: D0012 insecure "set" module attribute use
tornado/tornado/test/testream.py:1454:66: D0012 insecure "set" module attribute use
tornado/tornado/test/testream.py:1455:27: D0012 insecure "set" module attribute use
tornado/tornado/test/testream_test.py:611:18: D00102 improper use of "random" module, please use "random.SystemRandom"
tornado/tornado/test/testream_test.py:622:72: D00122 insecure "set" module attribute use
tornado/tornado/test/testream_test.py:645:52: D00122 insecure "set" module attribute use
tornado/tornado/test/testream_test.py:654:34: D00122 insecure "set" module attribute use
tornado/tornado/test/testream_test.py:655:34: D00122 insecure "set" module attribute use
tornado/tornado/test/testream_test.py:656:32: D00122 insecure "set" module attribute use
tornado/tornado/test/testream_test.py:656:62: D00122 insecure "set" module attribute use
tornado/tornado/test/testream_test.py:657:62: D00122 insecure "set" module attribute use
tornado/tornado/test/testream_test.py:658:62: D00122 insecure "set" module attribute use
tornado/tornado/test/testream_test.py:659:62: D00122 insecure "set" module attribute use
tornado/tornado/test/testream_test.py:660:62: D00122 insecure "set" module attribute use
tornado/tornado/test/testutil.py:931:5: D00105 use of "exec" not allowed
tornado/tornado/test/testutil.py:115:11: D00102 improper use of "random" module, please use "random.SystemRandom"
tornado/tornado/test/testutil.py:163:16: D00122 insecure "set" module attribute use
tornado/tornado/test/testutil.py:169:16: D00122 insecure "set" module attribute use
tornado/tornado/test/testutil_test.py:370:21: D00103 improper use of "pickle" or "cPickle"
tornado/supervisor/benchmark/benchmark.py:73:12: D00103 improper use of "random" module, please use "random.SystemRandom"
```

Figura 6.44 Ejecución de Dlint sobre un proyecto que usa el framework tornado.

Finalmente, podríamos ejecutar Dlint como parte de un proceso de desarrollo y podríamos hacerlo con herramientas de entrega e integración continua, como Gitlab y Travis CI.

## 6.11 Gestión de dependencias

La instalación de paquetes de terceros, ya sea a través de un entorno virtual como virtualenv <https://virtualenv.pypa.io/en/latest> o de forma global en su sistema operativo mediante el comando **pip install**, lo expone a agujeros de seguridad en esos paquetes.

Otra situación que podríamos considerar son las dependencias a nivel de módulos y paquetes que incluimos en nuestros proyectos. Podrían contener vulnerabilidades y también podrían anular el comportamiento predeterminado en Python a través del sistema de importación.

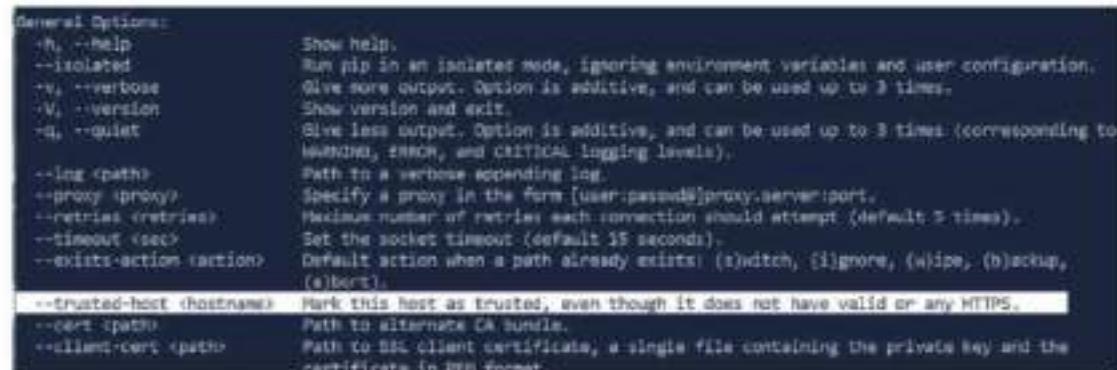
### 6.11.1 Instalación de dependencias

La instalación de un módulo de Python y sus dependencias se realiza a través del comando **pip install**. Si el protocolo HTTPS está bloqueado en determinadas redes, entonces la instalación puede bloquearse y obtener el siguiente mensaje:

"InsecurePlatformWarning: A true SSLContext object is not available. This prevents urllib3 from configuring SSL appropriately and may cause certain SSL connections to fail."

Para evitar esto, puede ejecutar el siguiente comando para añadir el repositorio pypi.python.org como fuente de confianza al instalar un determinado paquete:

```
$ pip install --trusted-host pypi.python.org <nombre_paquete>
```



General Options:

- h, --help Show help.
- isolated Run pip in an isolated mode, ignoring environment variables and user configuration.
- verbose Give more output. Option is additive, and can be used up to 3 times.
- V, --version Show version and exit.
- q, --quiet Give less output. Option is additive, and can be used up to 3 times (corresponding to WARNING, ERROR, and CRITICAL logging levels).
- log <path> Path to a verbose appending log.
- proxy <proxy> Specify a proxy in the form [user:password]@proxy.server:port.
- retries <retries> Maximum number of retries each connection should attempt (default 5 times).
- timeout <sec> Set the socket timeout (default 35 seconds).
- exists-action <action> Default action when a path already exists: (i)gnore, (w)itch, (b)ackup, (a)bort.
- trusted-host <hostname> Mark this host as trusted, even though it does not have valid or any HTTPS.
- cert <path> Path to alternate CA bundle.
- client-cert <path> Path to SSL client certificate, a single file containing the private key and the certificate in PEM format.

Figura 6.45 Opción de instalación de paquetes de forma segura.

Respecto a la instalación de paquetes de fuentes externas o no oficiales, se debe tener en cuenta que se han detectado paquetes que se publicaron en PyPi con nombres similares a los paquetes populares, pero que, en realidad, ejecutan código malicioso.

SK-CSIRT <http://www.nbu.gov.sk/skcsirt-sa-20170909-pypi/> identificó librerías con uso malintencionado que estaban disponibles en el repositorio oficial de paquetes de Python. Un ejemplo destacado es un paquete falso urllib-1.21.1.tar.gz, basado en el paquete conocido urllib3-1.21.1.tar.gz.

### 6.11.2 Requires.io

Requires.io <https://requires.io> es un servicio que permite detectar librerías y dependencias en nuestros proyectos que no estén actualizadas y, desde el punto de vista de la seguridad, pueden suponer un riesgo para nuestra aplicación.



Figura 6.46 Portal del proyecto y acceso a los repositorios en GitHub y BitBucket.

Podemos ver, para cada paquete, cuál es la versión que estamos utilizando actualmente y la compara con la última versión disponible, de forma que podemos observar los últimos cambios realizados por cada módulo y comprobar si resulta recomendable usar la última versión en función de lo que necesitemos en nuestro proyecto.

requirements/base.txt

Package	Requirement	Latest	Status
analytics-python	==1.2.0	1.2.0	up-to-date
bleach	==3.1.0	3.1.0	up-to-date
boto3	==1.9.146	1.9.146	up-to-date
bs4	==0.0.1	0.0.1	up-to-date
celery	==4.2.1	4.3.0	vulnerable
certifi	==2019.3.9	2019.3.9	up-to-date

Figura 6.47. Detección de las versiones de nuestro proyecto para cada módulo

Si detecta una versión insegura para un paquete en concreto, mostrará el flag correspondiente.

 PyYAML	 requests	 rsa	 ==3.13	 ==2.21.0	 ==3.4.2	 5.1	 2.21.0	 4.0				Insecure
												upgrade
												obsolete

Figura 6.48 Detección de una versión insegura del módulo PyYAML.

### 6.11.3 Safety

Otra herramienta que nos puede ayudar a comprobar las dependencias de nuestro proyecto es Safety <https://pyup.io/safety>, que cuenta con la capacidad de analizar el entorno de Python instalado en su máquina y detectar las versiones de los paquetes que tengamos instaladas en nuestro entorno, para detectar librerías desactualizadas o que puedan contener algún tipo de vulnerabilidad.



Figura 6.49 Herramientas que proporciona Safety.

### 6.11.4 Paquetes maliciosos en PyPI

PyPI <https://pypi.org/> es el repositorio oficial de software de terceros para Python y contiene una gran fuente de bibliotecas y módulos de código abierto para implementar funcionalidades que pueden resultar comunes en aplicaciones Python.

Desafortunadamente, se han encontrado paquetes maliciosos cuyo comportamiento es diferente al del paquete original. Esto sucedió con los paquetes libpeshnx, libpesh y libari, que fueron publicados en noviembre de 2017. Algunos de estos paquetes maliciosos lo que hacen es bajarse un archivo de forma oculta y ejecutar un proceso en segundo plano que crea una shell interactiva sin inicio de sesión.

The screenshot shows the Python Package Index (PyPI) interface. At the top, there is a search bar with the placeholder "Search projects" and a magnifying glass icon. To the right of the search bar are links for "Help", "Donate", "Log in", and "Register". On the left, there is a user profile for "nurl2" featuring a large icon of a power button. Below the profile, it says "Joined on Nov 23, 2017". There is also a "Statistics" section with a link to view stats via Libraries.io or Google BigQuery.

**3 projects**

- libpestme**  
Last released on Nov 23, 2017  
Libarl wrapper for python
- libpesth**  
Last released on Nov 23, 2017  
Libarl wrapper for python
- libarl**  
Last released on Nov 23, 2017  
Libarl wrapper for python

Figura 6.50 Paquetes maliciosos que alguna vez podemos encontrar en PyPI.

Evidentemente, estos paquetes se han eliminado del repositorio por el equipo de seguridad de PyPI, pero es probable que nos encontremos con esos casos en un futuro.

## 6.12 Python code checkers

### 6.12.1 Pyflakes

Pyflakes <https://pypi.org/project/pyflakes> permite analizar el código de Python en busca de errores. Si lo comparamos con otras herramientas del estilo, como PyLint o Pychecker, Pyflakes analiza de forma más rápida, ya que solo examina el árbol de sintaxis de cada archivo de forma individual.

El código fuente se encuentra disponible en el repositorio de GitHub <https://github.com/PyCQA/pyflakes> se puede instalar desde el repositorio oficial de Python mediante el comando pip install y resulta compatible con todas las versiones de Python: 2.7 y 3.4 a 3.7.

### 6.12.2 PyLint

PyLint <https://www pylint.org> tiene como objetivo analizar código en Python en busca de errores o síntomas de mala calidad en el código fuente. Cabe destacar que, por defecto, la guía de estilo que trata de seguir es la descrita en PEP-8 <https://www.python.org/dev/peps/pep-0008>

Entre las **revisiones básicas** que realiza, podemos destacar:

- Presencia de cadenas de documentación (*docstring*)
- Nombres de módulos, clases, funciones, métodos, argumentos y variables
- Número de argumentos, variables locales, retornos y sentencias en funciones y métodos
- Atributos requeridos para módulos
- Valores por omisión no recomendados como argumentos
- Redefinición de funciones, métodos y clases
- Uso de declaraciones globales

En cuanto a la **revisión de variables**, es capaz de detectar los siguientes casos:

- Determinación de si una variable o import no está siendo usado
- Detección de variables indefinidas
- Redefinición de variables provenientes de módulos builtins o de ámbito externo
- Uso de una variable antes de asignación de valor

Entre las **revisiones de clases**, podemos destacar:

- Métodos sin *self* como primer argumento
- Acceso único a miembros existentes vía *self*
- Atributos no definidos en el método *\_init\_*
- Código inalcanzable

Entre las **revisiones de diseño**, podemos destacar:

- Número de métodos, atributos y variables locales
- Tamaño, complejidad de funciones y métodos

Entre las **revisiones de importaciones**, podemos destacar:

- Dependencias externas
- Imports relativos o imports de todos los métodos y variables via \* (*wildcards*)
- Uso de imports cílicos
- Uso de módulos obsoletos

**Otras revisiones:**

- Código fuente con caracteres no ASCII sin tener una declaración de encoding
- Búsqueda por similitudes o duplicación en el código fuente
- Revisión de excepciones
- Revisiones de tipo haciendo uso de inferencia de tipos

Con posterioridad a los mensajes de análisis mostrados, PyLint genera una serie de reportes, cada uno de ellos enfocándose en un aspecto particular del proyecto, como el número de mensajes por categorías y las dependencias de módulos.

## 6.13 Escáner de seguridad de aplicaciones web

### 6.13.1 WAScan

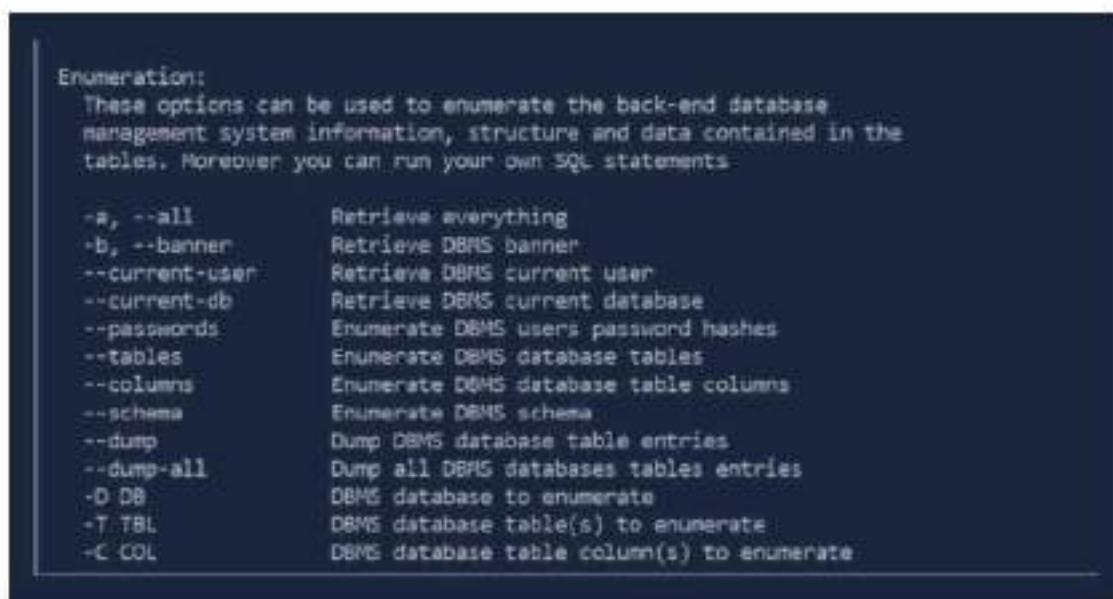
WAScan <https://github.com/m4ll0k/WAScan> es un escáner de seguridad de aplicaciones web desarrollado en Python 2.7, diseñado para encontrar configuraciones inseguras y erróneas, y puede ejecutarse en cualquier plataforma que tenga un entorno Python instalado.

Está diseñado para encontrar vulnerabilidades utilizando el método de “caja negra”, lo que significa que no analiza el código fuente de las aplicaciones web, sino que funciona como un fuzzer, escanea las páginas de la aplicación web, extrae enlaces y ataca formularios para obtener mensajes de error. Es capaz de descubrir diferentes ataques, entre los que podemos destacar la fuerza bruta, la inyección de SQL, de XSS o la Shell Injection.

### 6.13.2 SQLmap

SQLmap <http://sqlmap.org/> es una de las herramientas más conocidas escrita en Python para detectar vulnerabilidades de tipo SQL Injection. Se trata de una herramienta desarrollada en Python que permite automatizar el reconocimiento y la explotación de múltiples bases de datos, como MySQL, Oracle o PostgreSQL.

Para hacer esto, la herramienta permite solicitar los parámetros de una url, ya sea a través de una solicitud GET o POST, y detectar si, para algún parámetro del dominio para analizar, se muestra vulnerable a algún tipo de ataque de SQL Injection, debido a que los parámetros no se están validando correctamente.



```
Enumeration:
These options can be used to enumerate the back-end database
management system information, structure and data contained in the
tables. Moreover you can run your own SQL statements

-e, --all          Retrieve everything
--banner          Retrieve DBMS banner
--current-user    Retrieve DBMS current user
--current-db      Retrieve DBMS current database
--passwords       Enumerate DBMS users password hashes
--tables          Enumerate DBMS database tables
--columns         Enumerate DBMS database table columns
--schema          Enumerate DBMS schema
--dump            Dump DBMS database table entries
--dump-all        Dump all DBMS databases tables entries
-D DB             DBMS database to enumerate
-T TBL            DBMS database table(s) to enumerate
-C COL            DBMS database table column(s) to enumerate
```

Figura 6.51 Opciones de ejecución del comando SQLmap.

Además, si detecta alguna vulnerabilidad, tiene la capacidad de poder atacar el servidor para descubrir nombres de tablas, descargar la base de datos y realizar consultas SQL de forma automática.

Una manera sencilla para identificar sitios web con la vulnerabilidad de SQL Injection consiste en añadir algunos caracteres a la url; por ejemplo, comillas, coma o punto, o, si la página está en PHP y se tiene una url donde pase un parámetro para una búsqueda, se puede intentar añadir una comilla al final.

Hacer inyecciones básicamente será usar querys SQL, como son el caso de union y select, y también los famosos joins. Solo es cuestión de manipular en la url de la página hasta poder encontrar un error que indique que la sintaxis

no resulta correcta y dar con el nombre de la tabla propensa o vulnerable a su acceso.

The screenshot shows a web page titled "TEST and Demonstration site for Acunetix Web Vulnerability Scanner". The main content area displays an error message: "Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '' at line 1 Warning: mysql\_fetch\_array() expects parameter 1 to be resource, boolean given in /hj/var/www/listproducts.php on line 74". To the left of the error message is a sidebar containing links: "search art", "Browse categories", "Browse artists", "Your cart", "Signup", "Your profile", "Our guestbook", and "AJAX Demo".

Figura 6.52 Error de sintaxis indicando la presencia de una inyección de SQL.

### 6.13.3 XSScrapy

XSScrapy <https://github.com/DanMcInerney/xsscrapy> es una aplicación basada en Scrapy <https://scrapy.org> y permite encontrar vulnerabilidades de los tipos cross-site scripting (XSS) y de inyección de SQL.

```
usage: xsscrapy.py [-h] [-u URL] [-l LOGIN] [-p PASSWORD] [-c CONNECTIONS]
                   [-r RATELIMIT] [--basic] [-k COOKIE]

optional arguments:
  -h, --help            show this help message and exit
  -u URL, --url URL    URL to scan; -u http://example.com
  -l LOGIN, --login LOGIN
                        Login name; -l danmcinerney
  -p PASSWORD, --password PASSWORD
                        Password; -p pa$$w0rd
  -c CONNECTIONS, --connections CONNECTIONS
                        Set the max number of simultaneous connections
                        allowed, default=30
  -r RATELIMIT, --ratelimit RATELIMIT
                        Rate in requests per minute, default=0
  --basic              Use HTTP Basic Auth to login
  -k COOKIE, --cookie COOKIE
                        Cookie key; --cookie
                        SessionID=af0h3193e9103bca9318031bcdf
```

Figura 6.53 Opciones de ejecución de XSScrapy.

Al intentar ejecutarlo sobre un dominio vulnerable a XSS y SQL Injection, podremos ver en los mensajes de información de log el caso detectado.

```
[scrappy] DEBUG: Crawled (200) <GET http://testphp.vulnweb.com/php/params.php?param1=1&param2=2> (referer: http://testphp.vulnweb.com/php/params.php?param1=1&param2=2)
[scrappy] DEBUG: Crawled (200) <GET http://testphp.vulnweb.com/artists.php?artist=2> (referer: http://testphp.vulnweb.com/php/params.php?param1=1&param2=2)
[scrappy] DEBUG: Crawled (200) <POST http://testphp.vulnweb.com/secured/meuser.php> (referer: http://testphp.vulnweb.com/php/params.php?param1=1&param2=2)
[scrappy] DEBUG: CRAWLED (200) <GET http://testphp.vulnweb.com/php/params.php?param1=1&param2=2> (referer: tagjuv?"O")
[scrappy] DEBUG: Crawled (404) <GET http://testphp.vulnweb.com/listproducts.php?artist=1&category=4>
[scrappy] INFO: URL: http://testphp.vulnweb.com/artists.php?artist=3
[scrappy] INFO: Response URL: http://testphp.vulnweb.com/artists.php?artist=2&id=1&name=clickable/&tagjhj
[scrappy] INFO: unfiltered: N/A
[scrappy] INFO: Payload: tagjhj%"O<></tagjhj>
[scrappy] INFO: Type: url
[scrappy] INFO: Injection point: artist
[scrappy] INFO: Line: Possible SQL injection error - suspected DBMS: MySQL, regex used: warning_revasq_...
[scrappy] WARNING: dropped: No XSS vulns in http://testphp.vulnweb.com/artists.php?artist=2&id=1&name=clickable/&tagjhj
[scrappy] DEBUG: sending payloads url: http://testphp.vulnweb.com/listproducts.php?artist=1&category=4&id=1
[scrappy] DEBUG: sending payloads url: http://testphp.vulnweb.com/listproducts.php?artist=1&category=4&id=2
[scrappy] DEBUG: sending payloads cookie header
[scrappy] DEBUG: sending payloads referer header
```

Figura 6.54 Opciones de ejecución de XSScrapy.

## 6.14 Seguridad en Django

El proyecto Pony Checkup <https://www.ponycheckup.com> permite realizar comprobaciones de seguridad de forma automatizada para los sitios web desarrollados con Django. Entre las principales comprobaciones de seguridad, podemos destacar HTTPS habilitado, manejo de cabeceras y cookies de forma segura, protección contra clickjacking y vulnerabilidad Heartbleed.



Figura 6.55 Proyecto para comprobar la seguridad de aplicaciones Django.

**Pony checkup report for http://djangoproject.com**

Check another URL   [Twitter](#) [Follow @pentestmonkey](#)

**Overall score**  
We found no ways you could improve the security of this website. Your overall rating is 100%.

We think this site runs Django.  
We checked all four out of the five Django-specific security checks and found them to be valid. However, if it's possible to exploit them, the website will be 100% safe.

Note that the results of this test are only valid for websites that actually run Django – otherwise the test will generate mostly false positives.

**HTTPS**  
 **HTTPS available**  
We were able to connect to your website using HTTPS. Note that this does not perform any checks of the configuration options. If you'd like to get a more detailed analysis, use the [Quick SSL Test](#).

Figura 6.56 Validación de HTTPS sobre el dominio djangoproject.com.

#### 6.14.1 Protección ante ataques XSS

Los ataques XSS permiten a un usuario injectar scripts del lado del cliente en los navegadores de otros usuarios. Esto, generalmente, se logra al almacenar los scripts maliciosos en la base de datos, donde se recuperarán y mostrarán a otros usuarios, o al hacer que los usuarios hagan clic en un enlace que hará que el navegador del usuario ejecute el JavaScript del atacante.

El uso de plantillas de Django puede proteger las aplicaciones contra la mayoría de los ataques XSS. Las plantillas de Django permiten escapar caracteres especiales que son particularmente peligrosos para HTML.

#### 6.14.2 Protección ante ataques CSRF

Django cuenta con protección integrada contra la mayoría de los tipos de ataques CSRF, siempre que lo haya habilitado en los formularios. La protección CSRF funciona al verificar un token secreto en cada solicitud POST. Esto garantiza que un usuario malintencionado no pueda reproducir un formulario POST y que otro usuario que haya iniciado sesión envíe involuntariamente ese formulario. El usuario malintencionado tendría que conocer el secreto, que es específico del usuario que realizó la solicitud original.

Cuando se implementa Django con HTTPS, **CsrfViewMiddleware** verificará que la cabecera de referencia HTTP esté configurada en una url en el mismo origen (incluido el subdominio y el puerto). Debido a que HTTPS proporciona seguridad adicional, resulta obligatorio garantizar que las conexiones usen HTTPS donde esté disponible, que reenvíen solicitudes de conexión inseguras y que utilicen la cabecera **HSTS** para los navegadores compatibles.

En cualquier plantilla que el usuario pueda enviar datos mediante POST, podríamos añadir una etiqueta **csrf\_token**. En la documentación oficial, podemos obtener más información: <https://docs.djangoproject.com/en/2.2/ref/csrf/#using-csrf>

```
<form action="/submit" method="POST">
    {% csrf_token %}
    <!-- other form fields -->
</form>
```

El **token CSRF** se validará automáticamente en el momento en el que se realice el submit del formulario:

```
@app.route("/submit", methods=["GET", "POST"])
def submit():
    f = MyForm()
    if form.validate_on_submit():
        # csrf token also validated
        return redirect("/")
    return render_template("view.html", form=f)
```

#### 6.14.3 Protección de inyección de SQL

La inyección de SQL constituye un tipo de ataque en el que un usuario malintencionado puede ejecutar código SQL arbitrario en una base de datos. Esto puede resultar en la eliminación de registros o la fuga de datos.

Los conjuntos de consultas de Django se encuentran protegidos de la inyección de SQL, ya que sus consultas se construyen utilizando consultas parametrizadas. El código SQL de una consulta se define por separado de los parámetros de la consulta. Dado que los parámetros pueden ser proporcionados por el usuario y, por lo tanto, son inseguros, el controlador de base de datos subyacente los escapa.

Django también ofrece a los desarrolladores la opción de poder ejecutar consultas SQL personalizadas. Estas capacidades deben usarse con precaución y siempre se ha de tener cuidado de escapar de cualquier parámetro que el usuario pueda controlar.

#### 6.14.4 Protección de clickjacking

Clickjacking es un tipo de ataque en el que un sitio malicioso envuelve otro sitio en un marco. Dicho ataque puede hacer que un usuario confiado sea engañado para que realice acciones involuntarias en el sitio de destino.

Django contiene protección de clickjacking en forma de Middleware con la cabecera **X-Frame Options** que, en un navegador compatible, puede evitar que un sitio se renderice dentro de un marco.

#### 6.14.5 SSL/HTTPS

Siempre supone una buena práctica implementar su sitio de forma segura mediante HTTPS. En otro caso, pudiera ocurrir que los usuarios malintencionados obtuvieran credenciales de autenticación o cualquier otra información transferida entre el cliente y el servidor. Si desea la protección que proporciona HTTPS y la ha habilitado en su servidor, existen algunos pasos adicionales que necesitaría realizar en un proyecto con Django:

- Configurar la cabecera **SECURE\_PROXY\_SSL\_HEADER**:
  - [https://docs.djangoproject.com/en/2.2/ref/settings/#std:setting-SECURE\\_PROXY\\_SSL\\_HEADER](https://docs.djangoproject.com/en/2.2/ref/settings/#std:setting-SECURE_PROXY_SSL_HEADER)
- Establecer la cabecera **SECURE\_SSL\_REDIRECT** en True para que las solicitudes, a través de HTTP, se redirijan a HTTPS:
  - [https://docs.djangoproject.com/en/2.2/ref/settings/#std:setting-SECURE\\_SSL\\_REDIRECT](https://docs.djangoproject.com/en/2.2/ref/settings/#std:setting-SECURE_SSL_REDIRECT)
- Utilizar cookies de forma segura. Es importante establecer la configuración de **SESSION\_COOKIE\_SECURE** y **CSRF\_COOKIE\_SECURE** en True. Esto le indica al navegador que solo envíe estas cookies a través de conexiones HTTPS.
- Establecer la cabecera **HTTP Strict Transport Security (HSTS)**. HSTS es una cabecera HTTP que informa al navegador de que todas las conexiones futuras a un sitio en particular siempre deben usar HTTPS:
  - <https://docs.djangoproject.com/en/2.2/ref/middleware/#http-strict-transport-security>

## 6.15 Otras herramientas de seguridad

### 6.15.1 Yosai

Yosai <https://yosaiproject.github.io/yosai> es un framework con el cual puede integrar algunas de las funcionalidades de autenticación y autorización en aplicaciones Python. Se basa en Apache Shiro <https://shiro.apache.org>, un proyecto muy utilizado en aplicaciones Java Enterprise.



Figura 6.57 Autenticación con el framework.

El código fuente del proyecto lo encontramos en el repositorio de GitHub <https://github.com/YosaiProject/yosai>.

Además, la documentación ofrece muchos ejemplos de aplicación, junto con ejemplos de código.

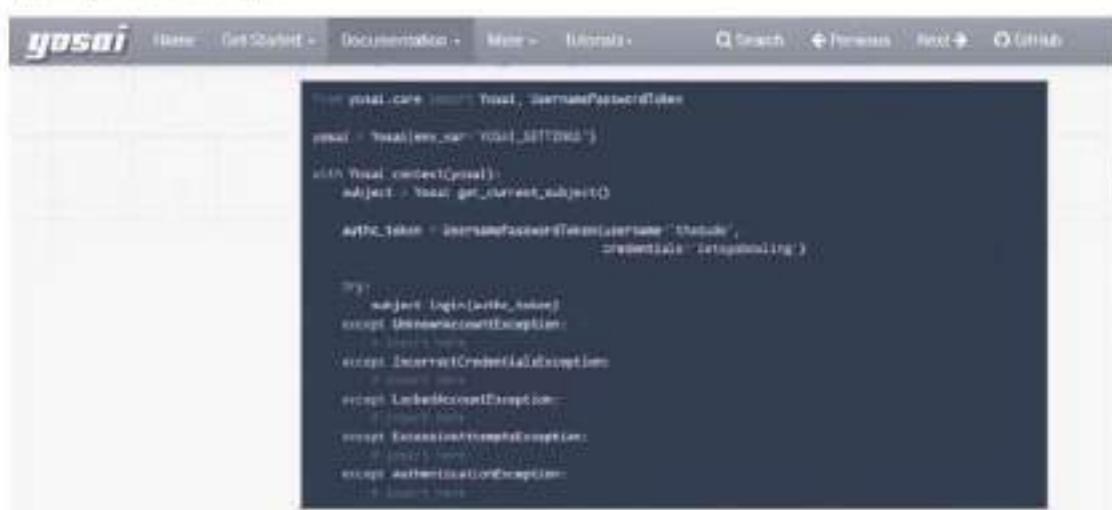


Figura 6.58 Ejemplos de uso del framework.

### 6.15.2 Flask-Security

Flask-Security <https://pythonhosted.org/Flask-Security> constituye una extensión de Flask, que añade funcionalidades relacionadas con la seguridad en aplicaciones desarrolladas en Flask, como **autenticación, generación de contraseñas y administración de roles**. Entre las principales características, podemos destacar:

- **Autenticación basada en sesión.** La autenticación basada en sesión se realiza con la extensión **Flask-Login** <https://flask-login.readthedocs.io>. Flask-Security maneja la configuración de Flask-Login automáticamente, en función de algunos de sus propios valores de configuración, y utiliza la función de token alternativo de Flask-Login para recordar a los usuarios cuándo su sesión ha expirado.
- **Acceso basado en roles.** Flask-Security implementa una administración de roles que puede asociar un rol o varios roles a cualquier usuario; por ejemplo, puede asignar roles como administrador, editor, superusuario o una combinación de dichos roles a un determinado usuario. El control de acceso se basa en el nombre del rol y todos los roles han de disponer de un nombre único. Esta característica se implementa utilizando la extensión **Flask-Principal**.
- **Contraseña hashing.** El hashing de contraseña está habilitado con **passlib**. Las contraseñas se incluyen con la función **bcrypt** por defecto, con la cual se puede configurar el algoritmo de hash que se utilizará.
- **Autenticación HTTP básica.** La autenticación HTTP básica se implementa usando un decorador en el método que renderiza la vista de autenticación.
- **Autenticación basada en token.** La autenticación basada en token se habilita al recuperar el token de autenticación del usuario, realizando una POST HTTP con los detalles de autenticación como datos JSON contra elEndPoint de autenticación. Una llamada exitosa a esteEndPoint devolverá la ID del usuario y el token de autenticación. Este token se puede utilizar en solicitudes posteriores a recursos protegidos. El token de autenticación se proporciona en la solicitud a través de un encabezado HTTP o un parámetro de cadena de consulta.
- **Confirmación de correo electrónico.** Resulta posible solicitar que los nuevos usuarios confirmen su dirección de correo electrónico. Flask-Security enviará un mensaje de correo electrónico a cualquier usuario nuevo con un enlace de confirmación. Al navegar hasta el enlace de confirmación, el usuario iniciará sesión automáticamente.

También existe una vista para reenviar un enlace de confirmación a un correo electrónico determinado, en el caso de que el usuario intente utilizar un token caducado o si perdió el correo electrónico anterior. Los enlaces de confirmación se pueden configurar para que caduquen después de un periodo de tiempo específico.

- **Restablecimiento y recuperación de contraseña.** El restablecimiento y la recuperación de la contraseña se encuentran disponibles cuando un usuario olvida su contraseña. Flask-Security envía un correo electrónico al usuario con un enlace a una vista donde puede restablecer su contraseña. Una vez que se restablece esta, se registra automáticamente y puede usar la nueva contraseña a partir de ese momento. Los enlaces de restablecimiento de contraseña se pueden configurar para que caduquen después de un periodo de tiempo específico.
- **Registro de usuario.** Flask-Security viene empaquetado con una vista de registro de usuario básica. Con esta vista, los usuarios solo necesitan proporcionar una dirección de correo electrónico y su contraseña para registrarse.

#### 6.15.3 OWASP Python Security Project

[https://www.owasp.org/index.php/OWASP\\_Python\\_Security\\_Project](https://www.owasp.org/index.php/OWASP_Python_Security_Project) es un proyecto de código abierto que tiene como objetivo mostrar aquellas herramientas Python que facilitan a los desarrolladores y profesionales de la seguridad desarrollar aplicaciones más resistentes a posibles ataques.

El proyecto ha sido diseñado para explorar cómo se pueden desarrollar aplicaciones web en Python, al abordar el problema desde varios puntos de vista diferentes: análisis de caja blanca y de caja negra, así como análisis estructural y funcional. En la dirección <http://www.pythonsecurity.org/libs> encontramos las principales librerías organizadas por categoría funcional.

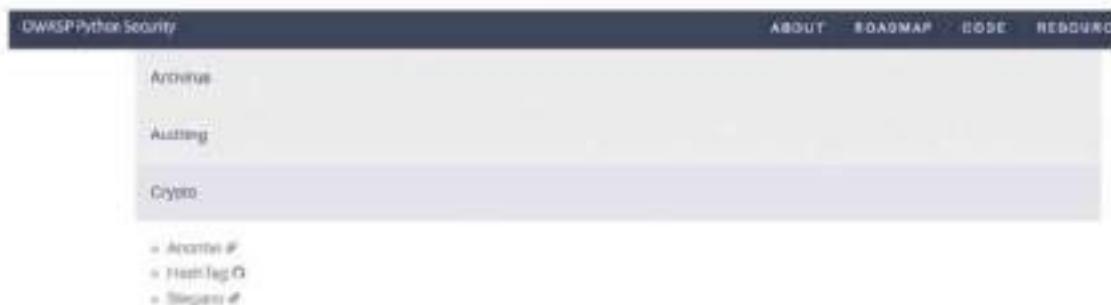


Figura 6.59 Portal OWASP Python Security Project.

## **Capítulo 7. Análisis estático y dinámico en aplicaciones C/C++**

El desarrollo de aplicaciones software robustas, predecibles en su ejecución y libres de vulnerabilidades conforma una tarea difícil, y desarrollarlas sin fallos de seguridad desde cero es una tarea hoy por hoy imposible. Casi todos los ataques a las aplicaciones software poseen una causa fundamental: el software no es seguro debido a defectos en su diseño, codificación, pruebas y operaciones.

Una **vulnerabilidad** constituye un defecto de software que un atacante puede explotar. Los defectos típicos caen en una de dos categorías: **errores y defectos**.

Un **error o bug** es un problema introducido durante la implementación del software. La mayoría de los bugs pueden ser fácilmente descubiertos y corregidos. Algunos ejemplos se hallan en los desbordamientos de búfer, las condiciones de ejecución, las llamadas al sistema no seguras o la validación incorrecta de una entrada.

Un **defecto** supone un problema a un nivel mucho más profundo. Los defectos se muestran más sutiles; por lo general, se originan en el diseño y se ejecutan en el código. Ejemplos de defectos incluyen problemas en el diseño o en el control de errores y control de acceso, que no siguen una lógica.

En la práctica, nos encontramos con que los problemas de seguridad de software se dividen 50/50 entre los errores y defectos. Así, el descubrimiento y la eliminación de errores durante el análisis de código contemplan la mitad del problema al abordar la seguridad del software, de forma que, incluso para expertos en el mundo de la programación, cuando el desarrollo de una aplicación comprende miles de líneas de código y cuando es desarrollada por múltiples equipos, los errores de este tipo se vuelven más complejos de detectar.

Por dicho motivo, estas herramientas pueden constituir un buen apoyo para detectar algunas vulnerabilidades de carácter crítico para la aplicación. Cabe mencionar que dichas herramientas pueden generar una falsa sensación de seguridad, al no reportar cierto tipo de vulnerabilidades, las cuales, sin embargo, pueden producirse bajo determinadas condiciones o que, simplemente, no son capaces de detectar, debido a su propia naturaleza.

El uso, por tanto, de tales herramientas ha de considerarse únicamente como un paso más durante el proceso de auditoría de software. Entre las ventajas que ofrecen este tipo de análisis, podemos destacar:

- Se pueden detectar fugas de memoria y desbordamiento de búfer.
- Se ha de analizar la complejidad ciclomática, según los estándares de codificación de cada lenguaje particular.
- Podemos encontrar muchas herramientas que permiten automatizar los análisis.
- Las herramientas de automatización proporcionan recomendaciones a nivel de código, con lo que reducen el tiempo de la investigación.
- El análisis dinámico identifica problemas en un entorno en tiempo de ejecución.
- Se puede llegar a reducir el porcentaje de errores durante la fase de pruebas.

## 7.1 Análisis estático

El análisis de código estático es una técnica para analizar el código sin ejecutarlo realmente. En la mayoría de los casos, el análisis se realiza en el código fuente o en el código objeto. Como ya sabemos, resulta trivial descompilar una aplicación de Android.

A partir de la experiencia con el desarrollo de algunas aplicaciones, podemos llegar a la conclusión de que lo mejor es siempre primero realizar un análisis estático a nivel de código a través de herramientas que se integren en nuestro entorno de desarrollo. Las herramientas de análisis estático se integran muy bien con los sistemas de integración continua, como Sonar y Jenkins. Además, al integrarse dentro del ciclo de integración continua, permite mejorar la calidad del código en cada iteración; por ejemplo, si estamos haciendo algún tipo de refactor. El objetivo del análisis estático reside en disponer de una cobertura de código, junto con un control de flujo analizando la lógica de la aplicación.

Los beneficios de hacer el análisis estático parte de la integración continua estriban en obtener retroalimentación mucho más rápidamente después de haber realizado y testeado los cambios en el código. Cuanto más rápido sea el ciclo de retroalimentación, mejor será el desarrollo.

Entre las **ventajas del análisis estático**, podemos destacar:

- Permite detectar vulnerabilidades de forma temprana durante el ciclo de desarrollo de software.

- Permite llevar a cabo análisis de aplicaciones de las cuales disponemos de su código fuente (enfoque **whitebox**).
- El análisis no requiere inputs de entrada por parte del usuario, por lo que se elimina la necesidad de crear gran variedad de test.

Entre las **desventajas del análisis estático**, podemos destacar:

- Pueden generar una falsa sensación de seguridad si no se reportan vulnerabilidades.
- No existen herramientas automatizadas para todos los lenguajes.
- Si se dispone de herramientas automatizadas, puede resultar un proceso lento.

### 7.1.1 Code Analyzer

Las técnicas de análisis estático se pueden aprovechar para descubrir problemas como fugas de memoria, variables sin inicializar, código muerto o desbordamientos de búfer, entre otros. Esto se puede hacer usando IDE, como Android Studio o Xcode, si el código fuente de la aplicación se halla disponible. El analizador estático recorre cada posible ruta de código, e identifica errores lógicos, como fugas de memoria.

Code Analyzer <http://www.codeanalyzer.teel.ws/> es una aplicación Java para analizar diferentes métricas de software definidas por lenguajes como C, C++ o Java.

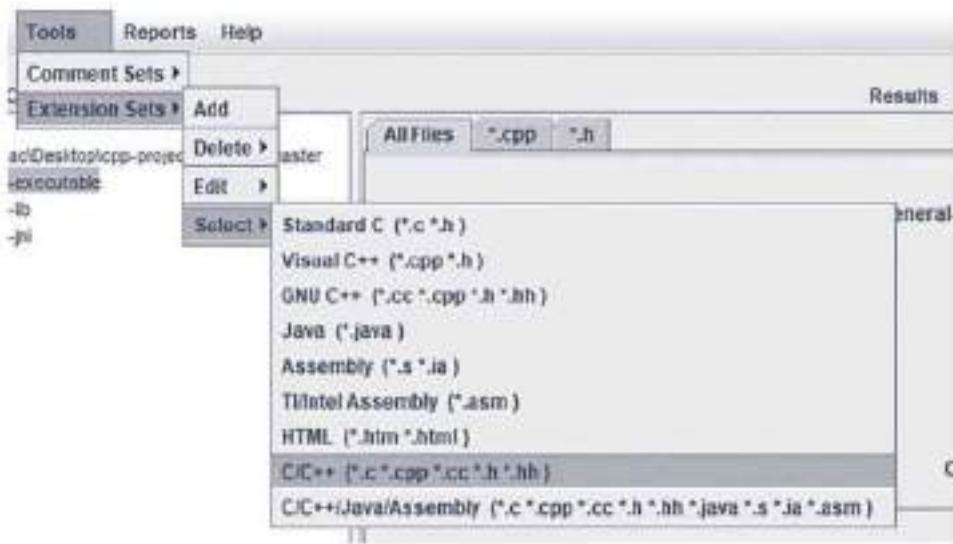


Figura 7.1 Selección del tipo de proyecto para analizar desde el menú Tools>Extension Sets.

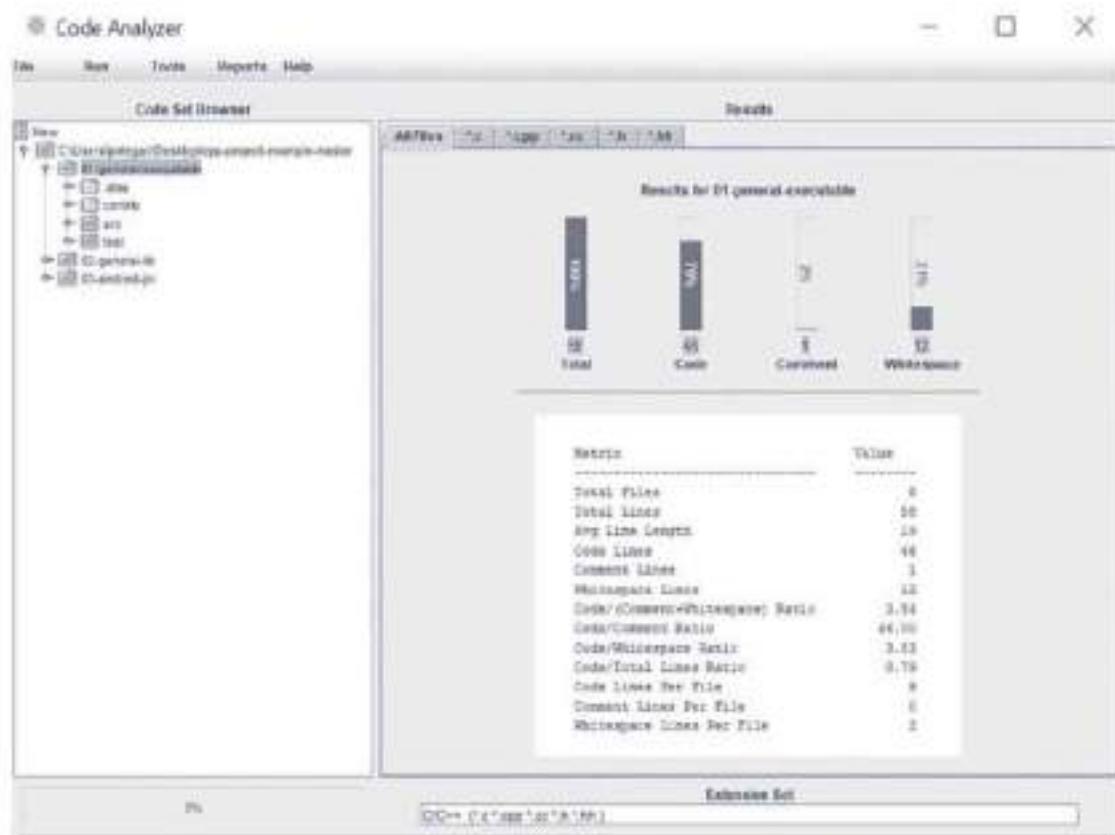


Figura 7.2 Interfaz Code Analyzer, donde podemos ver un análisis del código.

## 7.2 Análisis estático de código C/C++

Testear nuestro código de forma estática ayudará a localizar funciones que pueden ser propensas a vulnerabilidades. Las herramientas que veremos a continuación servirán de complemento a los warnings que puede ofrecernos un compilador para alertarnos sobre posibles usos indebidos del lenguaje utilizado. El análisis estático para las aplicaciones en C/C++ podría realizarse utilizando herramientas gratuitas como **Flawfinder** o **Clang**.

### 7.2.1 Flawfinder

Flawfinder, <https://packages.debian.org/search?keywords=flawfinder> es un analizador de propósito general para encontrar e informar de debilidades potenciales en código fuente, tanto de C como de C++. Flawfinder permite detectar

casos como riesgos de **desbordamiento de búfer** (*buffer overflow*), uso de funciones de acceso al sistema de ficheros o ejecución de comandos y procesos en background.

Para ello, utiliza la coincidencia de patrones basada en texto para detectar llamadas a funciones de C/C++ vulnerables o mal utilizadas. Permite examinar ficheros C/C++ en busca de vulnerabilidades potenciales catalogándolas de 0 a 5, en función de su nivel de criticidad. Un enfoque utilizado para analizar código con esta herramienta radica en buscar posibles bugs de criticidad alta, además de analizar los puntos del código donde se manejen entradas, para determinar si se están aplicando los controles adecuados. Esto último puede hacerse con el parámetro `-inputs`.

Entre las vulnerabilidades que es capaz de detectar, podemos destacar:

- Llamadas a funciones de la librería de C, que pueden constituir el origen de **vulnerabilidades de desbordamiento de búfer** (`strcpy` o `sprintf`).
- Llamadas a funciones de la librería de C potencialmente vulnerables a **ataques de formato de cadena** (`sprintf` o `printf`).
- Posibles condiciones de carrera (**race conditions**) en el manejo de archivos.

La instalación se puede realizar directamente desde los repositorios de Debian y se halla disponible en la mayoría de las distribuciones basadas en Debian: `sudo apt-get install flawfinder`.

Internamente, utiliza una base de datos denominada **Ruleset**, que incluye un elevado número de funciones de C/C++, que pueden desencadenar determinadas vulnerabilidades, así como funciones específicas de Unix y Windows que pueden ser problemáticas. Para utilizar esta herramienta, únicamente especificamos el directorio o el fichero fuente que deseemos analizar. En nuestro caso, utilizaremos el siguiente código en C++, extraído de *stackoverflow*: <http://stackoverflow.com/questions/8782852/c-buffer-overflow>

```
#include <string>
#include <iostream>

using namespace std;

int main()
{
```

```

begin:
int authentication = 0;
char cUsername[10], cPassword[10];
char cUser[10], cPass[10];
cout << "Username: ";
cin >> cUser;
cout << "Pass: ";
cin >> cPass;

strcpy(cUsername, cUser);
strcpy(cPassword, cPass);

if(strcmp(cUsername, "admin") == 0 && strcmp(cPassword, "adminpass") == 0)
{
    authentication = 1;
}
if(authentication)
{
    cout << "Access granted\n";
    cout << (char)authentication;
}
else
{
    cout << "Wrong username and password\n";
}

system("pause");
goto begin;
}

```

Al ejecutar Flawfinder, generamos un reporte en formato html de la siguiente forma:

```
flawfinder --html --context -F example1.cpp > report.html
```

Examining example1.cpp

- example1.cpp:19: [4] (buffer) strcpy: Does not check for buffer overflows when copying to destination. Consider using strncpy or strlcpy (warning, strncpy is easily misused).
 

```
strcpy(cUsername, cUser);
```
- example1.cpp:20: [4] (buffer) strcpy: Does not check for buffer overflows when copying to destination. Consider using strncpy or strlcpy (warning, strncpy is easily misused).
 

```
strcpy(cPassword, cPass);
```
- example1.cpp:36: [4] (shell) system: This causes a new program to execute and is difficult to use safely. Try using a library call that implements the same functionality if available.
 

```
system("pause");
```

Figura 7.3 Reporte de Flawfinder, donde se detecta el uso de funciones inseguras.

El reporte de Flawfinder nos muestra tres llamadas que no se están usando de forma segura: por un lado, del uso de funciones inseguras como strcpy y, por otro, del uso de la función system, la cual puede dar lugar a ciertos problemas de seguridad; por ejemplo, la función **char \*strcpy (char \* destino, const char \* origen)** permite copiar una cadena origen en otra cadena destino, pero, si no se validan los tamaños de ambas cadenas, se puede considerar un uso seguro.

Lo mismo ocurre con otras funciones como **int sscanf (const char \* s o const char \* format)**. En el siguiente ejemplo vemos cómo detecta el uso de la función sscanf de forma insegura, ya que puede originar un *buffer overflow*, debido a que no se está validando la longitud de los datos de entrada:

```
#include <stdio.h>
#define LIST(...) __VA_ARGS__
#define scanf_param( fmt, param, str, args ) {
    char fmt2[100];
    sprintf( fmt2, fmt, LIST param );
    sscanf( str, fmt2, LIST args );
}

enum { X=3 };
#define Y X+1

int main(){
    char str1[10], str2[10];
    scanf_param( " %%%is %%%is", (X,Y), " 123 4567", (&str1, &str2) );
    printf("str1: '%s' str2: '%s'\n", str1, str2 );
}
```

La salida muestra el uso inseguro de la función sscanf:

```
Number of dangerous functions in C/C++ ruleset:160
Examining example.c
Example.c:8 [4] (buffer) sscanf:
The sscanf family's % operation, without a limit specification, permits buffer overflow.
Specify a limit to %s or use a different input function. If the scanf format is influenceable
by an attacker, it's exploitable.
sscanf(str,fmt2,LIST args);
```

Flawfinder puede producir algunos falsos positivos; después de todo, no todas las llamadas a strcpy resultan en una vulnerabilidad de seguridad e incluyen

un mecanismo para informar falsos positivos al incluir un comentario en el código. Al introducir este comentario, ignorará los errores encontrados en esa línea que ha identificado como falso positivo:

```
strcpy (largebuffer, smallbuffer) / * Flawfinder: ignore* /
```

### 7.2.2 Clang

Clang [http://clang-analyzer.llvm.org/available\\_checks.html](http://clang-analyzer.llvm.org/available_checks.html) presenta la capacidad de detectar memory leaks, variables que no han sido inicializadas, posibles problemas con null pointers o uso de punteros de forma incorrecta. Entre los casos que puede detectar, podemos destacar el **uso de variables sin inicializar** y el **uso inseguro de funciones**.

### 7.2.3 Uso de variables sin inicializar

En este ejemplo vemos cómo, en la llamada a la función f desde la función test, se está enviando un argumento pasado por valor de una variable que no está inicializada:

```
struct S {  
    int x;  
};  
  
void f(struct S s);  
  
void test() {  
    struct S s;  
    f(s); // warning: Argumento pasado por valor contiene datos sin inicializar  
}
```

En este ejemplo, declaramos un puntero \*pc de objeto de una clase C, pero dicho punto no se ha inicializado, lo que puede originar un null pointer:

```
class C {  
public:  
    void f();  
};  
  
void test() {  
    C *pc;  
    pc->f(); // warning: puntero a objeto sin inicializar  
}
```

El mismo problema lo podemos tener si inicializamos el puntero a null:

```
class C {
public:
    void f();
};

void test() {
    C *pc = null;
    pc->f(); // warn: puntero a objeto con valor null
}
```

A continuación se muestra el uso de un puntero después de liberar la memoria con delete:

```
void f(int *p);

void testUseMiddleArgAfterDelete(int *p) {
    delete p;
    f(p); // warning al intentar usar un puntero después de liberarlo
}
```

Si liberamos memoria de un puntero que, probablemente, estemos utilizando, tendremos un problema de memoria:

```
void test() {
    int *p = (int *)__builtin_alloca(sizeof(int));
    delete p; //warning al intentar liberar memoria que se está utilizando
}
```

También pueden surgir problemas si liberamos memoria dos veces seguidas con el mismo puntero:

```
void test() {
    int *p = new int;
    delete p;
    delete p; //warning al intentar liberar memoria de un puntero que ya ha sido liberado
}
```

#### 7.2.4 Uso inseguro de funciones

Funciones como strcpy y gets resultan inseguras si no se validan las longitudes de las cadenas que se pasan por parámetro:

```
void test() {
    char x[4];
    char *y = "abcd";
    strcpy(x, y); // uso inseguro de la función strcpy
}
```

```
void test() {
    char buff[1024];
    gets(buff); // uso inseguro de la función gets
}
```

El uso de la función strcpy está desaconsejado, ya que no comprueba los desbordamientos al copiar en el búfer destino. Se recomienda mejor el uso de strncpy o strlcpy. En este ejemplo vemos que, si el argumento 2 que se pasa a esta llamada de función es mayor que 1024 caracteres, no podrá copiar más datos de los que pueden manejarse, lo que resulta en un desbordamiento del búfer:

```
int main( int argc, char *argv[] )
{
    char cmd[1024];
    if ( argc == 2){
        strcpy(cmd, argv[1]);
    }
}
```

Tampoco se aconseja el uso de sprintf, ya que no comprueba desbordamientos de búfer. Se recomienda, en cambio, el uso de snprintf o vsnprintf.

Si estamos usando **popen** para ejecutar un comando, resulta importante que el primer argumento de esta llamada se compruebe, para asegurarse de que no proviene de una fuente no confiable sin verificar primero que no contiene un comando malicioso:

```
FILE *fp = NULL;
fp = popen(cmd, "r" );
```

### 7.2.5 RATS

RATS <https://packages.debian.org/search?keywords=rats> es un analizador de propósito general utilizado para detectar potenciales problemas de seguridad en varios lenguajes de programación.

De forma similar a como realizan los análisis las herramientas comentadas anteriormente, RATS (Rough Auditing Tool for Security) es una herramienta open source que permite analizar, de forma estática, nuestro código, con el fin de localizar diversos tipos de vulnerabilidades en lenguajes como C, C++, Perl, PHP y Python. Su uso resulta muy similar a Flawfinder: únicamente especificamos como parámetro el fichero o directorio que queremos analizar. En el ejemplo hemos añadido a nuestro **script en Python** una llamada a la función eval que se considera insegura. En la salida, nos muestra una advertencia sobre el uso de dicha función y nos indica también los inputs que pueden ser vulnerables:

```
$ echo 'eval(token, {"__builtins__":{}})' >>>pythonVulnerable.py
$ rats -resultsonly -w 1 --context pythonVulnerable.py
pythonVulnerable.py: 1: High: eval
eval(token, {"__builtins__":{}})
Argument 1 to this function call should be checked to ensure that it does not come from
and untrusted source without first verifying that it contains nothing dangerous
```

### 7.2.6 Vulnerabilidad cadena de formato (format string)

En lenguajes de programación como C/C++, donde un programador no está obligado a definir el tipo de dato que mostrar durante la declaración de la función encargada de dicha tarea, a veces podemos encontrarnos con código vulnerable a una explotación de tipo format string, como podemos ver en el siguiente artículo: [https://www.owasp.org/index.php/Format\\_string\\_attack](https://www.owasp.org/index.php/Format_string_attack)

El principal problema para detectar tales vulnerabilidades estriba en que el compilador solo notifica al programador si la función empleaba menos parámetros de entrada de los estrictamente necesarios para su funcionamiento. Este comportamiento ha sido modificado en las últimas versiones de los compiladores, indicando al programador un mensaje de advertencia por los riesgos que supone la forma en que ha declarado la función; por ejemplo, el compilador GCC muestra el siguiente mensaje de advertencia: **warning: format not a string literal and no format arguments.**

Con el objetivo de entender, de forma más sencilla, el funcionamiento de los ataques **format string**, se mostrará el siguiente código en C, donde se solicita al usuario que introduzca una cadena de entrada para, posteriormente, ser mostrada con la función printf:

```
#include <stdio.h>
int main(void) {
    char texto[30];
    scanf("%29s", texto);
    printf("%s", texto); //ok
    return 0;
}
```

Cuando vamos a mostrar la variable texto, lo hacemos con el modificador %s, lo que indica que la variable "texto" será mostrada como de tipo cadena de caracteres (string). Si, en vez del código anterior, el programador hubiera omitido especificar el tipo de dato que mostrar mediante la sintaxis "%s", el código presentaría una vulnerabilidad de tipo format string:

```
#include <stdio.h>
int main(void) {
    char texto[30];
    scanf("%29s", texto);
    printf(texto); //VULNERABLE
    return 0;
}
```

### 7.2.7 Pscan para detectar vulnerabilidades format string

Pscan <https://www.debian.org/security/audit/examples/pscan> es un paquete que podemos encontrar en distribuciones basadas en Debian, que permite auditar archivos fuente de C y C++ en busca de vulnerabilidades del tipo cadena de formato. Las vulnerabilidades de dicho tipo se producen normalmente en las llamadas a la función printf donde no se citan los argumentos correctamente; por ejemplo, si queremos imprimir el contenido de la variable búfer, podemos hacerlo de esta forma: printf(buffer);.

El problema de lo anterior reside en que se podría permitir a un atacante controlar la salida del programa, y se crearía un archivo llamado "%s" o similar. Para solucionarlo, la recomendación radica siempre en definir como primer ar-

gumento de la función el tipo de lo que se espera imprimir; en este caso, con %s indicamos que se trata de una cadena de caracteres printf ("%s", buffer);.

En la guía **Secure Programming for Linux and Unix** (<https://d Wheeler.com/secure-programs>) se explica cómo protegerse frente a tales ataques.

### 7.2.8 Buffer overflow

Muchas veces, el programador, ajeno a aspectos relacionados con la seguridad, no es consciente de las implicaciones que puede tener un simple puntero no liberado correctamente o las consecuencias que un array, cuyos límites no se están controlando bien, pueden acarrear en un sistema. Errores de este tipo siguen predominando como una de las vulnerabilidades más extendidas en lenguajes como C, C++ o ensamblador. La utilización de bibliotecas estándar que hacen uso de funciones como gets, strcpy o scanf debería evitarse, además de tener en cuenta otra serie de errores con consecuencias similares.

Buffer overflow se produce cuando un programa intenta poner más datos en un búfer de lo que realmente puede almacenar, o cuando un programa trata de poner los datos en un área de memoria fuera de los límites de un búfer. La causa más común de desbordamientos de búfer es el típico caso en el que el programa copia el búfer sin restringir el número de bytes que copiar.

A pesar de resultar una vulnerabilidad bien conocida desde los años ochenta, sigue siendo uno de los motivos por el que muchos sistemas operativos y aplicaciones son comprometidos. La facilidad con la que se puede localizar y aprovecharse de este tipo de vulnerabilidades para injectar un payload o código malicioso en el espacio de direcciones del proceso representa uno de los motivos de ser de una de las vulnerabilidades más explotadas hoy día. La incorrecta validación de datos de entrada, así como la falta de control sobre el tamaño de las variables, arrays o la incorrecta gestión de punteros, constituye uno de los mayores problemas que, desde hace tiempo, resultan los causantes directos de muchos problemas de seguridad críticos.

Este tipo de errores, generalmente, se producen por una incorrecta validación en los límites de un array, lo que produce la sobreescritura de valores en el stack (variables locales, EBP, RET address, etc.). Algo tan simple como el siguiente código, donde se lee una cadena introducida desde el teclado, puede generar un buffer overflow:

```
char direccion[24];
printf("Introduzca su dirección y pulse <Enter>\n");
gets(direccion);
```

Funciones como **gets()**, **strcpy()**, **strcat()**, **sprintf()**, **scanf()**, **sscanf()**, **fscanf()**, **vscanf()**, **vsprintf** o **vscanf()** deben ser, por tanto, evitadas, ya que no hacen ningún tipo de comprobación sobre la longitud de sus argumentos; es decir, que bien se hace un chequeo previo de los parámetros pasados a este tipo de funciones —por ejemplo, `if(strlen(origen) >= destino)`—, bien se debe usar alguna alternativa algo más segura.

Por ejemplo, en el caso de **strcpy()**, podemos utilizar **strncpy()** de la siguiente forma para copiar una cadena origen a otra destino de forma segura. Como se observa, se pasa como argumento adicional el tamaño máximo aceptable por el array destino:

```
strncpy(destino, origen, destino_size - 1)
```

Cabe destacar asimismo que funciones como **strncpy**, aunque sí proporcionan más protección que la clásica función **strcpy**, siguen siendo propensas a buffer overflow. En este caso, es el usuario quien controla el tercer argumento de la llamada a la función **strncpy** utilizado para indicar el número de caracteres copiados a búfer (string destino), lo que hace esta función totalmente insegura:

```
#include <stdio.h>
main(int argc, char **argv)
{
    int incorrectSize = atoi(argv[1]);
    int correctSize = atoi(argv[2]);
    char *buffer = (char *)malloc(correctSize+1);
    strncpy(buffer, argv[3], incorrectSize);
}
```

Funciones como **strcpy**, **gets** o **scanf** suelen ser bastante propensas a buffer overflow, por lo que se recomienda utilizar funciones más seguras que descarten los datos que excedan la longitud definida. En algunas de dichas funciones, como **strcpy\_s()**, **strcat\_s()**, **strncpy\_s()** o **strncat\_s()**, definidas en la versión C++ 11

—por ejemplo, en el caso de `strcpy_s()`—, se llevarán a cabo las siguientes comprobaciones, con el fin de evitar cualquier intento de desbordamiento de búfer:

- Los punteros origen y destino son comprobados para ver si son NULL.
- La longitud máxima del búfer de destino se comprobará para ver si es igual a cero, mayor que `size_t` o menor o igual a la longitud de la cadena de origen.
- Evita la copia cuando ambos objetos se solapan.

En caso de éxito, la función `strcpy_s()` copiará el contenido del array origen al destino, añadirá el carácter de fin de cadena a este y devolverá 0. En caso de detectar un overflow, a la cadena destino se le asignaría el carácter nulo siempre y cuando no sea un puntero nulo, y la longitud máxima de este sea mayor que cero y no superior a `size_t`. En dicho caso, la función devolverá un valor distinto de cero.

También podríamos gestionar el tamaño de los datos, con el fin de validar que los datos entrantes coinciden con el formato predeterminado de los campos y, así, prevenir los desbordamientos de búfer. En este caso, el búfer es de 32 bytes y se desborda cuando recibe más de esa cantidad. Un programa más seguro utilizando la función `strcpy_s()` sería:

```
#include <string.h>
#include <stdio.h>
int main(int argc, char * argv[]) {
    char buffer[32];
    strcpy_s(buffer, sizeof(buffer), argv[1]);
    printf("Hola %s \n", buffer);
    return 0;
}
```

Compiladores como el de Microsoft C++ mostrarán un mensaje de warning cuando detecten el uso de funciones peligrosas como `strcpy`, y recomendarán el uso de funciones más seguras como `strcpy_s`.

```
int main(int argc, char* argv[])
{
    char buf[20];
    strcpy(buf, argv[0]);
    char *_cdecl strcpy(char *_Dest, const char *_Source)
}
Warning: This function is banned by the Microsoft SDL. Please use strcpy_s or StringCchCopy[Ex] instead.
```

Figura 7.4 Recomendación por parte del compilador de C++ para usar `strcpy_s`.

En este repositorio de GitHub <https://github.com/wchen-r7/VuInCases> encontramos algunas pruebas de concepto para diferentes tipos de buffer overflow en C/C++.

■ Linux Stack Buffer Overflow	Add some READMEs to explain what each case is trying to demo	last month
■ Windows COM Ref Counting Use After Free	Update README.md	29 days ago
■ Windows Format String Arbitrary Write	Update README.md	29 days ago
■ Windows Heap Memory Leak	Typeo	last month
■ Windows Heap Overflow Arbitrarily Control	Add README	23 days ago
■ Windows Heap Overflow Info Leak	Update README	27 days ago
■ Windows Integer Overflow	Update README.md	29 days ago
■ Windows Stack Buffer Overflow	Add some READMEs to explain what each case is trying to demo	last month
■ Windows Unicode Buffer Overflow	Update README.md	29 days ago
■ Windows Unsafe DLL Loading	forgot the system("PAUSE")	28 days ago
■ Windows Use After Free to Type Confusion	Update README.md	29 days ago
■ LICENSE	Create LICENSE	last month
■ README.md	Update README.md	27 days ago

Figura 7.5 Repositorio de GitHub con ejemplos de buffer overflow.

En este recurso <https://fundacion-sadosky.github.io/guia-escritura-exploits/buffer-overflow> encontramos más información acerca de cómo podemos explotar una vulnerabilidad de este tipo.



Figura 7.6 Repositorio de exploits para buffer overflow.

### 7.2.9 Tipos de heap overflow

Gran parte de las vulnerabilidades relacionadas con la memoria dinámica se corresponden con fallos de programación que encajan en alguna de las siguientes categorías: **use after free**, **double free** y **dereference after free**. Las malas

prácticas a la hora de utilizar funciones relacionadas con la memoria dinámica (como malloc o free) suelen representar el origen de este tipo de problemas. Aprender no solamente buenas prácticas de programación, sino también conocer cómo funciona la asignación de memoria dinámica, resulta vital para entender de qué forma el heap puede aprovecharse para ejecutar un shellcode con el que comprometer un sistema, servidor o aplicación.

### 7.2.10 Vulnerabilidad use after free

Las vulnerabilidades conocidas como use after free (dangling pointer) ocurren cuando un programa continúa utilizando un puntero después de que este haya sido liberado. Los navegadores web suelen ser propensos a este tipo de vulnerabilidades, cuyas consecuencias pueden afectar directamente a la integridad y disponibilidad de la aplicación, y las cuales, a diferencia de vulnerabilidades como buffer overflow, resultan complejas de detectar mediante análisis estático. Generalmente este tipo de bugs se producen debido a errores de condición o errores en la lógica del propio programa encargado de liberar y reservar memoria.

Veamos un ejemplo sencillo. La función `get_pointer` recibe un puntero a entero (`int **ant`). Si dicho puntero `ant` es `NULL`, el puntero local `b` será liberado llamando al método `free()`. Sin embargo, la función `get_pointer` devolverá `b` en todos los casos. En tal situación, podría generar algún tipo de problema si la función que recibe dicho puntero no comprueba el "estado" de este antes de utilizarlo, con lo que se produce un use after free:

```
int *get_pointer(int *ant) {
    int *b = (int *)malloc(sizeof(int));
    if (ant == NULL) {
        //liberamos la memoria de b
        free(b);
    } else { *b = *ant; }
    return b; //problema si ant = null
}
```

Cabe recordar que la función de C/C++ `void free (void * ptr)` se encarga únicamente de desasignar (*deallocate*) chunks de memoria previamente asignados a dicho puntero con funciones como `malloc`, `calloc` o `realloc`, pero no modifica la dirección asignada a este; es decir, la dirección a la que apunta `b` después del `free` sigue siendo la misma. Sin embargo, la memoria en el heap asociada al

puntero ahora forma parte de la lista de chunks libres (FreeLists) a disposición del gestor del heap (heap manager) para posteriores reservas; es decir, en el momento en el que se hace el return b, la zona de memoria asignada al puntero ya podría estar siendo utilizada por el heap manager para realizar otras asignaciones de memoria.

### 7.2.11 Dereference after free

Un caso concreto de use after free, denominado dereference after free, es precisamente cuando intentamos acceder a memoria dinámica previamente liberada como consecuencia de la llamada al método **free()**. El término dereference se refiere a la acción de acceder a la variable a la que apunta el puntero en cuestión.

Desde la página del MITRE <http://cwe.mitre.org/data/definitions/416.html>, pueden encontrarse ejemplos prácticos sobre este tipo de vulnerabilidad. En el siguiente ejemplo el puntero a char buffer2 es liberado con la instrucción **free(buffer2)** aunque, más adelante, se vuelve a referenciar desde la función **strncpy**. Como consecuencia, resulta probable que se corrompan datos asignados posteriormente en dicha dirección de memoria tras haberla liberado, o que se produzca un crash de la aplicación al intentar sobrescribir "memoria aleatoria", en la que el proceso que lo está ejecutando no tiene los permisos adecuados:

```
#include <stdio.h>
#include <unistd.h>
#define BUFSIZER1 512
#define BUFSIZER2 ((BUFSIZER1/2) - 8)
int main(int argc, char **argv) {
    char *buffer1;
    char *buffer2;
    char *buffer3;
    char *buffer4;
    buffer1 = (char *) malloc(BUFSIZER1);
    buffer2 = (char *) malloc(BUFSIZER1);
    free(buffer2);

    buffer3 = (char *) malloc(BUFSIZER2);
    buffer4 = (char *) malloc(BUFSIZER2);

    strncpy(buffer2, argv[1], BUFSIZER1-1);
    free(buffer1);
    free(buffer2);
    free(buffer4);
}
```

### 7.2.12 Vulnerabilidad double free

El tercer tipo de vulnerabilidad relacionada con el heap es el double free. Aunque es complejo aprovecharse de este tipo de errores para conseguir ejecutar código, puede constituir un fallo de graves consecuencias. Como su nombre indica, esta condición ocurre cuando se produce la liberación de un puntero más de una vez:

```
int * p = (int*)malloc (SIZE);
...
if (c == 'EOF') {
    free(p);
}
...
free(p);
```

En ocasiones, los desarrolladores cometen el error de liberar la memoria de un objeto más de una vez. Dicha acción puede resultar peligrosa por varias razones; por ejemplo, ¿qué sucede si un bloque de memoria se libera y luego se reasigna esa zona de memoria con otros datos? ¿Qué impide que tal ubicación de memoria contenga datos especialmente diseñados para explotar otros datos o añadir comportamiento malicioso mediante la ejecución de código arbitrario?

También existe una amenaza si la memoria no se reutiliza entre llamadas sucesivas a free(), porque el bloqueo de la memoria podría ingresarse dos veces en la lista de bloqueo libre. Más adelante en el programa, el mismo bloque de memoria podría devolverse dos veces desde una solicitud de asignación, y el programa podría intentar almacenar dos objetos diferentes en la misma ubicación, posiblemente permitiendo la ejecución de código arbitrario.

Cuando audita un código que utiliza asignaciones de memoria dinámicas, ha de realizar un seguimiento de cada ruta a lo largo de la vida útil de una variable, para ver si se ha liberado la memoria con la función free() más de una vez. El siguiente código muestra un ejemplo de una vulnerabilidad de este tipo, donde estamos liberando el puntero data tanto en el caso por default, dentro del switch, como al final del método, antes de realizar el return final:

```
int read_data(int sockfd)
{
    char *data;
    int length;
```

```

length = get_short_from_network(sockfd);
data = (char *)malloc(length+1);
if(!data)
    return 1;
read_string(sockfd, data, length);
switch(get_keyword(data)){
    case USERNAME:
        success = record_username(data);
        break;
    case PASSWORD:
        success = authenticate(data);
        break;
    default:
        error("unknown keyword supplied!\n");
        success = -1;
        free(data);
}
free(data);
return success;
}

```

### 7.2.13 Vulnerabilidad off by one

Las vulnerabilidades off by one tienen su origen en el cálculo o en el uso incorrecto de un valor que realmente es inferior o superior en 1 al valor esperado. Generalmente, dicho tipo de errores se produce por una mala interpretación del programador a la hora de contar o acceder a secuencias de datos; por ejemplo, cuando no se considera el valor de posición 0 en un array o cuando se itera en un bucle más allá del número esperado de veces.

En el siguiente ejemplo, si el usuario envía como parámetro una cadena de longitud igual a 64, la función strcpy añadirá un carácter más de final de cadena '/0', más allá de los límites del array sobrescribiendo, de esta forma, el byte menos significativo. El error principal en este caso radica en la comprobación de la longitud del argumento de la función **check\_param(param)**, al considerar la longitud total del array sin tener en cuenta el byte null del final de cadena:

```

int check_param(char *param){
char buffer[64];
if(strlen(buffer)>64){
print("Parámetro demasiado largo");
}
strcpy(buffer,param);
return 0;
}

int main(int argc,char * argv[]){
if(!argv[1]) return;
if(argv[1])check_param(argv[1]);
}

```

En el siguiente ejemplo, de forma parecida al anterior, la condición del bucle `for (i<=PATH_SIZE)` permitirá escribir un valor más allá del límite del array `filename` (en la asignación `filename[i] = '\0'`), al no considerar que los valores empiezan a contar desde la posición 0. Estableciendo como condición un `i<PATH_SIZE`, evitaremos el **error off by one**:

```
#define PATH_SIZE 60
char filename[PATH_SIZE];
for(i=0; i<=PATH_SIZE; i++) { //se soluciona con i<PATH_SIZE
    char c = getc();
    if (c == 'EOF') {
        filename[i] = '\0';
    }
    filename[i] = getc();
}
```

Este tipo de errores suelen predominar en lenguajes como C y C++, en los que se deja al programador la comprobación de los límites de arrays. Aunque generalmente las consecuencias de este tipo de error acaban en un crash de la aplicación, pueden aprovecharse para ejecutar código o eludir restricciones de seguridad. **Será fundamental, por tanto, controlar los límites del array al hacer asignaciones como las vistas anteriormente.** Esto implica que, si se declara un array de 256 elementos, únicamente estarán disponibles 255 caracteres para la cadena, ya que el último byte es un carácter nulo para indicar el final de la cadena.

#### 7.2.14 Vulnerabilidades race condition

Los errores generados como consecuencia de una condición de carrera se producen por el cambio que experimenta el estado de un recurso (ficheros, memoria, registros, etc.), desde que se comprueba su valor hasta que se utiliza. Tal tipo de errores pueden convertirse en vulnerabilidades serias cuando un atacante influye en el cambio de estado entre la comprobación y su uso. **Generalmente, este tipo de problemas suelen darse bien por la interacción entre hilos en un proceso multihilo, bien por la concurrencia de otros procesos ajenos al proceso vulnerable.**

El **MITRE** clasifica este tipo de vulnerabilidades con el código **CWE 367** y presenta algunos ejemplos prácticos que pueden servirnos para darnos una idea de sus implicaciones: <http://cwe.mitre.org/data/definitions/367.html>

El siguiente ejemplo puede significar un problema serio de seguridad cuando forma parte de un ejecutable con el bit setuid activado. Antes de acceder al fichero a través de la función fopen, se comprueba que el usuario cuenta con permisos suficientes para escribir con el método access(). Si un usuario pudiera cambiar el fichero file después de la llamada a access() —por ejemplo, con un enlace simbólico a otro fichero—, el atacante podría llegar a escribir sobre un fichero del cual no tiene permisos, ya que ambas funciones (access y fopen) trabajan sobre nombres de ficheros, en lugar de manejadores de ficheros:

```
if(!access(file,W_OK)) {
    fich = fopen(file,"W+");
    modificar_fichero(fich);
}
else {
    fprintf(stderr,"Sin permisos");
}
```

Se recomienda la lectura del libro *Secure Coding in C and C++ Race Conditions* <https://dl.acm.org/citation.cfm?id=1407003>, donde se explican las diversas condiciones que pueden desembocar en una race condition, además de mostrar buenas prácticas de programación para mitigar este tipo de problemas. Una de estas medidas, denominada **exclusión mutua**, suele emplearse cuando se utiliza programación concurrente y allí donde es necesario evitar el uso simultáneo de determinados recursos. Una de las técnicas empleadas para llevar a cabo esto reside en deshabilitar las interrupciones durante la ejecución del recurso susceptible de sufrir una condición de carrera. Funciones atómicas como **EnterCriticalSection()** o **pthread\_mutex\_lock()** constituyen algunos ejemplos de funciones que no pueden ser interrumpidas hasta su finalización.

Uno de los temas en los que hay que prestar especial atención es en el uso de enlaces simbólicos, así como en la gestión de ficheros (sobre todo, ficheros temporales) de forma segura. El uso de file locking, y también la correcta identificación de ficheros con funciones como **fstat()**, pueden servir como buena medida de solución para este tipo de problemas.

A diferencia de vulnerabilidades de tipo buffer overflow, las condiciones de carrera suponen un tipo de vulnerabilidad difícil de entender y detectar, y que genera multitud de falsos positivos/negativos en herramientas de análisis está-

tico. Por dicho motivo, deben ser cuidadosamente controladas por primitivas de sincronización y funciones seguras que restrinjan correctamente el acceso a los recursos compartidos.

### 7.2.15 Vulnerabilidad integer overflow

Las vulnerabilidades de tipo integer overflow generalmente ocurren al intentar almacenar un valor demasiado grande en la variable asociada, lo que genera un resultado inesperado (valores negativos o valores inferiores). Este tipo de error puede conllevar consecuencias graves cuando el valor que genera el integer overflow es resultado de alguna entrada de usuario (es decir, que puede ser controlado por él mismo) y, cuando de este valor se toman decisiones de seguridad, se toma como base para hacer asignaciones de memoria, índice de un array, concatenación de datos, reutilización de bucles, etc.

Al igual que cualquier tipo de variable, los números enteros disponen de un límite de tamaño. En este caso, generalmente tal tamaño será el mismo que el utilizado por los punteros de dicha arquitectura; es decir, en el caso de contar con una arquitectura de 32 bits, el entero podrá almacenar un total de  $2^{32} - 1 = 4\,294\,967\,295$  valores.

La vulnerabilidad integer overflow se trata de una variante de desbordamiento de búfer que ocurre cuando se declara una variable con signo entero de ocho bits y se usa para almacenar valores mayores de 127, donde el bit más significativo se usa para indicar el signo (+ o -). Si intentamos almacenar un valor que sea mayor de 127, se produce un desbordamiento de entero, pues el bit 8 está reservado para el signo. El valor 129, por ejemplo, produciría -1. Para evitar tal tipo de desbordamiento de enteros, resulta especialmente recomendable siempre declarar las variables enteras sin signo; por ejemplo, en C, podríamos utilizar **unsigned int myNumber**:

A diferencia de otros tipos de vulnerabilidades, y aunque los desbordamientos de entero resultan difíciles de explotar, al no producir una sobreescritura directa de memoria, en ocasiones, es posible ejecutar código arbitrario. Haciendo una búsqueda en Google con el dork **site:exploit-db.com "integer overflow"**, podemos hacernos una idea de la cantidad de exploits disponibles que hacen uso de dicho tipo de vulnerabilidad, principalmente explotada en navegadores como Firefox, Opera y Chrome.

site:exploit-db.com "integer overflow"



#### Chrome V8 - 'Runtime\_RegExpReplace' Integer Overflow

<https://www.exploit-db.com/exploits/44084> ▶ Traducir esta página

15 feb. 2018 - Chrome V8 - 'Runtime\_RegExpReplace' Integer Overflow... dos exploit for Multiple platform.

[PDF]

#### [Kernel Exploitation] 5: Integer Overflow (/2018/01/kernel-exploitation-5)

[https://www.exploit-db.com/.../43785-\[kernel-exploitation\]-5-inte...](https://www.exploit-db.com/.../43785-[kernel-exploitation]-5-inte...) ▶ Traducir esta página

13 ene. 2018 - Like the comment says, this is a vanilla integer overflow vuln caused by the programmer not considering a very large buffer size being passed ...

#### Opera Web Browser 11.00 - Integer Overflow - Exploit Database

<https://www.exploit-db.com/exploits/16042> ▶ Traducir esta página

25 ene. 2011 - Opera Web Browser 11.00 - Integer Overflow... dos exploit for Windows platform.

#### Chrome V8 - 'PropertyArray' Integer Overflow - Exploit Database

<https://www.exploit-db.com/exploits/44179> ▶ Traducir esta página

Figura 7.7 Búsqueda de exploits para vulnerabilidades integer overflow.

Dicho tipo de vulnerabilidades pueden ser detectadas mediante herramientas de análisis estático, o bien desde un enfoque black box (generalmente, mediante fuzzers) aunque, en este último caso, suele resultar complejo deducir el tipo de vulnerabilidad encontrada. La mejor manera de combatir este tipo de errores es asegurando que los valores de entrada, es decir, aquellos que pueden modificarse por el usuario, se encuentran dentro del rango de valores permitidos.

Se recomienda utilizar bibliotecas o frameworks que tengan un control estricto sobre las operaciones numéricas, así como sobre su almacenamiento, y que permitan prevenir los posibles errores overflow sin generar valores inesperados. Ejemplo de ello es **IntegerLib** para C y C++, librería que proporciona un conjunto de funciones libres de problemas típicos relacionados con enteros, como desbordamiento de enteros y errores de signo. Se puede encontrar más información sobre esta librería en la url <http://doublewise.net/c++/bounded/reference/>.

### 7.2.16 Uso de StackOverflow.

Podemos encontrar portales como CodeProject <https://www.codeproject.com> o StackOverflow <https://es.stackoverflow.com>, donde consultar y ayudar a otros

profesionales en aspectos relacionados con programación y seguridad. Si existen dudas de si cierto código puede ser vulnerable, se necesita optimizar cierto algoritmo o, si se tiene problemas para compilar código, StackOverflow constituye un buen lugar para presentar sus preguntas o ayudar a otros profesionales. Resulta importante ser cuidadoso con el código publicado en este tipo de páginas y la información pública en su perfil. Un atacante podría utilizar dicha información de forma ofensiva; por ejemplo, relacionando código vulnerable con determinado software de su organización.

### 7.3. Análisis dinámico

El análisis dinámico permite analizar el comportamiento de la aplicación en tiempo de ejecución. Desde el punto de vista del atacante, se han de identificar los puntos débiles de la aplicación, con el fin de planificar una estrategia de cómo defendernos de dichos puntos. El análisis dinámico también puede incluir testing a nivel de performance y analizar funcionalidades relacionadas con concurrencia, peticiones de red o caché.

Este análisis consiste en la ejecución de los programas en un entorno controlado, más conocido como sandbox, la cual nos permite monitorizar los accesos a recursos y el envío de información, con el objetivo de identificar su comportamiento.

Es importante analizar el tráfico de red que fluye entre la parte cliente y servidor de la aplicación. El tráfico se puede analizar con herramientas como Wireshark <https://www.wireshark.org>, que permite analizar los paquetes que se intercambian, o bien mediante herramientas que actúan a modo de proxy, como Burp Proxy <https://portswigger.net/burp/communitydownload>.

También cabe analizar credenciales, tokens de autenticación o claves que se están enviando sobre un canal no seguro, como http.

Tales herramientas de prueba permiten a los analistas de seguridad entender el comportamiento de los sistemas en ejecución, para que puedan identificar posibles problemas potenciales. Las herramientas que actúan a modo de proxy permiten a los analistas de seguridad observar y cambiar la comunicación entre el cliente de la aplicación y los servicios que ofrece.

Entre las ventajas del análisis dinámico, podemos destacar:

- Detecta vulnerabilidades en tiempo de ejecución.
- Permite llevar a cabo análisis de aplicaciones de las cuales no disponemos de su código fuente (enfoque black box).

- Permite identificar vulnerabilidades, que pueden haber sido reportadas como falsos negativos durante el análisis estático.

Entre las **desventajas** del análisis dinámico, podemos destacar:

- No existen herramientas automatizadas para todos los lenguajes ni plataformas.
- Se generan falsos negativos y falsos positivos.
- Resulta difícil localizar cierto tipo de vulnerabilidades sin disponer del código fuente.
- Resulta difícil reproducir todas las posibles vías de ejecución.

### 7.3.1 Análisis dinámico en C/C++ con Valgrind

Esta sección tendrá por objetivo mostrar algunos de los errores más comunes que todavía hoy día siguen generando serios problemas de seguridad: heap overflow, buffer overflow, use after free, off by one, format string, integer overflows, race conditions, memory leaks, etc. La idea estriba en explicar, de forma práctica, cómo se producen dichos errores, qué implicaciones pueden tener y algunos consejos para evitar tal tipo de vulnerabilidades.

Una de las herramientas más destacadas para realizar análisis dinámico en C++ es la suite Valgrind (<http://valgrind.org>). Dicha suite nos proporciona un conjunto de herramientas de debugging dirigidas a detectar posibles errores de memoria en lenguajes como C y C++, y que pueden originar importantes vulnerabilidades de seguridad.

Actualmente se encuentra disponible para la mayoría de las distribuciones Linux y una de las ventajas principales de la herramienta reside en que los ejecutables analizados no necesitan ser compilados de forma especial para analizarse en tiempo de ejecución. A tal fin, utiliza una máquina virtual (VEX), con la que podrá interceptar los accesos a memoria, así como algunas llamadas al sistema (syscalls).

Valgrind producirá cierto delay durante la depuración de errores, y será más eficiente en arquitecturas de 64 bits. La funcionalidad más interesante para nosotros será memcheck, gracias a la cual podremos detectar errores relacionados con el stack y el heap: uso de memoria no inicializada, liberaciones de memoria fuera del acceso permitido, escritura y lectura en direcciones de memoria no válidas, memory leaks y syscalls con parámetros ilegales.

En lenguajes como C o C++, que carecen de un recolector de basura, pueden producirse problemas serios, si no se hace un uso eficiente de la memoria. Esto es, precisamente, lo que comprueba Valgrind: que existe una correspondencia entre el número de mallocs y frees que realizamos en el código. En el ejemplo, se reservan 240 bytes mediante la función malloc, pero únicamente se liberarán estos si el número de argumentos suministrados es superior a dos. Si ejecutamos Valgrind como se muestra a continuación, observamos que este nos alerta de que existe un único malloc y cero frees:

#### memory\_leaks.c

```
int main(int argc,char * argv[])
{
    int *ptr;
    ptr =malloc(60*sizeof(ptr))
    if(argc>2)
        free(ptr)
    return 0;
}
```

En este ejemplo vemos cómo Valgrind nos informa de la presencia de un **memory leak**. Tal tipo de problemas se presentan cuando se asigna memoria dinámicamente, pero que no es bien liberada. A continuación se muestra la salida de ejecución sobre el fichero fuente **memory\_leaks.c**, después de generar el código objeto y el ejecutable:

```
$ valgrind --tool=memcheck --leak-check=yes ./memory-leaks
MemCheck, a memory error detector
Command: ./memory_leaks
HEAP SUMMARY:
    in use at exit:240 bytes in 1 blocks
    total heap usage:1 allocs, 0 frees, 240 bytes allocated

240 bytes in 1 blocks are definitely lost in loss record 1 of 1
    at 0x4024F20: malloc(vg_replace_malloc.c:236)
    by 0x8048428:main (memory_leaks.c:6)

LEAK SUMMARY:
    definitely lost: 240 bytes in 1 blocks
    indirectly lost: 0 bytes in 0 blocks
    possibly lost: 0 bytes in 0 blocks
    still reachable: 0 bytes in 0 blocks
    suppressed: 0 bytes in 0 blocks
```

Si compilamos el ejecutable con la opción `-g` (`gcc -g`) para añadir símbolos de depuración, Valgrind mostrará también la linea dentro de nuestro código donde se reservó dicha memoria mediante `malloc`.

Veamos otro ejemplo; en este caso, el código presenta múltiples errores: por un lado, si el número de argumentos pasados al ejecutable resulta diferente a 1, se imprimirá el número de bytes que se escribirán en `stdout` desde el `printf`; sin embargo, la variable `times` no ha sido inicializada.

**Memcheck** permite detectar el uso de variables no inicializadas, siempre y cuando esto pueda generar algún tipo de comportamiento que afecte al programa. En este ejemplo, la variable `times` no ha sido inicializada; se muestra la traza de la variable `times`, donde memcheck expone una advertencia, al comprobar que los datos presentes en él contienen valores no inicializados:

#### `write_memory.c`

```
int main(int argc,char * argv[]){
    int times;
    char *memory = malloc(50);
    if(argc==1){
        print("Introduzca parámetro");
    }else{
        print("Escribiendo en stdout %d bytes",times);
        (void)write(1,memory,strlen(argv[1]));
        free(memory);
    }
    free(memory);
    return 0;
}
```

Por otro lado, memcheck llevará a cabo algunas comprobaciones, siempre que se haga uso de ciertas *syscalls*, por ejemplo, que todos los parámetros hayan sido inicializados o que los arrays desde los que se va a leer o escribir se encuentren en direcciones válidas de memoria y contengan valores inicializados. El motivo por el que memcheck se queja sobre la syscall `write` es porque el puntero `memory` carece de valores inicializados que pueden generar problemas a la hora de la escritura en `stdout`.

También informa de la linea en la que se produjo la reserva de memoria dinámica de dicho array (línea 8). Por último, memcheck avisa de un invalid free en la línea 11, como consecuencia de un **double free** del puntero `memory`.

Cuando el número de argumentos es diferente a 0, se producirá un free dentro del else; sin embargo, antes del return, también se produce un free, el cual puede acarrear problemas de acceso a memoria.

Memcheck lleva la cuenta del número de bloques asignados con funciones como malloc o new, por lo que permite detectar si el argumento pasado a free o delete es legítimo (es decir, que apunta al comienzo de un bloque de memoria previamente reservado en el heap) o no.

Además de dichas funcionalidades, memcheck permite detectar si el uso de funciones de liberación de memoria dinámica (free o delete) se muestra acorde con la función con la que se asignó dicha memoria (malloc, calloc, realloc, valloc, etc.) o si existen ciertos problemas de solapamiento de bloques de memoria a la hora de utilizar funciones como memcpy, strcpy, strncpy, strcat o strncat.

Memcheck también soporta múltiples opciones que pueden ayudarnos a localizar rápidamente la raíz de muchos problemas; por ejemplo, si ejecutamos el mismo programa añadiendo la opción **-track-origins=yes**, memcheck mostraría la dirección en la pila en donde se encuentra la variable que no fue inicializada, así como la linea en nuestro código fuente:

```
$ valgrind --tool=memcheck --leak-check=yes --track-origins=yes ./write_memory
MemCheck, a memory error detector
Command: ./ write_memory
Use of uninitialized value of size 4:
  at 0x4077256: _itoa_word(_itoa_c:195)
  by 0x407AAE1: vfprintf (vfprintf.c:1613)
  by 0x408215F: printf (printf.c:35)
  by 0x80484F4: main (write_memory.c:11)
Uninitialized value was created by a stack allocation
  at 0x80484A: main (write_memory.c:8)
```

Se recomienda leer la documentación de <http://valgrind.org/docs/manual/mc-manual.html> sobre el uso de esta herramienta para testear aplicaciones, con el objetivo de detectar problemas relacionados con la reserva y liberación de memoria dinámica.

## 7.4 Herramientas de análisis

Los siguientes repositorios ofrecen una lista de herramientas que podrían utilizarse tanto para análisis estático como dinámico, y que permiten descubrir vulnerabilidades de seguridad y realizar revisiones de código:

- <https://oss-security.openwall.org/wiki/tools>
- [https://samate.nist.gov/index.php/Source\\_Code\\_Security\\_Analyzers.html](https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html)

#### Static Analysis

##### **Clang**

Compiler with static analysis capabilities for C/C++.

• <http://clang-analyzer.llvm.org/>

• <http://clang.llvm.org/>

Clang and very recent GCC also have dynamic "sanitizers". For example:

• <http://clang.llvm.org/docs/AddressSanitizer.html>

• <http://clang.llvm.org/docs/ThreadSanitizer.html>

• <http://clang.llvm.org/docs/MemorySanitizer.html>

##### **Coccinelle**

A tool for matching and fixing source code for C, C++, and other languages.

• <http://coccinelle.lip6.fr/>

##### **Coverity**

Provides static analysis tools for C, C++, and other languages (requires license).

• <http://www.coverity.com/>

Figura 7.8 Herramientas de análisis estático.

#### Dynamic Analysis

##### **Valgrind**

Detect many memory management and threading bugs, and profile your programs in detail.

• <http://valgrind.org/>

##### **KEDR**

Provides runtime analysis of Linux kernel modules including device drivers, file system modules, etc.

• <http://kedr.sberbank.ru/>

##### **kmemcheck, kmemleak**

Linux kernel debugging features for detecting memory issues.

• <http://lxr.free-electrons.com/v4.18.10/documentation/kmemcheck.txt> • <http://lxr.free-electrons.com/v4.18.10/documentation/kmemleak.txt>

Figura 7.9 Herramientas de análisis dinámico.

**GitLab** [https://docs.gitlab.com/ee/user/application\\_security](https://docs.gitlab.com/ee/user/application_security) también soporta tanto **análisis estático como dinámico** e incluye, automáticamente, un amplio análisis de seguridad cada vez que se realiza un commit o un pull request, incluidas las pruebas de seguridad a nivel estático y dinámico, junto con el análisis de dependencias y el análisis de contenidos:

- **Análisis estático:**
  - [https://docs.gitlab.com/ee/user/application\\_security/sast/index.html](https://docs.gitlab.com/ee/user/application_security/sast/index.html)
- **Análisis dinámico:**
  - [https://docs.gitlab.com/ee/user/project/merge\\_requests/dast.html](https://docs.gitlab.com/ee/user/project/merge_requests/dast.html)

- **Análisis de dependencias:**

- [https://docs.gitlab.com/ee/user/project/merge\\_requests/dependency\\_scanning.html](https://docs.gitlab.com/ee/user/project/merge_requests/dependency_scanning.html)

- **Análisis de contenedores:**

- [https://docs.gitlab.com/ee/user/project/merge\\_requests/container\\_scanning.html](https://docs.gitlab.com/ee/user/project/merge_requests/container_scanning.html)

Las pruebas de caja blanca (SAST) ahorran tiempo de desarrollo y costes al identificar vulnerabilidades durante el desarrollo. De esta forma, los desarrolladores pueden dedicar tiempo al desarrollo e innovación, en vez de corregir errores de aplicaciones desplegadas en producción.

Si está utilizando GitLab como aplicación de integración y despliegue continuo CI/CD, puede analizar su código fuente en busca de vulnerabilidades conocidas. GitLab realiza una comparación entre las ramas de origen y destino y muestra la información directamente al hacer el merge entre las dos ramas.



Figura 7.10 Análisis de dependencias en GitLab.

Entre los casos que puede detectar, destacan:

- Código inseguro, que puede llevar a la ejecución arbitraria de código malicioso.
- Su aplicación se muestra vulnerable a un ataque de cross-site scripting (XSS), que permitiría acceder a las cookies de sesión del usuario.

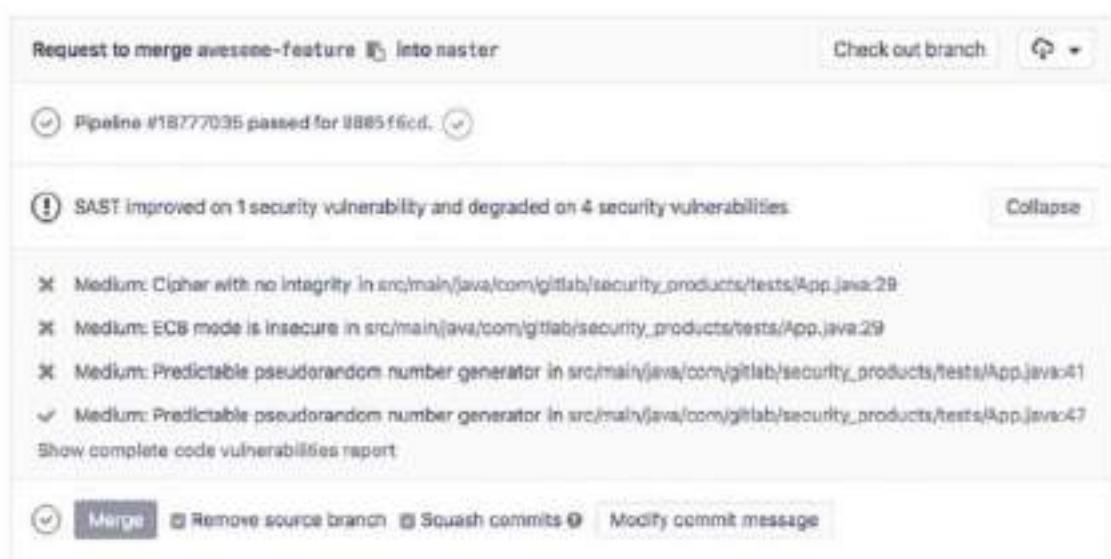


Figura 7.11 Análisis estático en GitLab.

El **Security Dashboard** de GitLab muestra los últimos informes de seguridad para un determinado proyecto y constituye un buen sitio para obtener una visión global de todas las vulnerabilidades de seguridad agrupadas por nivel de criticidad.

A screenshot of the GitLab Security Dashboard. The left sidebar has sections for "Project", "Issues" (0), "Merge Requests" (0), and "Scanning". The main area shows a summary: "Pipeline #18 triggered 6 days ago by John Doe" and "Last run: 24h30m". Below that is a "Vulnerabilities" section with filters for "Severity" (All severities), "Confidence" (All confidence levels), and "Report type" (All report types). A chart shows the count of vulnerabilities by severity: Critical (0), High (0), Medium (4), and Low (0). A table lists three vulnerabilities: "Cipher with no integrity" (Medium, High confidence) and "ECB mode is insecure" (Medium, High confidence) under "Operations", and "Predictable pseudorandom number generator" (Medium, Medium confidence) under "Languages".

Figura 7.12 Lenguajes y herramientas de scanning soportadas por GitLab.

Language (package managers) / framework	Scan tool	introduced in GitLab Version
.NET	Security Code Scan <a href="#">🔗</a>	11.0
Any	GitLeaks <a href="#">🔗</a> and TruffleHog <a href="#">🔗</a>	11.9
C/C++	Flawfinder <a href="#">🔗</a>	10.7
Elixir (Phoenix)	Sobelow <a href="#">🔗</a>	11.10
Go	Gosec <a href="#">🔗</a>	10.7
Groovy (Ant <a href="#">🔗</a> , Gradle <a href="#">🔗</a> , Maven <a href="#">🔗</a> and SBT <a href="#">🔗</a> )	SpotBugs <a href="#">🔗</a> with the find-sec-bugs <a href="#">🔗</a> plugin	11.3 (Gradle) & 11.9 (Ant, Maven, SBT)
Java (Ant <a href="#">🔗</a> , Gradle <a href="#">🔗</a> , Maven <a href="#">🔗</a> and SBT <a href="#">🔗</a> )	SpotBugs <a href="#">🔗</a> with the find-sec-bugs <a href="#">🔗</a> plugin	10.6 (Maven), 10.8 (Gradle) & 11.9 (Ant, SBT)
Javascript	ESLint security plugin <a href="#">🔗</a>	11.8
Nodejs	NodejsScan <a href="#">🔗</a>	11.1
PHP	phpcs-security-audit <a href="#">🔗</a>	10.8
Python (pip <a href="#">🔗</a> )	bandit <a href="#">🔗</a>	10.3

Figura 7.13 Lenguajes y herramientas de scanning soportadas por GitLab.

También se pueden obtener más detalles acerca de una vulnerabilidad, ver de qué proyecto proviene, el archivo en el que se encuentra y diversos metadatos para ayudar a medir el riesgo.

## **Capítulo 8. Metodologías de desarrollo**

La manera en que se desarrolla software ha evolucionado desde la forma de code and fix (desarrollar y después arreglar el bug), en la que un equipo de desarrollo concibe la idea general de lo que quiere desarrollar, pasando a metodologías formales (RUP o proceso unificado, entre otras) en las que existe una planificación detallada del desarrollo desde las primeras etapas de toma de requerimientos, documentación y diseño de alto nivel.

Algunas de las metodologías tienden a enfocarse en mejorar la calidad en el software, reducir el número de defectos y cumplir con la funcionalidad especificada, pero, en la actualidad, también se ha de entregar un producto que garantice tener cierto nivel de seguridad.

Podemos igualmente encontrar metodologías que contemplan, durante su proceso, un conjunto de actividades específicas para eliminar vulnerabilidades detectadas en el diseño o en el código, o la realización de pruebas que aportan datos para la evaluación del estado de seguridad, entre otras actividades relacionadas para mejorar la seguridad del software.

### **8.1. Metodologías de desarrollo de software seguro**

La seguridad ha pasado de ser un requerimiento no funcional, que podría implementarse como parte de la calidad del software, a un elemento primordial de cualquier aplicación. Para hacer frente a hackers y expertos en explotar vulnerabilidades de las aplicaciones, se necesita la implementación de metodologías que contemplen en su proceso de desarrollo de software la inclusión de la seguridad como un elemento básico en la arquitectura de cualquier aplicación o producto de software.

La clave de un software seguro es el proceso de desarrollo utilizado. Muchos de los defectos relacionados con la seguridad en software se pueden evitar si los desarrolladores conocieran mejor las técnicas que los hackers utilizan para reconocer las implicaciones de su diseño y las posibilidades de implementación desde el punto de vista de la seguridad, así como los riesgos que originan dicha implementación.

Una metodología de desarrollo de software, que incorpore mejoras en el sentido de incluir la seguridad en el ciclo de desarrollo, debe proporcionar un marco integrado que permita promover la seguridad del software a lo largo de

todo el ciclo de vida de desarrollo. El objetivo de este capítulo radica en presentar algunas de las metodologías de desarrollo que permiten producir productos de software seguros.

Existen varias metodologías que establecen una serie de pasos en búsqueda de un software más seguro y capaz de resistir ataques. Entre ellas se encuentran **Correctness by Construction (CbyC)**, **Security Development Lifecycle (SDLC)** y **Lightweight Application Security Process (CLASP)**. En este capítulo analizamos las características de las dos primeras, detallamos las fases que las conforman y destacamos sus particularidades.

### 8.1.1 Correctness by Construction (CbyC)

Representa un método efectivo para desarrollar software que demanda un nivel de seguridad crítico. Algunas de las empresas que han utilizado dicha metodología han producido software con un porcentaje de defectos por debajo de los 0.05 defectos por cada 1000 líneas de código, y con una productividad de 30 líneas de código por persona al día.

Los objetivos principales de esta metodología residen en obtener una traza de defectos al mínimo y una alta resiliencia al cambio, los cuales se logran siguiendo como principios fundamentales que resulte muy difícil introducir errores y asegurarse de que estos se eliminan tan pronto hayan sido detectados. CbyC busca desarrollar un producto que, desde el inicio, sea correcto, con requerimientos estrictos de seguridad y con una definición muy detallada del comportamiento del sistema.

CbyC combina los métodos formales con el desarrollo iterativo, y utiliza un enfoque basado en ciclos de entrega continuos y de forma incremental que permita mostrar avances para recibir retroalimentación y valoración del producto. En la figura 8.1 se muestran las fases propuestas por esta metodología para el desarrollo de software.



Figura 8.1 Fases de la metodología Correctness by Construction.

- **Fase de requerimientos.** En la fase de requerimientos se especifican el propósito, las funciones y los requerimientos no funcionales. Se escriben los requerimientos de usuario con sus respectivos diagramas contextuales, diagramas de clase y definiciones operativas. Cada requerimiento debe pasar por un proceso de trazabilidad.
- **Fase de diseño de alto nivel.** Se describe la estructura interna del sistema, la distribución de la funcionalidad, la estructura de las bases de datos y los mecanismos para las transacciones y comunicaciones, así como la manera en que los componentes se comunican a nivel de seguridad. Se utilizan los métodos formales para definir el diseño de alto nivel y obtener una descripción, un comportamiento y un modelo preciso. Los métodos formales han probado ser exitosos para especificar y probar el nivel de correctness y en la consistencia interna de las especificaciones de funciones de seguridad.
- **Fase de especificación del software.** En esta fase se documentan las especificaciones de la interfaz de usuario y las especificaciones formales de los niveles superiores, así como se desarrolla un prototipo para su validación.
- **Fase de diseño detallado.** Define el conjunto de módulos y procesos, junto con su funcionalidad.
- **Fase de especificación de los módulos.** Se definen el estado y el comportamiento esperados de cada módulo del software, teniendo en cuenta el flujo de información. Los módulos deberían seguir el enfoque de bajo acoplamiento y alta cohesión.
- **Fase de codificación.** La fase de codificación se debería complementar con pruebas de análisis estático del código y, en caso de ser necesario, se realizará una revisión al código.
- **Fase de las especificaciones de las pruebas.** Para obtener las especificaciones de las pruebas, se toman en cuenta las especificaciones del software, los requerimientos y el diseño de alto nivel. Se efectúan pruebas de valores límites, pruebas de comportamiento y pruebas para los requerimientos no funcionales. Todas las pruebas, en este punto, van encaminadas a nivel del sistema y se orientan a las especificaciones.
- **Fase de construcción del software.** CbyC utiliza el desarrollo de tipo ágil desde la primera entrega, donde se obtiene el esqueleto completo del sistema con todas las interfaces y los mecanismos de comunicación, con una funcionalidad limitada, que se irá incrementando en cada iteración del ciclo.

### 8.1.2 Security Development Lifecycle (SDLC)

Históricamente, las vulnerabilidades en el software han sido muy comunes y han acabado otorgando mala fama a organizaciones por su proceso de creación de aplicaciones. La conclusión a la que podía llegarse es que la mayoría de las vulnerabilidades se pueden solucionar en la fase de desarrollo de los procesos de desarrollo de aplicaciones. Las vulnerabilidades constituyan errores en la manera de llevar a cabo el desarrollo aunque, en algunas ocasiones, no se trataba de errores de fase de implementación.

Con la metodología Systems Development Life Cycle, es posible desarrollar software de calidad desde el primer instante del propio proyecto y se dispondrá de iteraciones que deben ser ejecutadas durante la vida del propio software. Obtener un 100 % de seguridad resulta imposible, pero estamos poniendo los cimientos para reducir notablemente el número de vulnerabilidades que pueden llegar a producción en un proceso como este.

Uno de los mejores métodos para evitar que aparezcan errores de seguridad en las aplicaciones de producción reside en mejorar el ciclo de vida de desarrollo de software (SDLC), al incluir seguridad en cada una de sus fases. Un SDLC es una estructura impuesta en el desarrollo de artefactos de software. En la figura 8.2 se muestra un modelo de SDLC genérico.

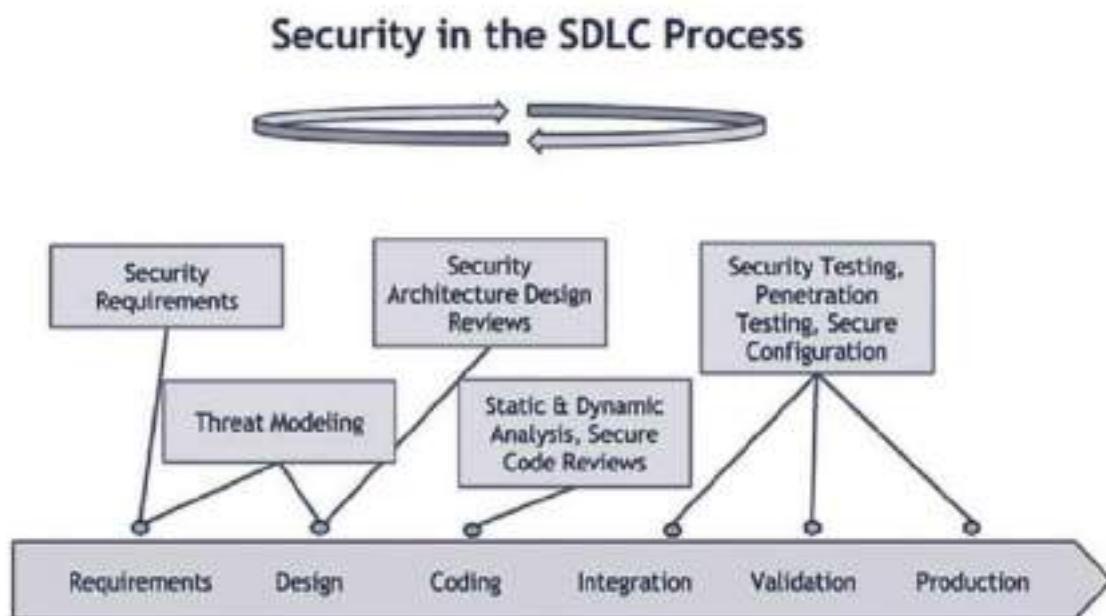


Figura 8.2 Fases de la metodología SDLC.

Security Development Lifecycle (SDLC) es un proceso para mejorar la seguridad de software propuesto por Microsoft, formado por 7 fases y 16 actividades enfocadas a mejorar la seguridad del desarrollo de un producto de software.

Las prácticas que propone SDLC van desde una etapa de entrenamiento sobre temas de seguridad, pasando por análisis estático, análisis dinámico y fuzz testing del código, hasta conseguir un plan de respuesta a incidentes. Una de las características principales de SDLC reside en el modelado de amenazas, que sirve a los desarrolladores para encontrar partes del código donde, probablemente, existan vulnerabilidades o las cuales sean susceptibles de sufrir ataques.

### 8.1.3 Fases de la metodología SDLC

El ciclo de vida de desarrollo de software seguro (Software Development Life Cycle) ayuda a las organizaciones a garantizar la seguridad de sus aplicaciones mediante la implantación de una serie de medidas, entre las que podemos destacar:

- Evaluación de las prácticas de seguridad de software existentes en las organizaciones.
- Construcción de un programa de aseguramiento de la seguridad de software equilibrado en iteraciones bien definidas.
- Demostración de mejoras concretas al programa de aseguramiento de la seguridad.
- Definición y medición de las actividades relativas a seguridad de una organización.

Existen dos versiones del SDLC: la versión que utiliza metodologías tradicionales y la orientada al desarrollo ágil. La primera resulta más apropiada para equipos de desarrollo y proyectos más grandes, que no sean susceptibles a cambios durante el proceso. La segunda desarrolla el producto de manera incremental y en la frecuencia de la ejecución de las actividades para el aseguramiento de la seguridad; por ejemplo, la versión ágil sería recomendable utilizarla para desarrollos de aplicaciones web.

En la figura 8.3, se encuentra el flujo de las fases en la metodología de SDLC, donde podemos resaltar la existencia de una etapa previa a los requerimientos llamada **Entrenamiento**, enfocada al entrenamiento y a la formación en seguridad, y la última etapa del proceso, llamada **Respuesta**, encargada de realizar un seguimiento, en el caso de algún incidente de seguridad en la aplicación.



Figura 8.3 Flujo de fases de la metodología SDLC.

Las 7 fases y 16 actividades que componen dicha metodología se pueden resumir en la tabla de la figura 8.4.

Actividades del SDL para la Seguridad	
1. Entrenamiento	5. Verificación
Entrenamiento de seguridad básica	Ánalisis dinámico
2. Requerimientos	Fuzz Testing
Establecer requerimientos de seguridad	Revisión de la superficie de ataques
Crear umbrales de calidad y límites de errores	6. Lanzamiento
Evaluación de los riesgos de seguridad y privacidad	Plan de respuesta a incidentes
3. Diseño	Revisión de seguridad final
Establecer requerimientos de diseño	Aprobar y archivar lanzamiento
Ánalisis de la superficie de ataques	7. Respuesta
Modelado de amenazas	Ejecutar el plan de respuesta a incidentes
4. Implementación	
Utilizar herramientas aprobadas	
Prohibir funciones no seguras	
Ánalisis estático	

Figura 8.4 Fases y actividades de la metodología SDLC.

## Fase de entrenamiento

Todos los miembros de un equipo de desarrollo de software deberían recibir una formación apropiada, con el fin de mantenerse actualizados acerca de los conceptos básicos y las últimas tendencias en el ámbito de la seguridad y privacidad. Las personas con roles técnicos (desarrolladores, evaluadores y administradores de programas) directamente implicadas en el desarrollo de programas de software deben asistir, como mínimo, una vez al año a un curso de formación en materia de seguridad en conceptos fundamentales como defensa en profundidad, principio de privilegios mínimos, inyección de código SQL, métodos criptográficos, modelos de riesgos, evaluación de riesgos y procedimientos de desarrollo de privacidad.

## Fase de requerimientos

En la fase de requerimientos, el equipo de desarrollo es asistido por un consultor de seguridad para revisar los planes y hacer recomendaciones para cumplir con los objetivos de seguridad, en concordancia con el tamaño del proyecto, la complejidad y los riesgos asociados. Dicho equipo identifica cómo será integrada la seguridad en el proceso de desarrollo, identificará los objetivos clave de seguridad y la manera en que se integrará el software en conjunto con otras aplicaciones o a nivel de arquitectura.

En esta fase se deben identificar aquellos requerimientos funcionales que tendrán impacto en los aspectos de seguridad de la aplicación. Algunos de ellos son requerimientos de compliance con normativas locales o internacionales, tipo de información que se transmitirá y requerimientos de registros de auditoría.

## Fase de diseño

Se identifican los requerimientos y la estructura del software. Se define una arquitectura segura y guías de diseño que identifiquen los componentes críticos para la seguridad, y se aplica el enfoque de mínimo privilegio y la reducción del área de ataques. Durante la fase de diseño, el equipo de desarrollo realiza un modelado de amenazas a nivel de componente, y detecta los activos que son administrados por el software y las interfaces por las cuales se puede acceder a esos activos.

Antes de comenzar a escribir líneas de código, existen numerosos aspectos de seguridad que deben ser tenidos en cuenta durante el diseño de la aplicación, entre los que podemos destacar:

- **Diseño de autorización.** Permite definir los roles, permisos y privilegios de la aplicación.
- **Diseño de autenticación.** Se debe diseñar el modo en el que los usuarios se van a autenticar, contemplando aspectos tales como los mecanismos o factores de autenticación con contraseñas, tokens, certificados, etc.; posibilidades de integrar la autenticación con servicios externos como LDAP, Radius o Active Directory, y los mecanismos que tendrá la aplicación para evitar ataques de diccionario o de fuerza bruta.
- **Diseño de los mecanismos de protección de datos.** Se debe contemplar el modo en el que se protegerá la información más sensible en tránsito o almacenada.

Una vez que se cuenta con el diseño detallado de la aplicación, una práctica interesante es la de realizar sobre este un análisis de riesgo orientado a software. Existen técnicas documentadas al respecto tales como **Threat Modeling**. Dichas técnicas permiten definir un modelo de amenazas y un marco para identificar debilidades de seguridad en el software, antes de la etapa de codificación. Como valor agregado, en el análisis de riesgo orientado a software se pueden obtener casos de prueba para ser utilizados en la etapa de Testing/QA.

## Fase de implementación

Durante la fase de implementación, se codifica, prueba e integra el software. Los resultados del modelado de amenazas sirven de guía a los desarrolladores para generar el código que mitigue las amenazas de alta prioridad. La codificación se lleva a cabo siguiendo los estándares de cada lenguaje desde el punto de vista de seguir buenas prácticas de desarrollo seguro. Las pruebas se centran en detectar vulnerabilidades y se pueden aplicar técnicas de fuzz testing y análisis de código estático por medio de herramientas como SonarQube.

En esta fase es donde deberíamos detectar distintos tipos de vulnerabilidades. Tales vulnerabilidades podríamos dividirlas en dos grandes grupos: vulnerabilidades clásicas y vulnerabilidades funcionales.

En el primer grupo entrarían las vulnerabilidades que podemos encontrar en el **"OWASP top 10"** (vulnerabilidades de inyección de SQL o cross-site scripting), así como se detecta también la presencia de otras vulnerabilidades no ligadas directamente con las aplicaciones web, como desbordamiento de búfer o denegación de servicio. Los frameworks de desarrollo de aplicaciones constituyen una buena ayuda en este punto, ya que hacen de intermediarios entre el programador y el código, y permiten prevenir la mayoría de las vulnerabilidades conocidas. Ejemplos de estos frameworks son Spring para Java, Ruby on Rails para Ruby y Django para Python.

Las vulnerabilidades funcionales son aquellas ligadas específicamente a la funcionalidad de negocio que posee la aplicación, por lo que no se encuentran previamente categorizadas. Algunos ejemplos ilustrativos de este tipo de vulnerabilidad son los siguientes: una aplicación de banca electrónica que permite realizar transferencias con valores negativos, un sistema de subastas que permite ver los valores de otros usuarios o un sistema de venta de entradas que no impone límites adecuados a la cantidad de reservas que un usuario puede hacer.

## Fase de verificación

En esta fase, el software ya es funcional en su totalidad y se encuentra en la fase beta de prueba. La seguridad se halla sujeta a una revisión más exhaustiva del código y a ejecutar pruebas específicas que permitan identificar superficies de ataque no identificadas en la fase de implementación.

## Fase de lanzamiento

En la fase de lanzamiento, el software se encuentra sujeto a una revisión de seguridad final, durante un periodo de dos a seis meses previos a la entrega final, con el objetivo de conocer el nivel de seguridad del producto y la probabilidad de soportar ataques en un entorno productivo. En el caso de encontrar vulnerabilidades que pongan en riesgo la seguridad, se regresa a la fase previa, para solucionar esos errores.

## Fase de respuesta

Sin importar la cantidad de revisiones que se haga al código o las pruebas en búsqueda de vulnerabilidades, teniendo en cuenta que resulta casi imposible entregar un software 100 % seguro, se ha de estar preparado para responder a incidentes de seguridad que, seguramente, se darán cuando la aplicación se encuentre en producción y probada por usuarios finales.

### 8.1.4 Vulnerabilidades en SDLC

En general, puede realizar una auditoría en cualquier etapa del ciclo de vida del desarrollo de sistemas (SDLC). Sin embargo, el coste de identificar y corregir vulnerabilidades puede variar ampliamente, según cuándo y cómo elija la auditoría. Es importante conocer las diferentes fases del SDLC antes de comenzar un proceso de auditoría:

1. **Estudio de viabilidad.** Esta fase se ocupa de identificar las necesidades que ha de cumplir el proyecto y determinar si el desarrollo de la solución resulta tecnológicamente y económicamente viable.
2. **Definición de requisitos.** En esta fase se realiza un estudio más profundo de los requisitos para el proyecto y se establecen los objetivos del proyecto.

3. **Diseño.** La solución ha sido diseñada y se toman decisiones sobre cómo el sistema logrará técnicamente los requisitos acordados.
4. **Implementación.** El código de la aplicación se desarrolla de acuerdo con el diseño establecido en la fase anterior.
5. **Integración y pruebas.** La solución se somete a un cierto nivel de control de calidad, para garantizar que funciona como se espera y para detectar cualquier error en el software.
6. **Operación y mantenimiento.** La solución está implementada y las revisiones, actualizaciones y correcciones se realizan como resultado de los comentarios de los usuarios.

Cada proceso de desarrollo de software sigue este modelo hasta cierto punto. Los modelos de cascada clásicos tienden hacia una interpretación estricta, en la que la vida útil del sistema pasa por una sola iteración a través del modelo. En contraste, las metodologías ágiles tienden a centrarse en refinar una aplicación al pasar por repetidas iteraciones de las diferentes fases del ciclo de vida de desarrollo. Por lo tanto, la forma en que se aplica el modelo SDLC puede variar, pero los conceptos y las diferentes fases no deberían ser muy distintos.

Al definir las diversas clases de vulnerabilidades, podemos diferenciarlas en función de las fases del ciclo de desarrollo SDLC. Por una parte, tenemos **vulnerabilidades de diseño** (fases 1, 2 y 3 de SDLC); por otra parte, tenemos **las vulnerabilidades de implementación** (fases 4 y 5 de SDLC) y, por último, **las vulnerabilidades operacionales** (fase 6 de SDLC). La comunidad de seguridad generalmente acepta vulnerabilidades de diseño como errores en la arquitectura y las especificaciones de un sistema de software. Las vulnerabilidades de implementación son errores técnicos de bajo nivel en la construcción de un sistema de software. La categoría de vulnerabilidades operativas aborda los errores que surgen en la implementación y configuración del software en un entorno en particular.

### Vulnerabilidades de diseño

Una vulnerabilidad de diseño representa un problema que surge de un error fundamental o de una supervisión en el diseño del software. Estos tipos de errores normalmente ocurren debido a suposiciones sobre el entorno en el que se ejecutará un programa o el riesgo de exposición al cual los componentes de la aplicación se enfrentarán en un entorno de producción.

El diseño de un sistema de software se basa en la definición de los requisitos del software, que son una lista de objetivos que un sistema de software debe cumplir. Normalmente, un ingeniero toma el conjunto de requisitos y construye especificaciones de diseño, que se centran en cómo crear el software que cumpla con esos objetivos.

Las especificaciones constituyen los planes de cómo se debe construir el programa para cumplir con los requisitos establecidos. Por lo general, incluyen una descripción de los diferentes componentes de un sistema de software, información sobre cómo se implementarán los componentes y qué harán, e información sobre cómo interactúan. Las especificaciones pueden incluir diagramas de arquitectura, diagramas lógicos, diagramas de flujo de procesos, especificaciones de interfaz y protocolo, jerarquías de clase y otras especificaciones técnicas.

Cuando generalmente se produce un error debido al diseño, no suele distinguirse entre un problema con los requisitos del software y un problema con las especificaciones del software. Hacer tal distinción no resulta fácil, porque muchos problemas de alto nivel podrían explicarse como una supervisión de los requisitos o un error en las especificaciones.

### **Vulnerabilidades de implementación**

En una vulnerabilidad de implementación, el código generalmente hace lo que debe, pero surge un problema de seguridad en la forma en que se realiza la operación. Dichos problemas ocurren durante la fase de implementación del SDLC, pero, a menudo, se transfieren a la fase de integración y pruebas. Estos problemas pueden ocurrir si la implementación se desvía del diseño para resolver discrepancias técnicas. Sin embargo, en su mayoría, las situaciones explotables se originan por artefactos técnicos y el entorno en el que se construye el software. Las vulnerabilidades de implementación también se conocen como "errores de bajo nivel" o "errores de carácter técnico".

### **Vulnerabilidades operacionales u operativas**

Las vulnerabilidades operacionales u operativas son problemas de seguridad que surgen a través de los procedimientos operativos y el uso general de un componente de software en un entorno específico. Una forma de distinguir dichas vulnerabilidades se encuentra en que no están presentes en el código fuente del software que se está considerando; más bien tienen relación con el

software interactúa con su entorno. Específicamente, pueden incluir problemas con la configuración del software en su entorno y problemas causados por procesos manuales y automatizados que forman parte del sistema. Las vulnerabilidades operativas pueden incluso contener determinados tipos de ataques a los usuarios del sistema, como la ingeniería social y el robo de credenciales. Estos problemas ocurren en la fase de operación y mantenimiento de SDLC, aunque presentan algunos solapamientos en la fase de integración y pruebas.

### 8.1.5 Tipos de SDLC

En esta sección comentaremos los distintos tipos de SDLC que podemos encontrar y utilizar en nuestros ciclos de vida de desarrollo de software.

#### 8.1.5.1 Microsoft Trustworthy Computing SDL

Este ciclo de vida resulta bastante popular y es muy utilizado por las empresas que requieren realizar software seguro. En la figura 8.5 se puede visualizar el esquema.



Figura 8.5 Fases de la metodología seguida por Microsoft.

Las tareas que se llevan a cabo son las siguientes:

- En primer lugar, se debe **formar a los desarrolladores** en seguridad, para que todos los componentes se desarrollen conociendo las amenazas.
- **Las tareas de seguridad en las actividades de requisitos consisten en establecer qué requisitos** de seguridad existen en el proyecto. En

este punto, cada equipo de desarrollo debe tener en cuenta como requisitos las características de seguridad para cada fase.

- **Las tareas de seguridad en las actividades de diseño** consisten en definir los requisitos de diseño con sus necesidades de seguridad. Se aportará documentación sobre los elementos que se encuentren en la superficie de un ataque al software.
- **Las tareas de seguridad en las actividades de implementación** son la aplicación de los estándares de desarrollo y de pruebas. Posteriormente, se aplicará software que compruebe la seguridad. Además, se realizarán pruebas de code review.
- **Las tareas de seguridad en las pruebas de verificación y validación** son análisis dinámico sobre la aplicación, revisiones de código desde el punto de vista de la seguridad y pruebas centradas en la seguridad del software.
- Se necesita generar un **plan de incidentes al final del proceso**, una revisión final de toda la seguridad del proceso y crear un plan ejecutivo de respuesta ante incidentes, donde se obtendrá un feedback de todo lo que ocurre en la liberación del software.

#### 8.1.5.2 CLASP

CLASP (Comprehensive Lightweight Application Security Process) es un proceso de ciclo de vida que sugiere una serie de diferentes actividades en todo el ciclo de vida de desarrollo, en un intento de mejorar la seguridad. Entre ellas se encuentra un enfoque específico para los requisitos de seguridad.

Dispone de un enfoque metodológico, el cual podría ser integrado en otros procesos de desarrollo de software. Cuenta con determinadas propiedades, como son su fácil adaptación a otros entornos y la eficiencia. Su fuerte es la riqueza de recursos de seguridad de que dispone, lo cual deberá implementarse en las actividades del SDLC.

CLASP se organiza en forma de vistas, donde todas las vistas se interconectan entre sí:

- **Concepts view**
- **Role-Based view**
- **Activity-Assessment view**
- **Activity-Implementation view**
- **Vulnerability view**

Los roles dentro de CLASP son muy importantes y existen siete: gerente, arquitecto, ingeniero de requisitos, diseñador, codificador, tester y auditor de seguridad. Se puede ver que la seguridad se encuentra incluida, con una figura importante, dentro del proceso.

La vista **Activity-Assessment** es de las más importantes, ya que permite identificar actividades o tareas que pueden ejecutarse. Se trata de tareas relacionadas con la seguridad del desarrollo del software, como la identificación de una política de seguridad y la documentación de los requisitos relevantes con la seguridad.

#### 8.1.5.3 TSP-Secure

TSP-Secure extiende lo que plantea el TSP, **Team Software Process**, con el objetivo de conseguir un desarrollo de aplicaciones seguras. Uno de los principales objetivos de esta metodología estriba en conseguir que las organizaciones puedan mejorar su manera de construir software seguro y de calidad, y que, además, este sea compatible con el CMMI.

Los objetivos de TSP-Secure son, básicamente, tres:

- Ser capaces de detectar los fallos de seguridad en la generación de código.
- Ser capaces de responder rápidamente ante el descubrimiento de nuevas amenazas en el código.
- Promover la utilización de prácticas de seguridad para el desarrollo.

El flujo que cumple es el representado en la figura 8.6.

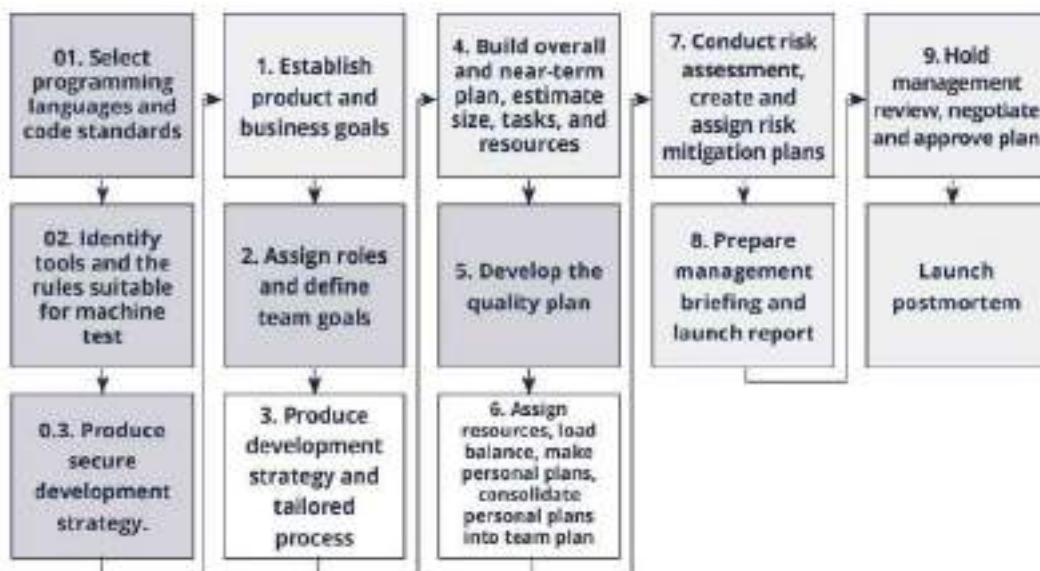


Figura 8.6 Fases de la metodología TSP-Secure.

TSP-Secure necesita que se seleccione alguna norma de codificación segura durante la fase de requisitos. Miembros del equipo aplican pruebas de conformidad en temas de seguridad de la aplicación como parte del propio proceso de desarrollo; esto se realiza en cada fase del SDLC, con el fin de verificar que el código es seguro.

#### **8.1.5.4 Oracle Software Security Assurance**

Este SDLC se compone de un número de actividades, incluidas en las fases conocidas de un SDLC, con el fin de garantizar la seguridad del software de la empresa Oracle. Para lograr dicho objetivo, se dispone de cinco elementos o tareas de seguridad en el S-SDLC:

- Manejo de vulnerabilidades
- Eliminación de vulnerabilidades a través de actualizaciones críticas
- Buenas prácticas y compartición de estas en seguridad y codificación segura
- Gestión de la configuración de seguridad y herramientas de validación y verificación
- Comunicación de fallos de seguridad

El manejo de vulnerabilidades se realiza en las fases de diseño, implementación y testing. La eliminación de vulnerabilidades se lleva a cabo durante todo el proceso, pero, sobre todo, en su parte final (puesta en marcha y testing). Las buenas prácticas se ofrecen a los desarrolladores en la fase de requisitos y diseño. La gestión de configuración se prepara al comenzar el proyecto y cubre la fase de requisitos, el diseño, la implementación y el testing. Por último, la comunicación de fallos de seguridad es la respuesta que pone la empresa a los usuarios; se trata de un plan de respuesta ante incidentes cuando la aplicación se libera en producción para usuarios finales.

#### **8.1.5.5 Propuesta híbrida**

Elegir la mejor metodología parece difícil, por lo que podríamos proponer realizar una mezcla de algunas de ellas tomando como base la propuesta por Microsoft y, sobre todo, ser capaces de adaptar las cosas que nos interesan a nuestra forma de trabajar y modificar determinados aspectos que creo que podrían ayudar a la mejora del proceso, al menos de forma práctica. La propuesta es modificarlo de la manera explicada en la figura 8.7.

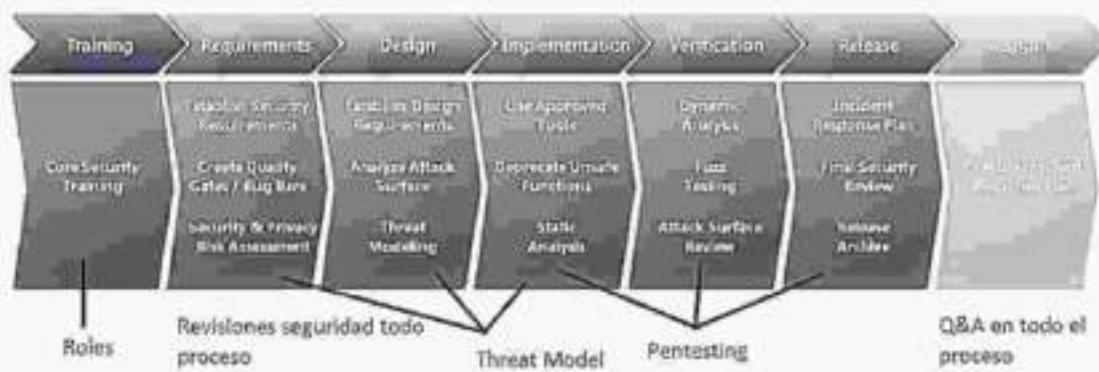


Figura 8.7. Fases de una metodología híbrida tomando como base Microsoft Trustworthy Computing SDL

En primer lugar, y tras definir los roles de los participantes en el proyecto, todos deben asistir a la formación en seguridad por parte de expertos en la materia. Siempre habrá una figura que tenga el rol de **security manager**, que tomará la decisión final sobre las posibles incidencias en temas de seguridad en el proyecto. La persona que tenga el rol de security manager realizará, en paralelo con el **project manager**, un seguimiento del proyecto.

En todas las fases del SDLC, se realizará una review de estado; es decir, desde la fase de requisitos hasta la liberación, se procederá a una revisión de seguridad de todo lo que se tenga hasta ese instante. Además, se extiende el modelado de amenazas a cada una de las siguientes fases (requisitos, diseño e implementación).

Se harán pruebas de pentesting sobre las aplicaciones en tres fases, después de implementar, en la de verificación y, antes de liberar, se pasará una nueva prueba de intrusión. Todo esto debe ser guiado por Q&A, donde se integrará un proceso de seguridad en el que se encuentre la figura del *security manager*, quien dispondrá de un equipo con él para llevar a cabo todo lo necesario en el proyecto.

#### 8.1.6 Tipos de pruebas de seguridad SDLC

Las actividades relacionadas con las pruebas se desarrollan durante todo el ciclo de vida del software. Los preparativos para las pruebas de seguridad pueden comenzar incluso antes de que se hayan definido los requisitos; por ejemplo, la experiencia con sistemas similares puede proporcionar una gran cantidad de información relevante sobre las pruebas que se vayan a realizar.

Al definir estos tipos de pruebas, hay que adaptarlos al tipo de aplicación. Para una aplicación móvil, se han de tener en cuenta otros detalles, como la

multitud de tipos de dispositivos donde puede instalarse y la gestión que la aplicación hace de los recursos como el uso de CPU o memoria.

El software se pone a prueba en muchos niveles en un típico proceso de desarrollo. A continuación se describen los principales tipos de pruebas que resultan comunes en las aplicaciones y los procesos de testing, algunos de los cuales se repiten en diferentes momentos dentro del ciclo de desarrollo con distintos niveles de complejidad. Entre los principales tipos de pruebas, podemos destacar:

- **Pruebas unitarias.** Las pruebas unitarias suelen ser la primera etapa del proceso de pruebas. Este tipo de pruebas consiste en la prueba de funciones individuales, métodos, clases u otros componentes. Como un enfoque funcional para las pruebas unitarias, las pruebas de caja blanca suelen mostrarse muy eficaces en la validación de las decisiones de diseño y los supuestos, y en la búsqueda de errores de programación y errores de implementación.
- **Pruebas de integración.** Las pruebas de integración cuentan como objetivo probar si los componentes de software trabajan juntos como deberían. Los errores de integración surgen normalmente cuando un subsistema hace supuestos injustificados sobre otros subsistemas; por ejemplo, un error de integración puede ocurrir si la función de llamada asume que la función que llama es la responsable de la comprobación de los parámetros. A su vez, los errores de integración constituyen una de las fuentes más comunes de los valores de entrada sin marcar, ya que cada componente puede asumir que las entradas se están comprobando en otros lugares. Durante las pruebas de seguridad, resulta especialmente importante determinar qué flujos de datos pueden ser explotados por un potencial atacante.
- **Pruebas de sistema.** En este punto podríamos englobar pruebas de pentesting y pruebas de estrés, de forma que, cuando se encuentra una vulnerabilidad de esta manera, proporciona una prueba tangible de que la vulnerabilidad pueda ser explotada por un posible atacante.
- **Pruebas de estrés.** Las pruebas de estrés son importantes para la seguridad, ya que el software reacciona de manera diferente cuando se encuentra bajo condiciones de alta carga en nivel de peticiones o procesos. Los atacantes podrían ser capaces de suplantar subsistemas lentos o con debilidades, y las condiciones de carrera podrían llegar a ser más fáciles de explotar.

- **Pruebas de pentesting.** Otro enfoque común para la realización de determinados aspectos de las pruebas de la seguridad del sistema son las pruebas de pentesting, lo que permite evaluar cómo de fácil sería atacar el sistema. En este tipo de pruebas podemos incluir aquellas que tienen como objetivo comprometer la seguridad de la aplicación en particular, la seguridad del sistema operativo sobre el que se ejecuta la aplicación y configuraciones inseguras a nivel de servidor.
- **Pruebas de caja negra.** Un método popular para las pruebas del sistema son las pruebas de caja negra, que utilizan métodos que no requieren acceso al código fuente. Este tipo de pruebas se centran en el comportamiento externamente visible de la aplicación, tales como los requisitos, protocolos o API. Dentro del campo de pruebas de seguridad, las pruebas de caja negra se asocian, normalmente, con las actividades que tienen lugar durante la fase de prueba previa al despliegue o de forma periódica después de que el sistema ha sido implantado en pre/producción.

#### **8.1.7 Conclusiones de ciclo de vida de desarrollo de software (SDLC)**

Los jefes de proyectos y los ingenieros de software deben tratar todos los fallos de software y debilidades como potencialmente explotables. La reducción de las debilidades explotables comienza con la especificación de los requisitos de seguridad de software, junto con la consideración de los requisitos que pueden haber sido pasados por alto. El software que incluye requisitos de seguridad (tales como restricciones de seguridad sobre las conductas de proceso y la resistencia y tolerancia de fallos intencionados) presenta más probabilidades de ser diseñado para permanecer fiable y seguro de cara a un ataque.

Al integrar la seguridad en cada fase del SDLC, permite un enfoque holístico de la seguridad de las aplicaciones, la cual aprovecha los procedimientos que ya existen en la organización. Se debe tener en cuenta que, si bien los nombres de las diversas fases pueden cambiar dependiendo del modelo SDLC utilizado por una organización, cada fase conceptual del arquetipo SDLC se usará para desarrollar la aplicación (es decir, definir, diseñar, desarrollar, implementar y mantener). Cada fase cuenta con consideraciones de seguridad que deberían formar parte del proceso existente, para garantizar un programa de seguridad integral y rentable.

La formación en pruebas de seguridad también ayuda a los desarrolladores a adquirir la mentalidad adecuada para probar una aplicación desde la pers-

pectiva de un atacante. Esto permite que cada organización considere los problemas de seguridad como parte de sus responsabilidades.

Aunque no existe una herramienta mágica, las herramientas desempeñan un papel fundamental en el programa de seguridad general. Existe una gama de herramientas comerciales y de código abierto que pueden automatizar muchas tareas de seguridad de rutina. Tales herramientas simplifican y aceleran el proceso de ayudar, ayudando al personal de seguridad en sus tareas.

Con tantas técnicas y enfoques para probar la seguridad de las aplicaciones web, puede resultar difícil entender qué técnicas utilizar y cuándo usarlas. Es evidente que no existe una técnica única que pueda aplicarse para cubrir eficazmente todas las pruebas de seguridad y garantizar que se han resuelto todos los problemas.

El enfoque correcto es un enfoque equilibrado que incluye varias técnicas, desde revisiones manuales hasta pruebas técnicas donde se cubran las pruebas en todas las fases del SDLC. Dicho enfoque aprovecharía las técnicas más apropiadas disponibles, dependiendo de la fase donde estemos trabajando.

## 8.2 Modelado de amenazas

El modelado de amenazas [https://www.owasp.org/index.php/Modelado\\_de\\_Amenazas](https://www.owasp.org/index.php/Modelado_de_Amenazas) se ha convertido en una técnica bastante popular para ayudar a los diseñadores de sistemas a pensar en las amenazas de seguridad a las que podrían enfrentarse sus sistemas y aplicaciones.

El modelado de amenazas puede ser visto como una evaluación de riesgos para las aplicaciones. De hecho, permite al desarrollador plantear estrategias de mitigación para vulnerabilidades potenciales y lo ayuda a concentrar sus recursos en las partes del sistema que más lo requieren.

En este punto, se recomienda que todas las aplicaciones dispongan de un modelo de amenaza desarrollado y documentado. Los modelos de amenazas deben crearse lo antes posible en el SDLC y deben revisarse a medida que la aplicación evolucione y avance el desarrollo.

Para desarrollar un modelo de amenazas, recomendamos adoptar un enfoque simple que siga la norma **NIST 800-30** para la evaluación de riesgos. Este enfoque implica:

- **Descomponer la aplicación.** Es posible utilizar un proceso de inspección manual para comprender cómo funciona la aplicación, sus activos, su funcionalidad y conectividad.
- **Definir y clasificar los activos.** Se han de clasificar los activos en activos tangibles e intangibles, y hay que clasificarlos según su importancia.
- **Explorar vulnerabilidades potenciales**, ya sean técnicas, operativas o de gestión.
- **Explorar las amenazas potenciales.** Resulta importante obtener una visión de los posibles vectores de ataque desde la perspectiva del atacante.
- **Crear estrategias de mitigación.** Podemos desarrollar controles de mitigación para cada una de las amenazas consideradas realistas.

No existe una forma única de desarrollar modelos de amenazas y realizar evaluaciones de riesgo de información en las aplicaciones. Podemos encontrar diferentes guías OWASP que describen una metodología de modelado de amenaza a nivel de aplicación que puede ser usada como referencia:

- [https://www.owasp.org/index.php/OWASP\\_Risk\\_Rating\\_Methodology](https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology)
- [https://www.owasp.org/index.php/Threat\\_Risk\\_Modeling](https://www.owasp.org/index.php/Threat_Risk_Modeling)

El modelado de amenazas constituye una práctica de seguridad que permite identificar amenazas, ataques y riesgos basados en el diseño de la arquitectura existente, y también para mitigar riesgos potenciales de seguridad. Entre los puntos clave en el modelado de amenazas, podemos destacar:

- Constituye una actividad de equipo. Será más efectivo con la participación de los equipos de operaciones, arquitectos y el equipo de seguridad.
- El propósito del modelado de amenazas no estriba en ofrecer una lista completa de amenazas, sino en identificar amenazas de alto riesgo con módulos clave como autenticación, autorización o manejo de información de los usuarios.
- Se sugiere realizar un modelado de amenazas cuando se realiza el diseño de la arquitectura o en etapas tempranas de diseño y codificación detalladas.

Un proceso típico de modelado de amenazas incluye un diagrama de DFD a través de la revisión de la arquitectura, un análisis de amenazas, una evaluación del impacto del riesgo y una revisión de la implementación que se vaya a realizar. Un modelado de amenazas normalmente comienza con un análisis de la arquitectura. Los diagramas UML se pueden usar en la actividad de modelado de amenazas. Dichos diagramas permiten que se pueda entender el diseño de la arquitectura completa y el flujo de información. El objetivo del modelado de amenazas radica en discutir los riesgos de seguridad más relevantes y de mayor prioridad.

Dependiendo de la complejidad de las aplicaciones, podemos hacer un modelado de amenazas con arquitectura o diseño de alto nivel. Si se trata de un proyecto muy grande y la mayoría de los módulos cuentan con funciones similares, se sugiere realizar el modelado de amenazas con aquellas funcionalidades que puedan representar un mayor riesgo. Estos son los módulos recomendados para el modelado de amenazas:

- Módulos con controles de seguridad como autenticación, autorización, administración de sesiones, cifrado, validación de datos, manejo de errores o registro, administración y manejadores de bases de datos.
- Módulos que pueden interactuar externamente con usuarios externos o API de terceros.
- Módulos que manejan información sensible.
- Módulos heredados que posean algún tipo de vulnerabilidad CVE.

#### 8.2.1 Modelado de amenazas con STRIDE

El modelo de amenaza STRIDE define las amenazas en seis categorías, que son **suplantación de identidad, manipulación, repudio, divulgación de información, denegación de servicio y elevación de privilegios**. Normalmente se utiliza para evaluar el diseño de la arquitectura. En la siguiente tabla se resumen el modelo de amenaza STRIDE y las mitigaciones de seguridad. Además de STRIDE, también se sugiere incluir la privacidad en el análisis:

Amenazas STRIDE	Mitigación
<i>Spoofing</i> (suplantación de identidad)	Autenticación como credenciales, certificados y SSH

Manipulación de datos (HASH256 o firma digital)	Autenticación de repudio
Divulgación de información	Confidencialidad de divulgación de información (cifrado o ACL)
Denegación de servicio	Disponibilidad (balanceo de carga)
Elevación de privilegios	Authorization (listas ACL)
Privacidad	Control de acceso y consentimiento del usuario

El análisis de STRIDE normalmente involucra a la entidad (usuario, administrador o aplicación externa), el proceso (servidor web, FTP o servicio), el almacén de datos (base de datos o archivo) y el flujo de datos (parámetros o información entre módulos, procesos y sistemas). En la siguiente tabla se muestran algunos ejemplos de amenazas STRIDE:

Amenazas STRIDE	Ejemplos
Spoofing (suplantación de identidad)	<p>La entidad (usuario o cliente) puede falsificar su identidad.</p> <p>El proceso puede falsificar su fuente.</p>
Manipulación de datos	<p>El proceso puede ser manipulado, como en un ataque de inyección de DLL.</p> <p>El almacén de datos puede ser manipulado.</p> <p>El flujo de datos puede ser alterado con técnicas como MITM.</p>
Divulgación de información	<p>El proceso en sí puede incluir una clave de cifrado y puede revertirse.</p> <p>El almacén de datos guarda copias de contraseñas en texto claro.</p> <p>El flujo de datos transmite la contraseña sin un canal de cifrado.</p>

Denegación de servicio	<p>El proceso puede estar conectado a demasiados clientes y estar sobrecargado.</p> <p>El almacén de datos está dañado o lleno.</p> <p>El flujo de datos se desconecta y no puede llegar al destino.</p>
Elevación de privilegios	<p>El proceso se halla en modo usuario no administrador, pero puede ejecutar un comando de modo root.</p> <p>El proceso se ejecuta con permisos adicionales.</p>
Privacidad	<p>La entidad externa (aplicación cliente) puede recopilar la información, pero no informa al usuario.</p> <p>El almacén de datos mantiene la información en los registros sin anonimización.</p>

La guía **OWASP Application Threat Modeling** contiene más ejemplos basados en el diagrama DFD: [https://www.owasp.org/index.php/Application\\_Threat\\_Modeling](https://www.owasp.org/index.php/Application_Threat_Modeling)

Se recomienda también utilizar una lista de verificación o una lista de bibliotecas de amenazas, como una lista de CWE (<https://cwe.mitre.org/data/index.html>), enumeración y clasificación de patrones de ataque comunes (CAPEC), o técnicas tácticas adversas y conocimiento común (ATT & CK).

En cuanto a herramientas que nos ayuden a modelar los diagramas, podemos destacar algunas que incluyen una biblioteca de amenazas por defecto. Se trata de una herramienta ideal para documentar el informe de análisis de modelado de amenazas. Normalmente, la arquitectura de la aplicación y el diagrama del sistema DFD se presentaban seguidos de la identificación de la amenaza:

- **Microsoft Threat Modeling:**
  - Herramienta de modelado de amenazas de Microsoft
  - <https://www.microsoft.com/en-us/download/details.aspx?id=49168>
- **OWASP Threat Dragon:**
  - [https://www.owasp.org/index.php/OWASP\\_Threat\\_Dragon](https://www.owasp.org/index.php/OWASP_Threat_Dragon)
- **Mozilla SeaSponge:**
  - <http://mozilla.github.io/seasponge>

### **8.3 Perspectiva del atacante**

La perspectiva del atacante consiste en examinar el software en la dirección en que un posible atacante lo haría, es decir, desde el exterior al interior. Esto requiere pensar cómo piensan los atacantes, y el análisis y la comprensión del software de la manera en que podrían realizar un ataque con éxito. Gracias a una mejor comprensión de cómo resulta probable que sea atacada la aplicación, el equipo de desarrollo puede aplicar mejor las medidas desde el punto de vista del atacante.

El principal desafío en la construcción de software seguro reside en que es mucho más fácil de encontrar vulnerabilidades en el software de lo que es hacerlo seguro. Sus diseñadores necesitan asegurarse de que es seguro contra muchos tipos diferentes de ataques, no solo los aparentemente obvios. Esto, claramente, no constituye una tarea trivial. Sin embargo, el atacante puede solo necesitar encontrar una vulnerabilidad explotable para lograr su objetivo y acceder.

La construcción de software seguro se ve agravada por la naturaleza virtual (no física) del software. Con muchos sistemas, el atacante puede llegar a poseer el software (la obtención de una copia local de atacar es a menudo trivial) o podría atacarlo desde cualquier parte del mundo a través de las redes. Dada la capacidad de los atacantes para atacar de forma remota y sin acceso físico, las vulnerabilidades quedan ampliamente mucho más expuestas a un ataque.

Las ventajas de los atacantes se ven reforzadas por el hecho de que los atacantes han estado aprendiendo cómo explotar el software desde hace varias décadas, pero la comunidad de desarrollo de software, en general, no se ha mantenido al día con el conocimiento que los atacantes han ganado. El problema sigue creciendo, en parte por el temor tradicional de la enseñanza de cómo se explota el software; en realidad, podría reducir la seguridad de software, ayudando a los atacantes ya existentes e incluso crear nuevos.

Para identificar y mitigar vulnerabilidades en el software, la comunidad de desarrollo necesita, más que una buena ingeniería de software, prácticas de análisis, un sólido conocimiento de las características de seguridad de software y un potente conjunto de herramientas. Todas estas cosas resultan necesarias, pero no son suficientes. Para mostrarse eficaz, la comunidad ha de pensar de forma creativa y contar con una sólida comprensión de la perspectiva del atacante y de los métodos utilizados para explotar software.

## **8.4 Patrones de ataque**

Para que los equipos de desarrollo de software puedan tomar ventaja de la perspectiva del atacante en la construcción de la seguridad del software, primero ha de haber un mecanismo para capturar y comunicar esta perspectiva de los expertos, haciendo un traspaso de conocimientos a los equipos.

Los patrones de diseño son una herramienta muy utilizada por la comunidad de desarrollo del software para ayudar a resolver los problemas recurrentes encontrados durante el desarrollo del software. Con tales patrones, se intentan abordar de frente los problemas espinosos de la arquitectura segura, estable y eficaz del software y del diseño. Desde la introducción de patrones de diseño, la construcción del modelo se ha aplicado a muchas otras áreas de desarrollo de software. Una de estas áreas es la seguridad del software y la representación de la perspectiva del atacante en forma de patrones de ataque.

Los patrones de ataque aplican el paradigma problema-solución de patrones de diseño en un contexto constructivo-destructivo. Aquí, el problema común dirigido por el patrón representa el objetivo del atacante de software, y la solución del patrón representa los métodos comunes para llevar a cabo el ataque. En resumen, los patrones de ataque describen las técnicas que los atacantes podrían utilizar para romper software.

Los patrones de ataque pueden ser valiosos durante la planificación de la arquitectura del software y del diseño de dos maneras:

- En primer lugar, algunos patrones de ataque describen los ataques que explotan directamente fallos en la arquitectura y en el diseño del software; por ejemplo, al hacer el patrón de ataque invisible al cliente, se explotan problemas de confianza del lado del cliente que resultan evidentes en la arquitectura de software.
- En segundo lugar, los patrones de ataque de todos los niveles pueden proporcionar un marco útil para las amenazas a las que el software es probable que se enfrente y, con ello, determinar qué características arquitectónicas y de diseño se deben evitar o incorporar específicamente.

Los patrones de ataque pueden resultar útiles durante la ejecución, ya que enumeran las debilidades específicas dirigidas por los ataques utilizados y permiten a los desarrolladores asegurarse de que estas deficiencias no se producen en su código. Tales debilidades podrían adoptar los errores válidos de imple-

mentación o construcciones que, simplemente, se codifican y que pueden tener consecuencias para la seguridad si se usa incorrectamente. Desafortunadamente, los errores de ejecución no siempre resultan fáciles de evitar o detectar y corregir. Incluso después de que la aplicación tenga una revisión técnica básica, aún pueden seguir siendo abundantes y pueden hacer que el software sea vulnerable a acciones muy peligrosas.

Los problemas de seguridad subyacentes con errores de código no válido suelen ser más difíciles de identificar. No pueden ser identificados con un simple análisis de caja negra; en cambio, requieren un conocimiento especializado de lo que muestran estas vulnerabilidades. También se debe determinar cómo se pueden utilizar los patrones de ataque para identificar debilidades específicas para la orientación y la mitigación del ataque a través de la información suministrada al desarrollador de antemano. A menudo, este paso se puede automatizar mediante el uso de herramientas de análisis de seguridad.

Para finalizar, podríamos comentar que un atacante no está particularmente interesado en las características funcionales del sistema, a menos que dichas características propicien una vía de ataque. En su lugar, el atacante, normalmente, busca defectos y otras condiciones fuera de la norma que permitan una intrusión exitosa. Por esta razón, es importante que, entre los requisitos de los analistas y desarrolladores, deban pensar en la perspectiva del atacante y no solo en la funcionalidad del sistema desde la perspectiva del usuario final, que no tiene por qué poseer conocimientos técnicos avanzados.

## **8.5 OWASP Testing Framework y perfiles para pruebas de seguridad**

OWASP dispone de su propio framework de pruebas, que permite la participación de los analistas y testers desde las primeras etapas de desarrollo del proyecto. Entre las fases que proporciona el framework, podemos destacar:

### **Fase 1: antes de comenzar la implementación**

- 1A: revisión de políticas y estándares.
- 1B: desarrollo de criterios de medición y métricas.

### **Fase 2: durante la definición y diseño**

- 2A: revisión de los requisitos de seguridad.

- 2B: revisión de diseño y arquitectura.
- 2C: creación de modelos UML.
- 2D: creación de modelos de amenazas.

#### **Fase 3: durante el desarrollo**

- 3A: tutoriales de código.
- 3B: revisiones de código.

#### **Fase 4: durante el desarrollo**

- 4A: pruebas de penetración de aplicaciones.
- 4B: pruebas de gestión de la configuración.

#### **Fase 5: mantenimiento y operaciones**

- 5A: realización de revisiones de gestión operativa.
- 5B: desarrollo de controles de salud periódicos.
- 5C: aseguramiento de la verificación del cambio.

#### **OWASP sugiere cuatro técnicas de prueba para pruebas de seguridad:**

- Inspecciones y revisiones manuales. Pueden incluir revisiones de diseño, procesos, políticas, documentación e incluso entrevistas a personas.
- Revisiones de código.
- Modelado de amenazas.
- Pruebas de penetración.

En cuanto a los **perfiles necesarios para abordar el conjunto de pruebas de seguridad**, podemos destacar:

- **Security analyst.** Este rol tiene como objetivo dar soporte en las etapas iniciales. Se trata de un perfil experimentado que aporta conocimientos en las fases de requisitos y diseño, al proporcionar una base sólida sobre la que construir una aplicación segura. Se trata de un perfil que dispone de amplia experiencia en seguridad informática (más de cinco años) trabajando como penetration tester, en el campo de la seguridad web o cualquier otra posición que requiera realizar pruebas tanto a nivel estático como dinámico. Entre las principales responsabilidades, podemos destacar:
  - Definir los criterios de severidad de las vulnerabilidades.
  - Analizar la lógica de negocio de la aplicación.
  - Determinar los requisitos de seguridad aplicables.
  - Establecer el alcance del testing.
  - Revisar y colaborar en el diseño de la aplicación y su arquitectura.

- **Security tester.** El security tester representa un rol operativo cuya principal misión es la correcta ejecución de las pruebas de seguridad. Se trata de un rol clave, puesto que de sus conocimientos depende el porcentaje de detección de fallos de seguridad (vulnerabilidades). Entre las principales responsabilidades, podemos destacar:
  - Análisis estático y dinámico.
  - Evaluación de las vulnerabilidades.
  - Redacción de los informes.
  - Volver a probar los defectos y fallos que se hayan arreglado.
  - Información de los fallos por la vía adecuada.

## 8.6 OWASP Security Knowledge Framework (SKF)

OWASP Security Knowledge Framework <https://skf.readme.io/docs/introduction> es una herramienta que utiliza el estándar de verificación de seguridad de aplicaciones OWASP para ayudar a los desarrolladores, con el objetivo de que las aplicaciones sean seguras desde la fase de diseño. OWASP SKF dispone de un portal de conocimiento de pruebas de seguridad, que incluye listas de verificación, la base de conocimiento de seguridad y ejemplos de código de **OWASP ASVS (Application Security Verification Standard)**.

El primer paso para usar la herramienta estriba en añadir un nuevo proyecto. Esto se puede hacer desde la pestaña de “Proyectos”.

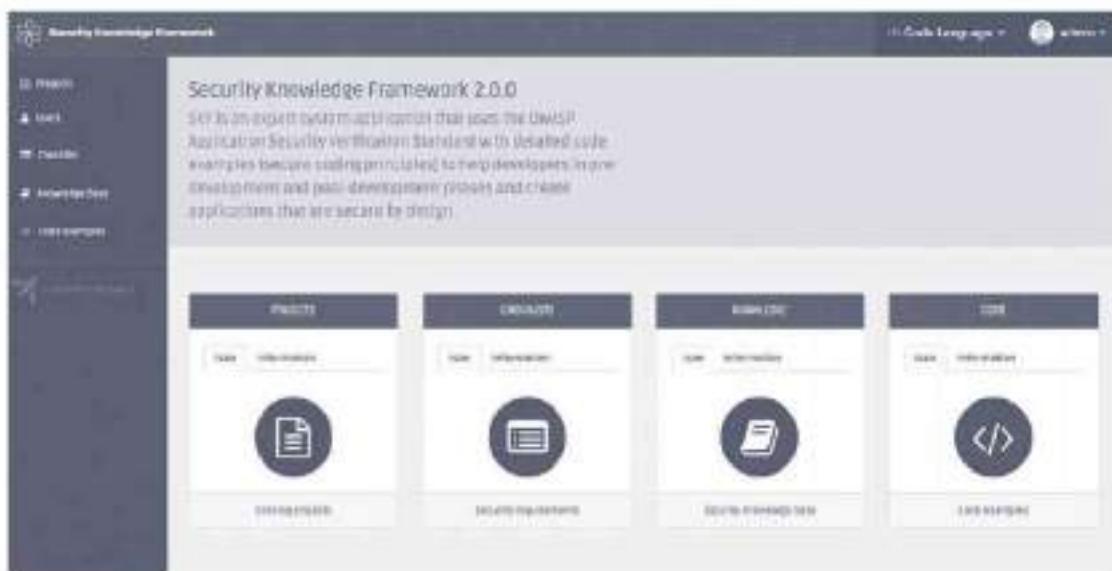


Figura 8.8 Pantalla principal del Security Knowledge Framework.

Después de crear el proyecto, disponemos de un asistente para configurarlo. Esto incluye seleccionar el nivel ASVS, configurar los ajustes previos al desarrollo y, finalmente, crear un primer sprint.



Figura 8.9 Pantalla de creación de un proyecto SKF.

En la siguiente pantalla (figura 8.10), podemos seleccionar nuestro nivel ASVS.

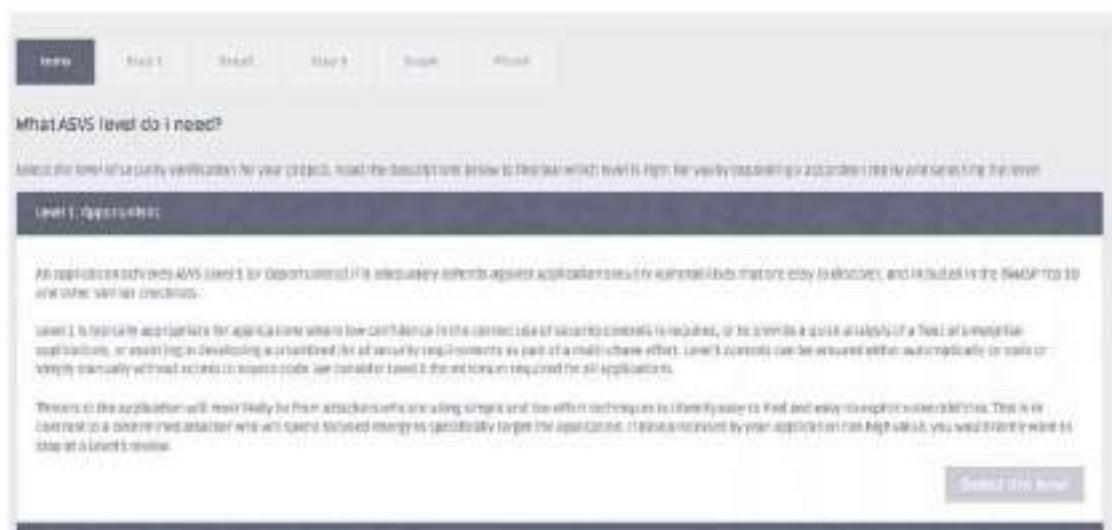


Figura 8.10 Selección del nivel ASVS de un proyecto SKF.

En el primer paso damos nombre al proyecto, junto con una descripción.

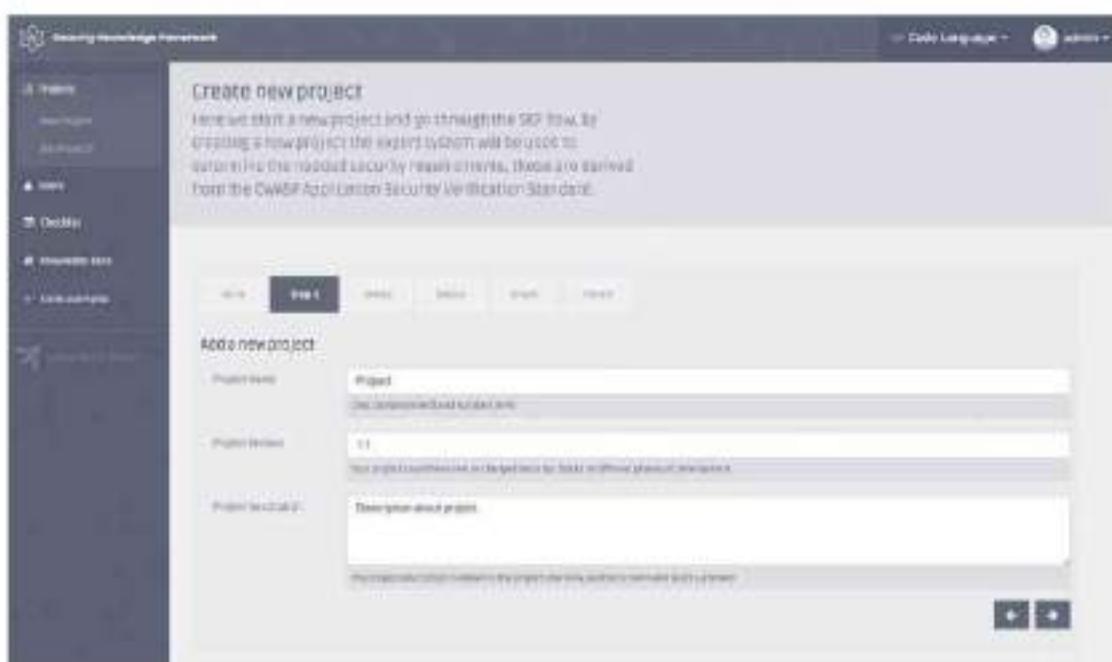


Figura 8.11 Nombre y descripción de un proyecto SKF.

En el segundo paso, seleccionamos algunas configuraciones previas al desarrollo sobre cosas como la arquitectura y el diseño, así como definimos el alcance del ASVS.



Figura 8.12 Configuraciones en el paso 2 de un proyecto SKF.

Después de configurar los ajustes previos al desarrollo, definimos el sprint donde se definen qué funcionalidades vamos a desarrollar y qué técnicas vamos a aplicar.

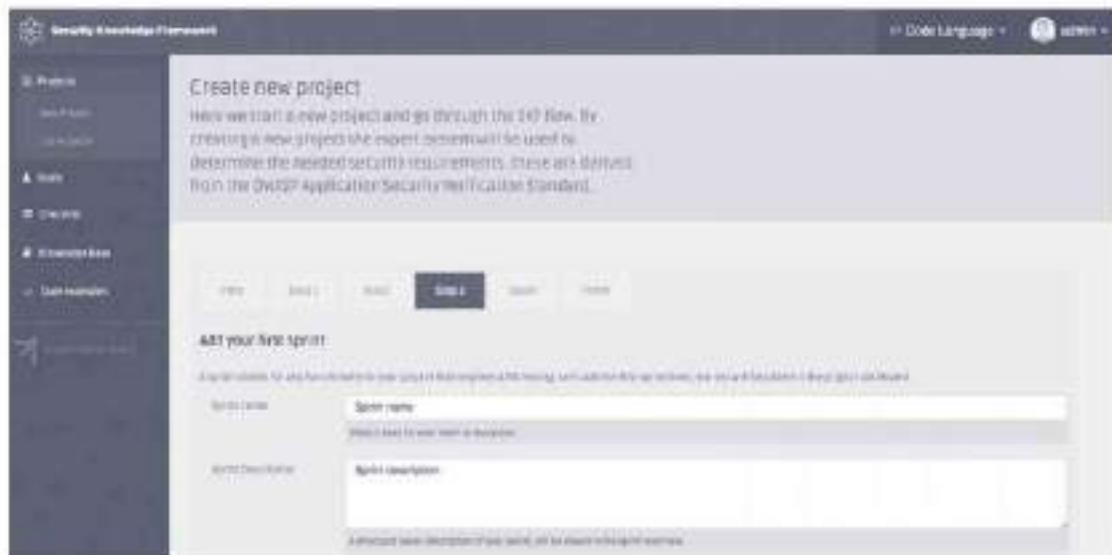


Figura 8.13 Configuraciones en el primer sprint de un proyecto SKF.

Posteriormente, seleccionamos las técnicas que vamos a utilizar para crear las funcionalidades del primer sprint. En este paso tenemos que responder a preguntas como:

- ¿Se va a implementar un mecanismo de autenticación?
- ¿Se van a implementar funciones que envíen parámetros mediante el método POST?
- ¿Se van a implementar funciones que envíen parámetros mediante el método GET?



Figura 8.14 Preguntas que responder para el primer sprint de un proyecto SKF.

Una vez se ha creado el proyecto, aparecerá en el panel de proyectos (**Project Dashboard**). En este caso, vemos los requerimientos de seguridad de nuestro sprint. El volumen de requisitos de seguridad se convierte en archivos cada vez que se van añadiendo sprints. En la figura 8.15 se muestran los estados del sprint en el panel de control. Se observan el estado y la cantidad de issues abiertos por sprint.

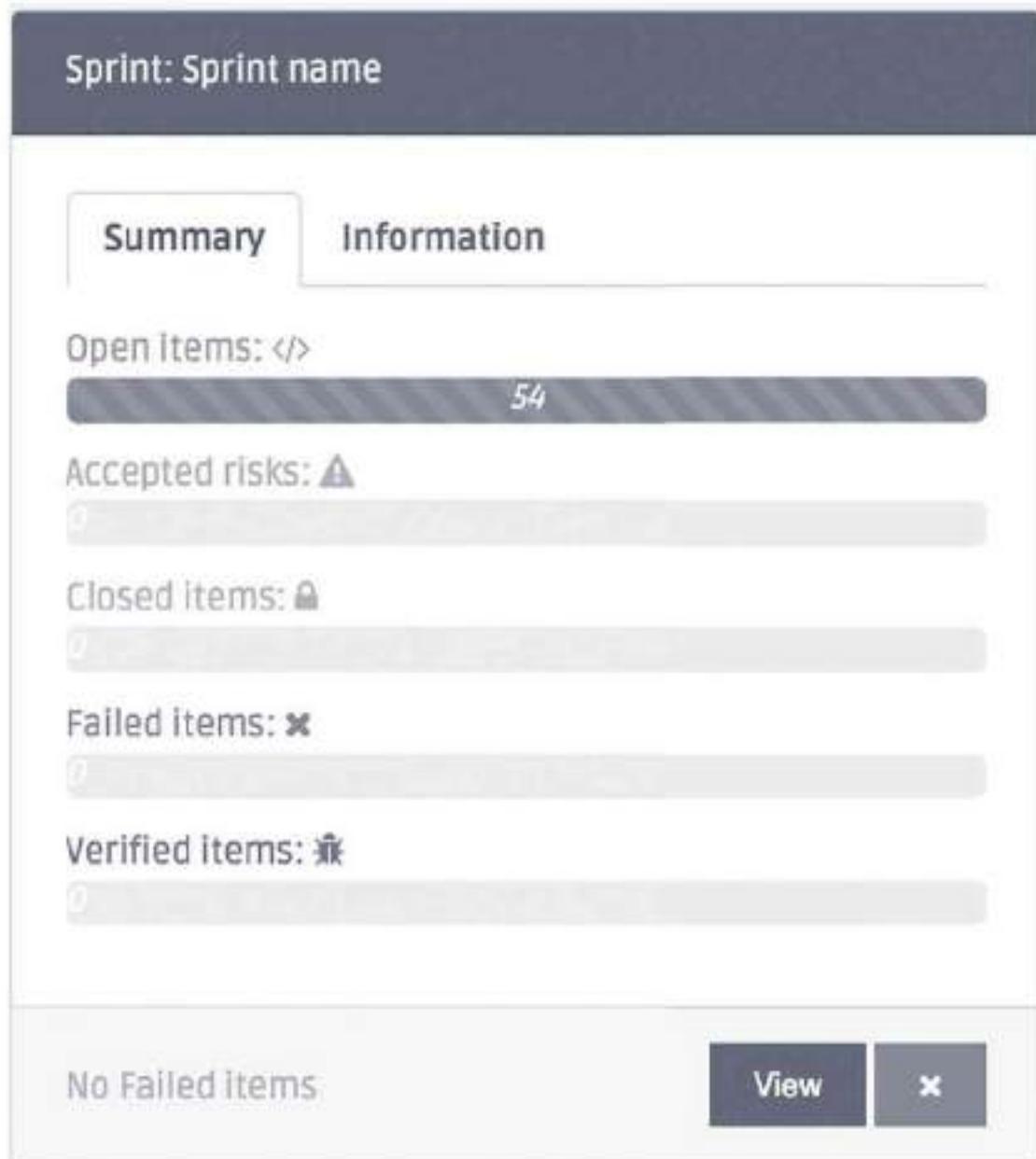


Figura 8.15 Project Dashboard de un proyecto SKF.

También podemos visualizar los requerimientos de seguridad que se correlacionan con los elementos de la base de conocimientos para proporcionar información más detallada sobre los vectores de ataque que se corresponden con ese elemento, así como las soluciones sobre cómo mitigar determinados ataques en el código.

The screenshot shows a web-based application for managing security requirements. The top navigation bar includes 'Code Language' and 'admin'. On the left, a sidebar lists 'Project', 'Requirements', 'Components', 'Audit', 'Reviews', and 'Code Examples'. The main content area has a header 'Sprint summary for: Development' with a note about 'CWEs, known or suspected vulnerabilities, and other components to review during the sprint'. Below this is a card titled 'Identify all application components' with tabs for 'Summary', 'Details', and 'Comments'. A detailed description of the requirement is provided, mentioning 'Identify all application components' and 'An external developer has access to the codebase via a public GitHub repository and can make changes to it.' At the bottom are buttons for 'Edit Details', 'Add Note', and 'Assign Task'.

Figura 8.16 Requerimientos de seguridad de un proyecto SKF.

Si seleccionamos la **pestaña de estado (Status)**, podemos añadir comentarios para un determinado requisito de seguridad y cambiar su estado a "Cerrado", "Aceptado" o "Volver a abrir". Los comentarios que se proporcionen serán importantes para realizar un seguimiento a nivel de auditoría de lo que está sucediendo y comprobar si los requisitos se cumplen. En este punto tanto desarrolladores como especialistas en seguridad se retroalimentan por los comentarios dados por cada uno de ellos.

This screenshot shows the 'Status' tab for a specific requirement. The requirement details remain the same as in Figure 8.16. The 'Status' tab is selected, showing a comment from a 'Security Specialist' stating: 'Requirement has been reviewed and accepted. No further action is required.' Below this, there is a text input field containing the message: 'The IP's that are allowed to access the interface are now defined.' At the bottom are buttons for 'Edit Details', 'Add Note', and 'Assign Task'.

Figura 8.17 Estado del proyecto para un requisito específico de seguridad.

En la pestaña de **registro de auditoría** podemos ver el seguimiento de auditoría del requisito de seguridad correspondiente.

The screenshot shows the 'Audit' tab selected in the left sidebar under 'Sprint summary for: Development'. The main content area displays three audit logs:

- [REDACTED] [REDACTED] [REDACTED]
- [REDACTED] [REDACTED] [REDACTED]
- [REDACTED] [REDACTED] [REDACTED]

Below the logs, there are tabs for 'Summary', 'Details', and 'Auditing'. A note at the bottom states: 'The IP address is allowed to access the application and has administrator rights.'

Figura 8.18 Registro de auditoría y resumen del estado.

En este punto, un especialista en seguridad usará la aplicación para comprobar los elementos que han sido cerrados por los desarrolladores. Tan pronto como un desarrollador cierra un elemento, obtiene un ícono verde, lo que significa que está abierto para que lo verifique un especialista en seguridad. Cuando el especialista en seguridad visite esta página, cambiará su rol, seleccionando el botón correspondiente, y verificará o proporcionará un error en función de los resultados de las pruebas de seguridad.

The screenshot shows the 'Audit' tab selected in the left sidebar under 'Sprint summary for: Security'. The main content area displays three audit logs:

- [REDACTED] [REDACTED] [REDACTED]
- [REDACTED] [REDACTED] [REDACTED]
- [REDACTED] [REDACTED] [REDACTED]

Below the logs, there are tabs for 'Summary', 'Details', and 'Auditing'. A note at the bottom states: 'User is allowed to access the application and has administrator rights.'

Figura 8.19 Aceptar o dar por inválido un determinado requisito de seguridad.

La base de conocimientos resulta especialmente útil cuando el sistema le proporcionó retroalimentación a través de diferentes “patrones de diseño seguro”. Tales patrones contienen múltiples elementos de la base de conocimientos que es posible examinar para obtener más información detallada.

The screenshot shows a web-based application titled "Security Knowledge Framework". The left sidebar contains navigation links: "Home", "Code", "Checklist", "Knowledge Base", and "Documentation". The main content area has a header "Code examples: php" with a sub-header "Use for quick evaluation of implementing this secure code patterns into your project. In-depth information how to implement specific security controls can be found in patterns". Below this is a search bar and a large dark panel containing a list of items under the heading "This pattern". The items listed are: "Anti-Reflection Methods", "Anti-Public Methods", "Control Responsibility", "Design Patterns", "Dynamic Control Checks", "Security Based on Structure", "SQL query", "Localization", "Implementation", and "References".

Figura 8.20 Base de conocimientos del framework SFK.

The screenshot shows the same web-based application. The main content area has a header "Knowledge Base" with a sub-header "In-depth information on how to approach specific horizontally or problems with explanation of attack surface and mitigation". Below this is a search bar and a large dark panel containing a list of items under the heading "Common Issues in Web Systems". The items listed are: "Information disclosure", "Cross-site Scripting (XSS)", "Cross-site Request Forgery (CSRF)", "Session hijacking", "SQL injection", "File inclusion", "Remote code execution", "Denial of Service (DoS)", "Cross-site Scripting (XSS) - Persistent", "Cross-site Request Forgery (CSRF) - Persistent", "Session hijacking - Persistent", "Remote code execution - Persistent", and "File inclusion - Persistent".

Figura 8.21 Base de conocimientos del framework SFK.

Se proporcionan también ejemplos de código que permiten seguir las mejores prácticas para desarrollo seguro.

The screenshot shows a web-based interface for the Security Knowledge Framework. On the left, there's a sidebar with navigation links: Projects, Assets, Checks, and Knowledge base. Below that is a 'Code Examples' section with a 'PHP' tab selected. The main content area displays a code editor with PHP code. The code includes imports for 'Image', 'ImageFilter', and 'Image Intervention'. It defines a function 'processImage' that takes a file path and a save path as parameters. Inside the function, it uses 'Image::load' to load the image, applies filters like 'filterGrayscale' and 'filterSepia', and then saves it using 'Image::save'. A preview image is shown above the code editor.

Figura 8.22 Ejemplos de código en PHP.

También podemos consultar las listas de verificación para cada nivel ASVS que contienen los elementos como base de conocimientos a nivel de arquitecturas, diseño y modelos de amenazas.

This screenshot shows the 'Security checklists' section of the framework. The sidebar remains the same. The main area has a heading 'Security checklists' with a sub-instruction: 'Review the ASVS checklist that you can select for reference material, topic or the security controls that you want to review the candidate knowledge base item.' Below this is a table titled 'ASVS Level' with three rows corresponding to ASVS Level 1, ASVS Level 2, and ASVS Level 3. Each row contains a link to the respective checklist.

ASVS Level	Description	# ASVS	View
ASVS Level 1	A summary of the ASVS Level 1 items.	3	<a href="#">View</a>
ASVS Level 2	A summary of the ASVS Level 2 items.	6	<a href="#">View</a>
ASVS Level 3	A summary of the ASVS Level 3 items.	3	<a href="#">View</a>

Figura 8.23 Listas de verificación para cada nivel ASVS.

Al seleccionar un elemento de la lista de verificación, podemos ver la descripción junto con la base de conocimientos correspondiente.

The screenshot shows a software interface titled "ASVS" at the top. Below it, a section titled "1.0 Architecture, design and threat modelling" is expanded. Under this section, several numbered items are listed:

- 1.10 Verify that there is no sensitive business logic, secret keys or other proprietary information in client side code.
- 1.11 Verify that all application components are identified and are known to be needed
- 1.12 Verify that all application components, libraries, modules, frameworks, platform, and operating systems are free from known vulnerabilities.
- 1.13 Verify that a highlevel architecture for the application has been defined.

Below the list, there are two expandable sections: "Description:" and "Solution:". The "Description:" section contains the text: "Whenever you are developing an application you want to map all the architecture it contains. Whenever there are breaches, updates, or other escalations it makes it easy and transparent for forensics, operators, and developers to do their job as fast as possible." The "Solution:" section contains the text: "Verify that a highlevel architecture for the application has been defined. This".

Figura 8.24 Descripción de un elemento de la lista de verificación.

## 8.7 Seguridad en ingeniería del software

La seguridad del software aplica los principios de la seguridad de información al desarrollo de software. El concepto de la seguridad en los sistemas de software constituye un área de investigación que ha pasado a ser vital dentro de la ingeniería de software. Con el crecimiento de Internet y otras aplicaciones sobre redes, como el comercio o el correo electrónico, la posibilidad de

ataques se ha incrementado notablemente, como también lo han hecho las consecuencias negativas de estos ataques.

En la actualidad, prácticamente todo sistema debe incorporar cuestiones de seguridad para defenderse de ataques maliciosos. El desarrollador ya no solo debe concentrarse en los usuarios y sus requerimientos, sino también en los posibles atacantes. Esto ha motivado cambios importantes en el proceso de diseño y desarrollo de software para incorporar a la seguridad dentro de los requerimientos críticos del sistema.

Estos cambios son los primeros pasos en la concienciación de los ingenieros de software acerca de la importancia de obtener un software seguro. Como todo gran cambio, no se logra de un día para otro, sino que se trata de un proceso gradual que requiere tiempo y maduración. Todavía la ingeniería del software no ha dado una respuesta eficaz, coherente y aplicable que satisfaga plenamente a la comunidad de desarrolladores, sino que las aproximaciones actuales se encuentran con errores y debilidades, fruto de que todavía es un proceso que se encuentra en sus inicios.

Resulta una práctica habitual en muchas organizaciones la "puesta en producción" de aplicaciones y sistemas sin haber pasado un porcentaje mínimo de pruebas de seguridad. Por lo general, en el mejor de los casos, se coordinan unas pruebas de seguridad una vez que la aplicación ya está desarrollada o puesta en producción. Aquí muchas veces se encuentran errores que requieren el rediseño de parte de la aplicación, lo cual implica un coste adicional en tiempo y esfuerzo.

En la figura 8.25 vemos los costes que supone solucionar un error en alguno de los entornos, donde podemos destacar que, una vez que hemos desplegado la aplicación en producción, es 30 veces más costoso corregir una vulnerabilidad si lo comparamos con el coste de corregirla durante la fase de diseño.

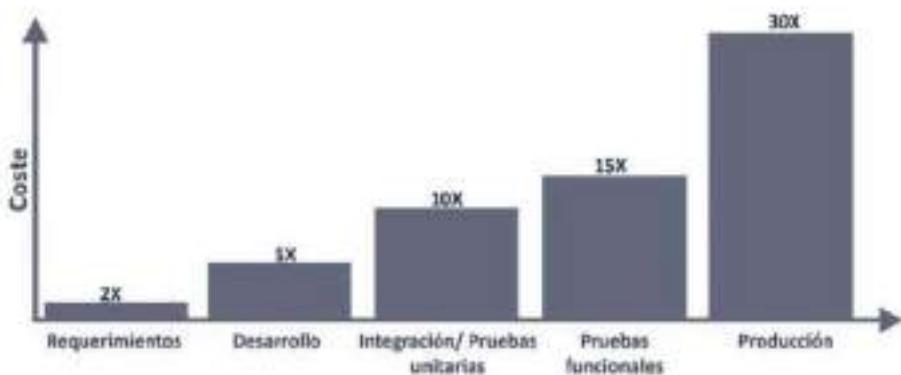


Figura 8.25 Costes de arreglar un fallo de seguridad en cada una de las fases de desarrollo.

La falta de seguridad se origina en dos problemas fundamentales: los sistemas que son teóricamente seguros pueden resultar inseguros en la práctica. Además, los sistemas se muestran cada vez más complejos y dicha complejidad proporciona más oportunidades para los ataques. Resulta mucho más fácil probar que un sistema es inseguro que demostrar que uno es totalmente seguro; de hecho, podemos decir que hoy día no hay sistema o aplicación seguro al 100 %.

Para el caso de que alguien externo a la organización descubra una vulnerabilidad en la aplicación, se debe dar a conocer algún canal para que el investigador informe correctamente a la empresa. Podemos agregar una política de seguridad para nuestra aplicación web utilizando el servicio <https://securitytxt.org>. Básicamente, permite definir el texto relacionado con la divulgación responsable de los fallos de seguridad e informar de forma adecuada a la empresa sobre las vulnerabilidades encontradas.

## 8.8 Bibliografía y fuentes de información

En el libro *The Art of Software Security Testing* de Chris Wysopal y Elfriede Dustin, publicado en el año 2006, se revisa el diseño de software y vulnerabilidades de código, y se proporcionan directrices sobre cómo evitarlos. En esta obra se describen formas de personalizar las herramientas de depuración de software, para poner a prueba los aspectos únicos de cualquier programa de software y, luego, analizar los resultados para identificar las vulnerabilidades explotables. Incluye los siguientes temas:

- Pensando en la manera que los atacantes piensan.
- La integración de las pruebas de seguridad en el SDLC.
- El uso de modelos de amenazas para priorizar las pruebas basadas en el riesgo.
- Laboratorios de construcción de pruebas para la realización de pruebas en caja blanca, gris y negra.
- Elegir y utilizar las herramientas adecuadas.
- La ejecución de los principales ataques desde la inyección de fallos hasta el desbordamiento de búfer.

- Determinar qué defectos son los más propensos a ser explotados.

El libro *Exploiting Software: How to Break Code*, publicado en el año 2004, <http://www.exploitingsoftware.com> proporciona ejemplos de ataques reales, patrones de ataque, herramientas y técnicas utilizadas por los atacantes para romper software. En él se discute sobre la ingeniería inversa y los ataques clásicos contra el servidor, así como sobre las técnicas para la elaboración de entradas malintencionadas, desbordamientos de búfer y rootkits.

Por su parte, en el libro *How to Break Software: A Practical Guide to Testing*, publicado en 2003 por James Whittaker, se definen técnicas y ataques diseñados para revelar vulnerabilidades de seguridad en las aplicaciones. En el libro se analizan los modelos de fallos para las pruebas de seguridad de software, la creación de escenarios de entrada de usuarios y las formas de atacar los diseños de software. Se analizan también el código, donde resulta más probable encontrar vulnerabilidades, tales como interfaces de usuario o las dependencias, así como el intercambio de datos entre los procesos y la memoria.

La página *CERT Secure Coding Standards* constituye igualmente una buena referencia en la que se definen estándares de programación segura para diferentes lenguajes, como Java y C/C++:

<https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards>

The Java rules and recommendations in this site are a work in progress and reflect the current thinking of the secure coding community. Because this is a development website, many pages are incomplete or contain errors. As rules and recommendations mature, they are published in report-as-bugs form as official releases. These releases are issued as dictated by the needs and interests of the secure software development community.

Code is copy-in account if you want to comment on existing content. If you wish to be more involved and directly edit content on the site, you will need an account. But you'll also need to request web publishing.

Front Matter

Rules

Front Matter

Content by tag

Java Coding Rules

Figura 8.26 Buenas prácticas de seguridad para Java según la guía CERT.

## SEI CERT C++ Coding Standard

Creado por: Adrien, modificaciones: Ultima vez: 2020 David Tardito el 05/05/2019

The C++ rules and recommendations in this wiki are a work in progress and reflect the current thinking of the secure coding community. Because this is a development website, many pages are incomplete or contain errors. As rules and recommendations mature, they are published in report or book form as official releases. These releases are issued as dictated by the needs and interests of the secure software development community.

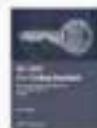
The CERT C++ Coding Standard does not currently expose any recommendations; all C++ recommendations have been removed (moved to The Void section) due to quality concerns pending further review and development.

Create a sign-in account if you want to comment on existing content. If you wish to be more involved and directly edit content on the site, you still need an account, but you'll also need to request edit privileges.

### Front Matter

- Usage
- Tool Selection and Validation
- System Qualities
- 

### Sobre C++ Coding Books and Downloads



The CERT C++ Coding Standard, 2014 Edition provides rules to help programmers ensure that their C++ code reduces security flaws by following secure coding best practices. It is downloadable as a PDF. (errata)



The CERT C++ Coding Standard references and relies on the CERT C

Figura 8.27 Buenas prácticas de seguridad para C++ según la guía CERT y libros recomendados.

## 8.9 Conclusiones

En este libro se han presentado, de manera general, procesos de desarrollo de software que tienen actividades enfocadas a la mejora de la seguridad, siempre recordando que la mejor metodología es aquella que se adapta al contexto del producto que se vaya a desarrollar.

La seguridad, hoy día, se encuentra en cada capa de la arquitectura de software y, por lo tanto, no se puede dejar como un elemento aislado, sino que es transversal y multidisciplinar. Si se continúa haciendo software de la manera tradicional, esa brecha será aprovechada por hackers y expertos en seguridad, que siempre van dos pasos por delante de las organizaciones, y se seguirán explotando vulnerabilidades que pudieron haber sido evitadas utilizando metodologías como las que se comentaron.

Para hacer software seguro, se necesita, en todas las fases del desarrollo, tener presentes dos puntos de vista fundamentales en la ingeniería del software seguro: los usuarios y los atacantes. Los primeros focalizan su atención

en la funcionalidad del sistema y asumen que el sistema es seguro; los segundos, excepto en algunos casos muy específicos, focalizan su atención en las características de seguridad del sistema y asumen que el sistema tiene cierta funcionalidad.

Si los gestores e ingenieros han sido previsores y el sistema informático se basa en software seguro, desarrollado siguiendo los principios de ingeniería del software seguro, un atacante se encontrará con dificultades y con menos vulnerabilidades de lo que es habitual en la actualidad. No podremos eliminar los ataques —estos se seguirán produciendo—, pero, si usamos software seguro y de calidad, nuestro sistema podrá soportar mucho mejor los ataques.

Las experiencias están demostrando que resulta necesaria una disciplina de la ingeniería para comprender en profundidad las cuestiones de la seguridad en el desarrollo de sistemas de información. Conforme las tendencias reseñadas, se puede considerar que nos hallamos ante el surgimiento de lo que algunos investigadores proponen llamar **ingeniería del software seguro**, cuyo alcance comprende, entre otras, a la ingeniería de requerimientos de seguridad, el modelo de seguridad y el desarrollo de software seguro.

Su principal objetivo como campo de investigación es la producción de técnicas, métodos, procesos y herramientas que integren los principios de la ingeniería de seguridad y de calidad, y que permitan a los desarrolladores de software analizar, diseñar, implementar, testear y desplegar sistemas de software seguro.

La programación segura responde al deseo permanente de los desarrolladores por avanzar en medio de la dificultad y confrontar el reto de la perfección. Sabemos que el software libre de errores constituye una meta prácticamente inalcanzable, dado que los buenos hábitos y las prácticas de programación no se han desarrollado suficientemente para avanzar en sólidos desarrollos de sistemas de información vistos tanto desde la perspectiva técnica como desde la óptica de administración y gestión de proyectos. En este sentido, resulta importante que los profesionales más experimentados en el desarrollo de software orienten y comuniquen su práctica en esta disciplina, ofreciendo un contexto práctico para las nuevas generaciones de desarrolladores para que, desde sus inicios, se fomenten buenos hábitos al enfrentarse al reto de un desarrollo.

## Capítulo 9. Glosario de términos

**Análisis dinámico:** A través de la revisión dinámica, se analiza la aplicación sobre un entorno de ejecución de testing lo más parecido posible al de producción. Se probarán los casos de tipo dinámico derivados de los requisitos aplicables que pueden detectar vulnerabilidades y, en función de los resultados de los casos de prueba, se podrá derivar el cumplimiento o no de los requisitos de seguridad.

**Análisis estático:** El análisis estático de seguridad permite, de una manera sencilla y con poco impacto en el equipo, analizar el código fuente para detectar posibles vulnerabilidades durante el desarrollo, antes de que estas se puedan convertir en incidentes de seguridad para la aplicación. Se basa en los estándares de seguridad con OWASP top 10 (data injection, broken authentication and session management, sensitive data exposure, broken access control, security misconfiguration y cross-site scripting [XSS]).

**Buffer overflow:** Constituye un error de software que se produce cuando un programa no controla adecuadamente la cantidad de datos que se copian sobre un área de memoria reservada a tal efecto (búfer). Si dicha cantidad se muestra superior a la capacidad preasignada, los bytes sobrantes se almacenan en zonas de memoria adyacentes y sobreescriben su contenido original; por ejemplo, Java automáticamente comprueba la cantidad de datos y el espacio disponible en el búfer antes de almacenarlos. Si durante la ejecución de código se intentara sobrepasar el tamaño de búfer, se devolvería la excepción `ArrayOutOfBoundsException`.

**Certificado digital o electrónico:** Se trata de un fichero informático generado por una entidad de servicios de certificación que asocia unos datos de identidad a una persona física, organismo o empresa, de modo que se confirme su identidad digital en Internet. El certificado digital resulta válido, principalmente, para autenticar a un usuario o sitio web en Internet, por lo que es necesaria la colaboración de un tercero que sea de confianza para cualquiera de las partes que participen en la comunicación.

**Code Smell:** Se relaciona con un problema asociado a la mantenibilidad en el código. Dejarlo como está significa que, en el mejor de los casos, los mantenedores se enfrentarían a más dificultades de las que deberían al hacer cambios en el código. En el peor de los casos, estarán tan confundidos por el estado del código que introducirán errores adicionales a medida que realicen cambios.

**Cross-site request forgery (CSRF):** Exploit malicioso de un sitio web mediante el cual se puede conseguir que se ejecuten transacciones no autorizadas por la víctima. Al contrario que las vulnerabilidades de XSS, que se aprovechan de la confianza que un browser deposita en el sitio web al que accede, las vulnerabilidades de CSRF aprovechan la confianza que un determinado sitio web deposita en el navegador del usuario ya autenticado.

**CSR (Certificate Signing Request):** Texto cifrado que contiene la solicitud del certificado e incluye la información necesaria para su generación.

**CVE (Common Vulnerabilities and Exposures):** Es una base de datos de información registrada sobre conocidas vulnerabilidades de seguridad, donde cada referencia dispone de un número de identificación único del tipo CVE-2015-6259, donde el segundo término se refiere al año en el que se ha registrado la vulnerabilidad y el tercero indica el número de vulnerabilidad que representa, con base en un contador que se inicia anualmente.

**CVSS (Common Vulnerability Score System):** Se corresponde con un sistema de puntaje diseñado para proveer de un método abierto y estándar que permite estimar el impacto derivado de vulnerabilidades identificadas en tecnologías de información; es decir, contribuye a cuantificar la severidad que pueden representar dichas vulnerabilidades.

**Desarrollo guiado por pruebas de software:** El test-driven development (TDD) es una práctica de programación que involucra otras dos prácticas. La primera, escribir las pruebas (test first development) y la segunda, hacer refactorización (refactoring). Se trata de una práctica de desarrollo en la que se realizan pequeñas pruebas para verificar el comportamiento de una pieza de código escrita antes que el propio código. Las pruebas inicialmente fallan y el objetivo de los desarrolladores consiste en agregar código, para conseguir que estas alcancen el éxito.

**DevOps (“desarrollo” + “operaciones”):** Constituye un proceso fino y ágil por el cual el equipo de una empresa de desarrollo, operaciones de TI y los departamentos de control de calidad colaboran para ofrecer software de una manera continua. DevOps permite a esos departamentos desarrollar y probar el software en contra de sistemas que se comportan como plataformas; por ello, pueden ver cómo se comporta la aplicación y ejecutarse antes de la implementación.

**Ejecución de comandos del sistema operativo:** Resulta similar a la inyección de SQL, en la que determinados sistemas de bases de datos proporcionan un medio para ejecutar comandos a nivel del sistema operativo. Un atacante puede inyectar dichos comandos en una consulta, lo que hace que la base de datos ejecute tales comandos en el servidor, lo que le brinda al atacante privilegios adicionales, hasta e incluyendo el acceso al sistema de nivel raíz.

**Entrega continua:** Representa un conjunto de procesos y prácticas que eliminan gradualmente los desechos de su proceso de producción de software. Permite una entrega más rápida de la funcionalidad de alta calidad y establece un bucle rápido y una efectiva retroalimentación entre su empresa y sus usuarios.

**Escalado de privilegios:** Esto ocurre cuando un ataque aprovecha algún exploit para obtener un mayor acceso. En las bases de datos puede conducir al robo de datos confidenciales.

**Falsificación de solicitudes entre sitios (CSRF):** La falsificación de solicitudes entre sitios involucra a un atacante que crea solicitudes HTTP (webs) basadas en el conocimiento de cómo funciona una aplicación web en particular y engaña a un usuario o navegador para que envíe estas solicitudes. Si una aplicación web es vulnerable, el ataque puede ejecutar transacciones o envíos que parecen provenir del usuario. CSRF se utiliza, normalmente, después de que un atacante ya haya obtenido el control de la sesión de un usuario, a través de XSS, ingeniería social u otros métodos.

**Git:** Se trata de un sistema de control de versiones distribuido con un énfasis en la velocidad, la integridad de los datos y el soporte para flujos de trabajo no lineales. Git fue inicialmente diseñado y desarrollado por Linus Torvalds para el

desarrollo del kernel de Linux en 2005 y se ha convertido en el sistema de control de versiones más ampliamente adaptado para el desarrollo de software.

**GitHub:** Es un servicio de alojamiento basado en la web que ofrece todas las revisiones de control, la gestión de código fuente (SCM) y la adición a sus propias características. A diferencia de Git, una herramienta estrictamente de línea de comandos, GitHub proporciona una interfaz gráfica basada en web y de escritorio, así como la integración móvil.

**Hotspot de seguridad:** Se corresponde con un problema relacionado con la seguridad que resalta un fragmento de código que utiliza una API sensible a la seguridad (por ejemplo, el uso de un algoritmo débil o la conexión a una base de datos sin una contraseña). Los puntos de acceso a la seguridad deben ser revisados por un auditor de seguridad que pueda determinar que las API se utilizan de manera que presenten vulnerabilidades.

**Integración continua:** La integración continua es un modelo informático propuesto inicialmente por Martin Fowler que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes. Entendemos por integración la compilación y ejecución de test de todo un proyecto.

**Inyección de SQL:** Las interfaces que no validan correctamente la entrada del usuario pueden hacer que se inyecte SQL en una consulta de aplicación inocua, lo que provoca que la base de datos exponga o manipule los datos que normalmente deberían estar restringidos por el usuario o la aplicación.

**Jenkins:** Es una herramienta de integración continua de código abierto escrito en Java. Jenkins proporciona servicios de integración continua para el desarrollo de software. Se trata de un sistema en funcionamiento basado en un servidor en un contenedor de servlets, como Apache Tomcat. Resulta compatible con las herramientas de SCM, incluyendo AccuRev, CVS, Subversion, Git, Mercurial, Perforce, Clearcase y RTC, y puede ejecutar proyectos basados en Apache Ant y Maven, así como las secuencias de comandos shell arbitrarios y comandos por lotes de Windows.

**JIRA:** Constituye un producto de seguimiento de tareas, desarrollado por Atlassian. Proporciona seguimiento de fallos, de problemas y las funciones de gestión de proyectos.

**JSON:** Acrónimo de JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque hoy, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.

**Maven:** Se trata de una herramienta de automatización en la construcción de proyectos Java. Maven aborda dos aspectos de la construcción de software: en primer lugar, describe la construcción del software y, en segundo, define sus dependencias.

**Métricas:** Las métricas pueden tener valores o medidas variables a lo largo del tiempo. Ejemplos son el número de líneas de código o la complejidad. Una métrica puede ser cualitativa (proporciona una indicación de calidad en el componente, una EG densidad de líneas duplicadas, una cobertura de línea por pruebas, etc.) o cuantitativa (no da una indicación de calidad en el componente, un EG número de líneas de código, una complejidad, etc.).

**ORM:** El mapeo objeto-relacional (más conocido por su nombre en inglés, *object-relational mapping*) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, usando un motor de persistencia.

**Sandbox:** El código de ejecución y los datos de las aplicaciones se encuentran en memoria principal aisladas entre si; es decir, cada aplicación puede acceder únicamente a sus datos y código de ejecución.

**Secuencias de comandos entre sitios (XSS):** Las secuencias de comandos entre sitios son un ataque que implica la inyección de código JavaScript malicioso en un sitio web. Las páginas que se muestran vulnerables a este tipo de ataque devuelven la entrada del usuario al navegador sin desinfectarla adecuadamente. Este ataque se usa a menudo para ejecutar código automáticamente cuando

un usuario visita una página, consigue tomar el control del navegador de un usuario. Después de que se haya establecido el control del navegador, el atacante puede aprovechar ese control en una variedad de ataques, como la inyección de contenido o la propagación de código malicioso.

**SQL Injection:** Ataque consistente en la inserción de código SQL en un parámetro de la url de la aplicación web que es posteriormente introducido en una consulta en el lado del servidor.

**SSH:** Secure Shell o SSH es un cifrado (encriptado) protocolo de red para iniciar sesiones en máquinas remotas de una forma segura.

**Validación de entrada débil:** Muchos servicios web confían demasiado en la entrada proveniente de aplicaciones móviles, al confiar en la aplicación para validar los datos proporcionados por el usuario final. Sin embargo, los atacantes pueden forjar su propia comunicación con el servidor web u omitir por completo las comprobaciones lógicas de la aplicación, lo que les permite aprovechar la lógica de validación que falta en el servidor para realizar acciones no autorizadas.

**Vulnerabilidad:** Las vulnerabilidades son puntos débiles del software que permiten que un atacante comprometa su integridad, disponibilidad o confidencialidad.

**Vulnerabilidades de la plataforma:** Un atacante puede aprovechar las vulnerabilidades del sistema operativo, el software del servidor o los módulos de aplicación que se ejecutan en el servidor web. Las vulnerabilidades, a veces, se pueden descubrir al monitorizar la comunicación entre un dispositivo móvil y el servidor web, para encontrar puntos débiles en el protocolo o los controles de acceso; por ejemplo, un servidor web mal configurado puede permitir el acceso no autorizado a recursos que normalmente estarían protegidos.

**XML:** Siglas de Xtensible Markup Language ("lenguaje de marcas extensible"); se refieren a un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C), utilizado para almacenar datos en forma legible. A diferencia de otros lenguajes, XML da soporte a bases de datos y es útil cuando varias aplicaciones deben comunicarse entre si o integrar información.

**XSS Reflejado (o no persistente):** Estos bugs aparecen cuando los datos proporcionados por el cliente web, normalmente en parámetros de una petición HTTP, se usan inmediatamente en el lado del servidor para adecuar la página de resultados y devolverla al cliente, con este mismo parámetro en el código de la página web resultante.

<https://dogramcode.com/bloglibros>



Dogram