eman ta zabal zazu

Universidad del País Vasco Euskal Herriko Unibertsitatea

MÁSTER EN

CIBERSEGURIDAD 4.0

TRABAJO FIN DE MÁSTER

GUÍA DE BUENAS PRÁCTICAS DE SEGURIDAD EN EL DESARROLLO DE SOFTWARE DE APLICACIONES WEB

Estudiante	*Martínez Tadeo, Borja*
Directora	*Huarte Arrayago, Maider*
Departamento	**Ingeniería de Comunicaciones**
Curso académico	*2020/2021*

Bilbao, julio 2021

Resumen: Mediante este Trabajo de Fin de Máster (TFM) se ha pretendido crear una guía de buenas prácticas de seguridad en el desarrollo de software, haciendo hincapié en el desarrollo web. Esta guía es el resultado de la necesidad de seguridad en el desarrollo de software de aplicaciones web debido a su baja calidad en cuanto seguridad.

La seguridad en el desarrollo del software es muy importante debido a que muchas funciones son completamente dependientes del software. Un atacante puede aprovecharse de las distintas vulnerabilidades que tiene un sistema y este lucrarse con distintos fines como pueden ser, maliciosos, económicos, espionaje o terrorismo.

En la elaboración de esta guía nos hemos centrado fundamentalmente en la información existente en OWASP Testing Guide, ISO 27001, ISO 27034 y SAFECode. Por otra parte, se ha planteado varios casos prácticos para poner como ejemplo diferentes ataques relacionados con un mal desarrollo de software en aplicaciones web.

Palabras clave: Software, seguridad

Laburpena: Master Amaierako Lan (MAL) honen bidez, softwarearen garapenean segurtasunerako jardunbide egokien gida bat sortu nahi izan da, web-garapena azpimarratuz. Gida hau web-aplikazioen softwarea garatzeko segurtasun-beharraren emaitza da, segurtasun aldetik kalitate txikia duelako.

Softwarea garatzeko segurtasuna oso garrantzitsua da, funtzio asko softwarearen menpe baitaude erabat. Erasotzaile sistema batek dituen kalteberatasun desberdinez balia daiteke, eta dirua hainbat helbururekin atera dezake, hala nola maltzurrak, ekonomikoak, espioitza edo terrorismoa.

Gida hau egiteko, OWASP Testing Guide, ISO 27001, ISO 27034 eta SAFECode webguneetan dagoen informazioan jarri dugu arreta batez ere. Bestalde, zenbait kasu praktiko planteatu dira, web-aplikazioetan softwarea gaizki garatzearekin lotutako hainbat eraso adibide gisa jartzeko.

Gako-hitzak: Software, segurtasuna

Abstract: The aim of this Master's Thesis is to create a guide of good security practices in software development, with emphasis on web development. This guide is the result of the need for security in the development of web application software due to its low quality in terms of security.

Security in software development is very important because many functions are completely dependent on software. An attacker can take advantage of different vulnerabilities in a system and profit from them for different purposes such as malicious, economic, espionage or terrorism.

In the elaboration of this guide we have focused mainly on the existing information in OWASP Testing Guide, ISO 27001, ISO 27034 and SAFECode. On the other hand, several practical cases have been presented to give as an example different attacks related to a bad software development in web applications.

Keywords: Software, seguridad

Índice general

Índice de figuras

Índice de tablas

1. Memoria

1.1. Introducción

En la actualidad, los sistemas informáticos procesan, manejan y almacenan información confidencial de alto valor y están presentes en nuestro día a día, desde aplicaciones bancarias a aplicaciones médicas, que contienen información tan personal como nuestro estado de salud. Por ello, es necesario establecer unos patrones de seguridad durante el desarrollo de estas aplicaciones y así intentar evitar posibles ataques informáticos.

A la hora de entregar un producto a un cliente, muchas veces prima antes la funcionalidad que la seguridad de un producto, dar finalizado el producto una vez haya llegado al cliente y se haya sentido satisfecho. Cuando un producto sale al mercado y funciona correctamente, pero no es seguro, no puede decirse que dispone de la calidad suficiente. Esto hace que no tenga la calidad suficiente y el cliente no esté satisfecho.

Como resultado de este Trabajo de Fin de Máster se ha realizado una guía de buenas prácticas de seguridad para el desarrollo de software con base en una metodología de desarrollo seguro como las guías OWASP sobre pruebas [15], el estándar internacional ISO 27001 [7] sobre los sistemas de gestión de seguridad de la información de seguridad, la ISO 27034 [11] que es el estándar internacional para la seguridad en las aplicaciones y las guías de código seguro de SAFECode [12]. A su vez y para explicar diferentes vulnerabilidades que se suelen dar en el desarrollo de software, se han diseñado diferentes casos prácticos para los casos de: *Client-side security only*, inyección SQL e inclusión de archivos locales.

1.1.1. Definiciones, acrónimos y abreviaturas

- **OWASP:** Open Web Application Security Project
- **XSS:** Cross-site scripting
- **SQL:** Structured Query Language
- **Cookies:** Son ficheros de datos que guardan información de un sitio web para que la experiencia de navegación sea mas sencilla, pueden ayudar a recordar inicios de sesión o bien la reproducción de un vídeo de internet.
- **DoS:** Denial of Service.
- **VPN:** Virtual Private Network.
- **SGSI:** Sistema de Gestión de Seguridad de la Información
- **Exploit:** Exploit es una palabra inglesa que significa explotar diferentes vulnerabilidades de un sistema.

1.2. Contexto

En este apartado hablaremos sobre lo que es un desarrollo de software seguro partiendo de la experiencia profesional adquirida en una empresa dedicada al desarrollo de software. Partiendo de esta experiencia, surge la necesidad de crear una guía de desarrollo de software seguro.

1.2.1. ¿Que es el desarrollo de software seguro?

El desarrollo de software seguro supone el tener en cuenta la seguridad desde el inicio de ciclo de vida del proyecto. Muchos desarrolladores de software, no disponen de las habilidades de seguridad, debido muchas veces a la falta de formación en esta.

Por ello, es necesario disponer de algunas metodologías para poder utilizarlas durante el desarrollo del producto y este cuando salga al mercado sea un software seguro.

1.2.2. Ataques relacionados con un mal desarrollo de software

- **Client side security only**
  Un ataque *Client side security only*, es un ataque que se puede dar cuando solo hay seguridad del lado de cliente. Esto es, solo se comprueban los datos que se van a enviar a un servidor en el lado del cliente, pero no se validan en el lado del servidor. Este tipo de ataques ocurren tanto en aplicaciones web como en aplicaciones de escritorio que tienen una conexión a un servidor, donde los datos solo se validarían en la aplicación de escritorio.

  Uno de los ejemplos mas comunes en este tipo de ataques, es cuando desde un navegador web se intenta enviar una serie de datos en un formulario, pero estos solo se comprueban en el lado del cliente, por lo que esto daría lugar a otro tipo de ataques.

- **XSS**
  Un ataque XSS o *Cross-site scripting* [1], es un tipo de ataque donde los atacantes usan código malicioso en páginas web y aplicaciones de confianza que estas a su vez, instalan *malware* en los navegadores de los usuarios. Esto se realiza con el uso de secuencias de código en sitios cruzados.

  Otro posible ataque XSS [2] es falsificando una página web que el usuario final quiera acceder y este acceda mediante ingeniería social y de esta forma, una vez el usuario ingrese los datos en la página falsificada, obtener las *cookies* de la página original para así acceder con sus datos.

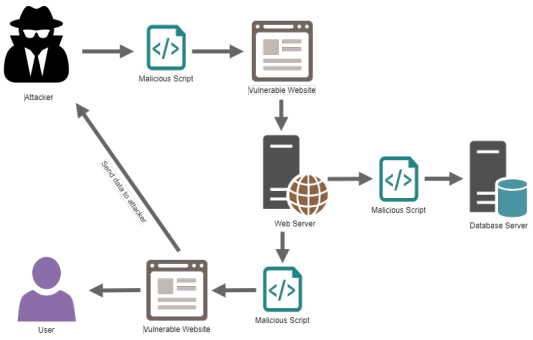


Figura 1: Ataque XSS [2]

Este tipo de ataque tiene dos formas de actuar, o bien de formar reflejada o de forma almacenada. Vamos a explicar ambos tipos de formas:

- **Reflejada**
  Este tipo de ataque XSS se basa en modificar los valores de una aplicación web para pasar valores entre dos páginas. El ejemplo más común para este tipo de ataques es el de intentar robar las *cookies* de un usuario. El método más común utilizado por los cibercriminales es usando ingeniería social, como enviar un correo electrónico engañoso, hacer clickar al usuario en cierto enlace para producir dicho robo.
- **Almacenada**
  A diferencia de la manera reflejada, esta manera consiste en insertar código HTML en sitios que lo permitan, por ejemplo, en formularios. Otra técnica es insertar código JavaScript en campos, como un formulario HTML y estos hagan otro tipo de funcionalidad menos para la que estuviesen preparados.

- **Inyección SQL**
  Un ataque por inyección de SQL [3], lenguaje de consultas de bases de datos relacionales, este ataque permite al atacante inyectar código SQL en un aplicación web. De esta forma, el atacante obtiene acceso a los datos de la base datos y así poder obtener usuarios, contraseñas o lo que sería peor, obtener el usuario y contraseña del administrador de la aplicación. Existen multitud de métodos para realizar una inyección SQL. Vamos a explicar los distintos tipos de formas que existen:

  - **Confidencialidad**
    Dado que las bases de datos SQL generalmente contienen datos confidenciales, la perdida de confidencialidad es un gran problema con los ataques de inyección SQL.
  - **Autenticidad**
    Si el desarrollador utiliza sentencias SQL deficientes para verificar los nombres de usuario y contraseña de un sistema, es posible que se conecte a un sistema como otro usuario sin tener conocimiento previo de la contraseña.

- **Autorización**
  Si la información de autorización se mantiene en una base de datos SQL, es posible cambiar esta información mediante inyección SQL.
- **Integridad**
  Si es posible leer información confidencial de la base de datos, también es posible realizar cambios o incluso eliminar dicha información con un ataque de inyección SQL.

- **Inclusión de archivos locales**
  El ataque de inclusión de archivos [6] permite a un atacante incluir un archivo, de esta forma le permite explotar los mecanismos de inclusión de archivos implementados en la aplicación. Esto puede conducir a:

  - Ejecución de código en el servidor
  - Ejecución de código en el lado de cliente, como código JavaScript, esto puede dar a otros ataques como él (XSS) *Cross-site Scripting*
  - Divulgación de información confidencial

  Uno de los casos más graves es poder acceder a los archivos de un servidor es, en un sistema Linux, donde los usuarios del sistema se almacenan en */etc/passwd*, por lo que estas estarían al descubierto del atacante y este podría iniciar sesión habiendo descubierto este fichero de usuarios.

1.2.3. Normas y guías de seguridad

- **OWASP Testing guide**
  La fundación OWASP [14] es una organización sin ánimo de lucro que apoya y gestiona los proyectos e infraestructura de OWASP. OWASP está formado por empresas, organizaciones educativas y particulares de todo el mundo. Todos ellos forman una comunidad de seguridad informática que trabaja para crear artículos, metodologías, guías, documentación, herramientas y tecnologías que se liberan y pueden ser usadas gratuitamente por cualquier persona.

Figura 2: OWASP [10]

- **ISO 27001**
  La norma ISO 27001 [7] es una norma internacional que profundiza en el aseguramiento, confidencialidad e integridad de los datos, así como de los sistemas que los procesan. La norma ha sido publicada por ISO (Organización Internacional de Normalización).

El objetivo de la norma ISO 27001 es ayudar a las organizaciones la evaluación de los riesgos y la aplicación de los controles necesarios para reducirlos o eliminarlos. La norma se divide en los siguientes capítulos, los 3 primeros son comunes en todas las normas ISO (Objeto y campo de aplicación, Referencias Normativas y Términos y Definiciones), por lo que solo se detallara a partir del capítulo 4:

4. **Contexto de la Organización:** Este apartado recoge indicaciones sobre el conocimiento de la organización y su contextos y el alcance del SGSI.
5. **Liderazgo:** Se destaca la necesidad de que todos los empleados han de contribuir al establecimiento de la norma. Para ello la alta dirección ha de demostrar su liderazgo y compromiso, ha de elaborar una política de seguridad que conozca toda la organización y ha de asignar roles, responsabilidades y supervisores dentro de la misma.
6. **Planificación:** Esta es una sección que pone de manifiesto la importancia de la determinación de los riesgos y oportunidades a la hora de planificar un SGSI, así como de establecer objetivos de seguridad y como lograrlos.
7. **Soporte:** En esta cláusula la norma señala que para el buen funcionamiento del SGSI la organización debe contar con los recursos necesarios.
8. **Operación:** Para cumplir con los requisitos de seguridad, este apartado de la norma indica que se debe planificar, implementar y evaluar los procesos de la organización, hacer una evaluación de los riesgos de la seguridad y un tratamiento de ellos.
9. **Evaluación del Desempeño:** En este punto se establece la necesidad y forma de llevar a cabo el seguimiento, la medición, el análisis, la evaluación, la auditoría interna y la revisión por la dirección del SGSI, para asegurar que funciona todo según lo establecido.
10. **Mejora:** En el último apartado, encontramos las obligaciones que tendrá una organización cuando encuentre una no conformidad y la importancia de mejorar continuamente la conveniencia, adecuación y eficacia del SGSI.

- **ISO 27034**
  La norma ISO 27034 [11] profundiza en el área de la tecnología de la información, técnicas de seguridad y seguridad de aplicaciones. La norma ha sido publicada por ISO (Organización Internacional de Normalización).

  El objetivo de la norma ISO 27034 es garantizar que los programas informáticos se aseguren de un nivel de seguridad de la organización, abordando la mayoría de riesgos de seguridad de las aplicaciones.

  Esta norma proporciona información sobre como especificar, diseñar e implementar diferentes controles de seguridad. Además, añade algunos aspectos sobre cómo determinar diferentes características de seguridad, protección y como prevenir el uso no autorizado de aplicaciones.

Por otra parte, la norma no proporciona ninguna directriz sobre seguridad física ni especificaciones de codificación segura para lenguajes de programación, tampoco proporciona controles de seguridad física.

- **SAFECode**
  SAFECode [17] es una organización sin ánimo de lucro que reúne a empresas líderes mundiales para intercambiar conocimientos e ideas sobre el desarrollo y la mejora de programas de seguridad de software.

  En SAFECode se pueden encontrar guías dedicadas a diferentes temáticas dirigidas al desarrollo de software y metodologías ágiles. Nos ha sido de gran particular ayuda las guías dedicadas a las buenas prácticas a la hora de desarrollar software seguro *Fundamental Practices for Secure Software Development*.

- **Comparativa**

Guía / Característica	OWASP	ISO 27001	ISO 27034	SAFECode
Métodos de trabajo	Sí	Da indicaciones	Da indicaciones	Si
Vulnerabilidades de código	Sí	No	No	Indica buenas prácticas de código
Métricas de seguridad	No	No	No	No

Tabla 1: Tabla comparativa de guías

1.3. Objetivos y alcance el proyecto

1.3.1. Objetivos

En este capítulo se definen el objetivo principal que se ha tenido en cuenta para la realización de proyecto, además de delimitar su alcance para delimitar los propósitos de éste.

El principal objetivo de este trabajo de fin de máster, es conseguir la realización de una guía de desarrollo de software seguro para aplicaciones web y la realización de diferentes casos prácticos que muestren diferentes ataques relacionados con un mal desarrollo de software en este tipo de aplicaciones. Por ello, se identifican los siguientes objetivos secundarios:

- **Elaboración de una guía en base a las metodologías de seguridad existentes**
  La elaboración de una guía en base a distintas metodologías de seguridad existentes en el mercado. Así mismo, esta guía luego pueda servir a desarrolladores de software y puedan implementar un software seguro durante todo el ciclo de vida de éste.

- **Elaboración de diferentes casos prácticos de fallos de seguridad relacionados con un mal desarrollo de software**
  Dado que existen multitud de ataques relacionados con un mal desarrollo de software, se realizarán distintos casos prácticos a tener en cuenta a la hora de desarrollar una aplicación y estos puedan ser solventados en caso de su existencia.

El alcance de este Trabajo de Fin de Máster se ha delimitado teniendo en cuenta las siguientes premisas en su realización:

- Dado que este trabajo tiene por objetivo el desarrollo de una guía de desarrollo de software seguro para aplicaciones web, no se profundizará en aspectos técnicos relativos a los diferentes lenguajes y *frameworks* de desarrollo existentes para este tipo de aplicaciones.
- Debido a que no hay presupuesto para la realización del trabajo, se trabajará a partir de la información tratada por terceros mencionando guías y normativas existentes de acceso público.

1.4. Beneficios

Los propios beneficios que pueda traer este proyecto son los que generan un interés en la ejecución del mismo, por ello, se hace un especial hincapié en este apartado, donde se resaltaran los distintos beneficios. A continuación, se separaran los distintos beneficios detallándolos con más precisión en distintos puntos.

En dichos puntos, vamos a hablar sobre los diferentes beneficios que puede tener un desarrollo de software seguro, tanto beneficios técnicos como económicos para cualquier organización.

1.4.1. Beneficios técnicos

Este proyecto será interesante desde el punto de vista técnico, puesto que enseña a futuros desarrolladores como implementar a partir de una guía de desarrollo de software seguro y con un alto grado de seguridad, de tal forma que cuando salga a producción sea un software seguro. Además, se presentan unos casos prácticos sobre distintos tipos de ataques y así estos puedan ayudar a resolver a futuros desarrolladores.

1.4.2. Beneficios económicos

Este proyecto es interesante desde el punto de vista económico, puesto que reportará beneficios económicos directos e indirectos. También habrá varios interesados desde el punto de vista económico; es decir, el cliente y el propio desarrollador.

Primero, por una parte, desde el punto de vista del cliente, dispondrá de un software que contendrá el menos número de vulnerabilidades posibles. Los incidentes de seguridad, provocan grandes pérdidas en las empresas industriales y tecnológicas [8], puesto que pueden poner en jaque el nivel productivo de la

empresa, sus datos, la confianza en sus clientes y proveedores...

Todo lo que incremente la protección a nivel de seguridad informática de la organización, puede generar una mayor estabilidad laboral y productiva, por lo cual, un mayor flujo beneficio económico futuro.

Por otra parte, desde el punto de vista del desarrollador de software, generara un beneficio económico desde el momento en el que este venda el producto y la implementación. Además, el beneficio económico también mejorara de cara al futuro, haciendo que futuros clientes quieran tener un software seguro como el de sus competidores.

1.5. Análisis de alternativas

Este proyecto presentara diversas variantes para la solución a nivel técnico. Por ello, es totalmente necesario un análisis exhaustivo de las diferentes metodologías de desarrollo de software seguro que existen, para poder desarrollar una guía basada en una argumentación firme y un trabajo de investigación serio.

- **S-SDLC (Secure Software Development Life Cycle)**
  S-SDLC [9] es el conjunto de principios de diseño y buenas prácticas a implantar para detectar, prevenir y corregir los defectos de seguridad en el desarrollo de software, de forma que se obtenga software de confianza y robusto frente ataques informáticos. No tiene una fecha concreta de creación, Elliott & Strachan & Radford sostienen que se origino en 1960 debido al impacto que comenzaban a tener los sistemas de información en la vida moderna.

  De tal forma que el software, solo realice las funciones para las que fue diseñado, que esté libre de vulnerabilidades, ya sean intencionalmente diseñadas o accidentalmente insertadas durante su ciclo de vida y se asegure su confidencialidad, disponibilidad e integridad.

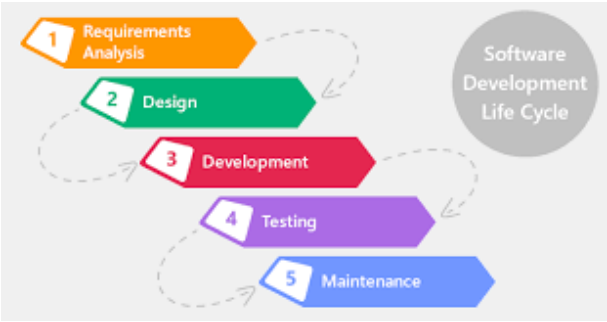


Figura 3: Secure Software Development Life Cycle [9]

- **Análisis:** Es la primera etapa del desarrollo del software, donde se identifican y recogen las características.

- **Diseño:** Es la etapa donde se valora el análisis funcional de la propuesta y la arquitectura propuesta. En esta parte, se definen los posibles ataques que puede sufrir nuestra infraestructura, por ello la seguridad debe formar parte del diseño del propio proyecto.
- **Desarrollo:** En esta fase se implementa la propuesta con todas las especificaciones recogidas en la parte del diseño. Ahora se tiene en cuenta el uso adecuado del entorno de desarrollo, el uso de buenas prácticas en la implementación y el cumplimiento de los términos de seguridad. Por otra parte, se realizan auditorías de código para evitar posibles desviaciones respecto a lo que estaba propuesto.
- **Pruebas:** En la etapa de realización de pruebas, se siguen teniendo en cuenta las distintas auditorías de código. Además, se realizan las pruebas de comportamiento de la aplicación para que la aplicación solo haga las funciones que tenía previsto realizar.
- **Fase de mantenimiento del software:** La fase de mantenimiento se encarga de la monitorización preventiva y reactiva frente a incidentes de seguridad de la aplicación.

- **CLASP (Comprehensive Lightweight Application Security Process)**
  CLASP [10] es un conjunto de componentes de proceso basado en roles y actividades guiado por las mejoras prácticas. Fue diseñado por OWASP para ayudar a los desarrolladores de software para incorporar seguridad en las primeras etapas de los ciclos de vida en los desarrollos de software nuevos y existentes.

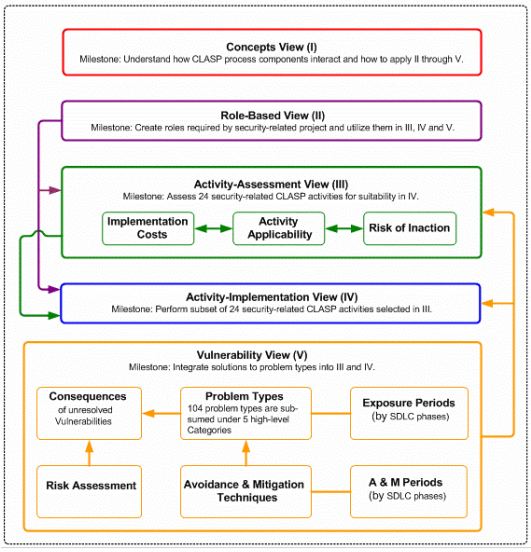


Figura 4: Comprehensive Lightweight Application Security Process [10]

CLASP se presenta en cinco vistas, estas son las 5 vistas:

(I) **Vista de conceptos:** Proporciona una introducción a CLASP, describe brevemente la interacción con las otras vistas de CLASP, mejores prácticas de CLASP y la relación de CLASP con las políticas de seguridad.

(II) **Vista basada en roles:** Este apartado contiene introducciones basadas en roles al método CLASP y proporciona una vista de alto nivel para jefes de proyecto de cómo ellos y su equipo de proyecto deben abordar los problemas de seguridad. Este apartado también introduce las responsabilidades básicas que tienen.

(III) **Vista de actividad-evaluación:** Esta vista ayuda a los jefes de proyecto a evaluar las actividades del CLASP.

(IV) **Vista de actividad-implementación:** Esta vista de CLASP contiene las actividades de CLASP relacionadas con la seguridad, pudiendo integrarlas en un proceso de desarrollo de software.

(V) **Vista de vulnerabilidades:** Esta vista contiene una lista de vulnerabilidades subyacentes identificados por CLASP que forman la base de las vulnerabilidades de seguridad en el código de una aplicación.

- **SSDF (Secure Software Development Framework)**

  SSDF [6] es un conjunto de prácticas de desarrollo de software basadas en documentos de prácticas de desarrollo de software seguro establecidos en organizaciones como BSA, OWASP y SAFECode. Estas organizaciones, son organizaciones sin ánimo de lucro que ayudan mediante sus guías a los desarrolladores a crear aplicaciones seguras.

  En la actualidad, existen pocos modelos de ciclos de vida de software que aborden específicamente la seguridad del software en detalle, por lo que las metodologías como SSDF deben agregarse e integrarse con cada implementación de cada ciclo de vida de software. Las características clave del SSDF son:

  - Definir criterios para las comprobaciones de seguridad en el software.
  - Proteger los accesos de código no autorizado. Usar el método del principio de privilegios mínimos. El fin es proporcionar el menor número de privilegios para posteriormente ir escalándolos, siendo el permisor de administrador el más alto de todos.
  - Usar firma digital para proporcionar un mecanismo de integridad del software en el ciclo de vida.
  - Realizar un diseño de software para cumplir con los requisitos de seguridad y minimizar riesgos.
  - Verificar que el software de terceros cumpla con los requisitos de seguridad establecidos por los desarrolladores.
  - Analizar el código para detectar vulnerabilidades y verificar el cumplimiento de los requisitos de seguridad establecidos.
  - Verificar el cumplimiento de los requisitos de seguridad.
  - Configurar el software, de tal forma que tenga una configuración segura determinada.

- Identificar, analizar y solventar vulnerabilidades de seguridad de forma continua.

- Comparativa

Característica \ Metodología	S-SDLC	CLASP	SSDF
Es cíclico	Sí	Sí	No
Utiliza diferentes guías	No	No	Si
Métricas de seguridad	No	No	Si

Tabla 2: Tabla comparativa de guías

El sistema para elegir la alternativa a seguir en la solución ha sido elegir la metodología que más 'Sí' tuviese. A partir del sistema de puntuación seguido, la metodología mas completa en cuanto a las características descritas en la tabla es SSDF.

1.6. Análisis de riesgos

En este apartado se recogen todos los posibles riesgos y problemas que pueden surgir a lo largo de todo el proyecto. También, se recogen posibles soluciones y planes de prevención que podemos realizar en caso de que se produzca algún fallo. Por otro lado, también se recogen las probabilidades de que suceda el fallo así como el impacto que tendría en el proyecto. Para cuantificar la probabilidad y el impacto de cada riesgo se evaluaran de la siguiente manera.

1. Probabilidad
   1.1. Baja($<30\,\%$)
   1.2. Media ($30 < \,\% \leq 70$)
   1.3. Alta ($>70\,\%$)
2. Impacto
   2.1. Bajo
   2.2. Medio
   2.3. Alto

1.6.1. Enfermedad

Descripción	Aparición de una enfermedad que no permita continuar con el proyecto
Prevención	Llevar el proyecto al día o adelantado para superar cualquier contingencia
Plan de contingencia	Invertir mas horas en el proyecto.
Probabilidad	Baja
Impacto	Medio

Tabla 3: Riesgo 1: Enfermedad

1.6.2. Rotura del equipo de trabajo

Descripción	El equipo de trabajo se averíe durante la realización del proyecto
Prevención	Llevar el mantenimiento correcto del equipo
Plan de contingencia	Reparar el equipo o sustituirlo por otro
Probabilidad	Baja
Impacto	Baja

Tabla 4: Riesgo 7: Rotura del equipo de trabajo

1.7. Descripción de la solución

En este apartado, se presenta una guía de desarrollo de software seguro. Se pretende que esta guía pueda servir a desarrolladores de software a programar aplicaciones que sean seguras cuando salgan al mercado o se desplieguen en producción.

1.7.1. Concepto de software seguro

Antes de desarrollar la guía, tenemos que tener claro cual es el concepto de un software seguro. Para ello, lo vamos a dividir en: Confidencialidad, Integridad, Disponibilidad y Autenticidad.

- **Confidencialidad**
  La confidencialidad es la propiedad que no permite divulgar datos e información de individuos y organizaciones. Mediante esta propiedad se asegura que la información es solo accesible por aquellas personas que tienen autorización.
  Uno de los métodos más conocidos de atacar esta propiedad, es cuando se dan a conocer los datos de una base de datos que ha sido atacada.

- **Integridad** La integridad es la propiedad que trata de impedir que los datos no se han modificado sin ninguna autoridad. Es decir, la integridad lo que pretende es mantener los datos con la misma exactitud que se

añadieron en su momento y estos se mantengan así, hasta que se autorice la modificación o eliminación.
Uno de los métodos más comunes de vulnerar esta propiedad, es cuando el objetivo de un atacante es tratar de modificar los datos. Normalmente se suele realizar mediante ataque de inyección SQL, intentando modificar o eliminar los datos.

- **Disponibilidad** La disponibilidad es la propiedad que trata de garantizar en todo momento el acceso a los datos. Es decir, los datos deben de estar accesibles en todo momento que una identidad autorizada quiera acceder a ellos.

  El método más común para vulnerar la propiedad de disponibilidad son los ataques de denegación de servicios (DoS) y ataques de denegación de servicios distribuidos (DDoS). Garantizar la propiedad de disponibilidad es prevenir este tipo de ataques disponiendo de distintos servidores alternativos para entrar en funcionamiento en caso de estos ataques.

- **Autenticidad** Es la propiedad que permite identificar a una persona o entidad para acceder a los datos. La autenticidad en los sistemas informáticos se consigue mediante usuarios, contraseñas y si se puede, reforzarla con autenticidad de doble factor. Otra forma de garantizar la autenticidad es mediante sistemas biométricos, los cuales permiten mayor seguridad, como pueden ser:

  - **Huella dactilar:** Reconocimiento a través de la huella dactilar de la persona.
  - **Reconocimiento de la voz:** Reconocimiento a partir de la voz de la persona.
  - **Reconocimiento facial:** El reconocimiento facial de la persona.

1.7.2. Fases a la hora de desarrollar software seguro

A la hora de desarrollar un software hay diferentes fases diferenciadas en él, desde la recogida y el análisis de requisitos a las pruebas de la aplicación. Estas fases vienen descritas en las alternativas encontradas en el análisis de alternativas y que son comunes en los proyectos de desarrollo de software.

A continuación, se van a detallar distintos aspectos que hay que tener en cuenta por cada fase.

1. **Análisis de requisitos**
   Durante esta fase primera fase del desarrollo de software, se realiza la captura de requisitos y el análisis para un posterior desarrollo. A la hora de analizarlos, hay que tener en cuenta tanto los requisitos funcionales como los requisitos de seguridad para entregarle al cliente una aplicación funcional y segura a la vez.

2. **Desarrollo**
   La fase de desarrollo es la etapa en la que se construye la aplicación. La complejidad y la duración de esta etapa no es exacta, ya que viene

directamente ligada a los lenguajes de programación que se utilizan y la complejidad del proyecto.

- **Elección del lenguaje y tipo de base de datos**
  Esta elección depende mucho del sistema que se quiera desarrollar. Por una parte, hay que decidir si se quiere almacenar los datos en una base de datos relacional o no relacional. En este caso, los lenguajes son distintos ya que una base de datos relacional SQL tiene un lenguaje propio (SQL), en cambio, una base de datos no relacional el lenguaje o sentencias a utilizar se rigen por el sistema de gestión que se elija: MongoDB, Google Firebase... lo cual hay que tener en cuenta a la hora de evitar ataques tipo SQL Injection.

  Por otra parte, el lenguaje de programación a elegir para la aplicación y los *framework* a utilizar. Esto hay que tenerlo en cuenta para el futuro de la aplicación y las futuras actualizaciones de seguridad de los lenguajes y del propio *framework* a utilizar. Ya que no es lo mismo, usar un lenguaje que se esta quedando obsoleto a usar un lenguaje que es mas reciente y seguramente vaya a recibir mas actualizaciones de seguridad para prevenir futuros ataques.
- **Autenticación**
  La importancia de vincular una unidad del sistema a un usuario o entidad mediante el uso de un usuario y contraseña. De esta forma, se proveen controles de autenticación de acuerdo con el riesgo de la aplicación y los requisitos del cliente y así, denegamos el acceso a los diferentes atacantes que intentar atacar el sistema de autenticación.

  Uno de los mejores métodos actuales para autenticar a una persona en un sistema, es la autenticación mediante doble factor. Así mismo, nos aseguramos de que la persona que se esta autenticando dice ser quien es aunque otra persona conozca su contraseña, ya que la persona que esta intentando iniciar sesión, debe tener con ella un código de seguridad que se actualiza cada vez que esta quiere iniciar sesión.
- **Integridad de los datos**
  Además de estar autenticados en los sistemas para poder acceder a los diferentes datos, cuando disponemos de datos almacenados en una base de datos, tenemos que asegurarnos de salvaguardar la integridad de estos.

  Debemos de asegurar que se usa algún tipo de cifrado, como por ejemplo: RSA, de manera que, se pueda proteger la confidencialidad e integridad de los datos sensibles de los usuarios, como pueden ser contraseñas o datos sensibles personales de los usuarios.
- **Pruebas**
  A la hora de probar el software implementado, hay que realizar diferentes tipos de pruebas para así establecer que el software se ejecuta correctamente y no proporciona resultados inesperados. Entre otros, hay que realizar pruebas unitarias, pruebas de bibliotecas, pruebas de integración y pruebas de sistemas.

- Pruebas unitarias: Son aquellas pruebas que sirven para comprobar que funciona correctamente una parte del código. La idea de estas pruebas es escribir un caso de prueba para cada método, de forma que cada caso sea diferente al resto.
- Pruebas de bibliotecas: En un software se utilizan diferentes bibliotecas, desde bibliotecas para la gestión de la conexión de la base de datos a bibliotecas para usar una cámara web. Por ello, es importante realizar pruebas con este tipo de bibliotecas para proporcionar un software seguro, ya que muchas veces se utilizan bibliotecas de terceros, como repositorios de GitHub, que no han sido verificadas.
- Pruebas de integración: Es importante, que una vez se pasa del sistema de desarrollo al de integración, realizar las pruebas pertinentes de la aplicación para no tener ni fallos de seguridad ni de ejecución. Siempre hay que realizar este tipo de pruebas a pesar de que ambos sistemas sean iguales, siempre puede haber cambios menores en ambos sistemas.
- Pruebas de sistemas: Realizar las pruebas pertinentes cuando la aplicación se vaya a instalar en los sistemas de producción y estos no tengan ningún fallo ante el despliegue.

- **Manejo de las excepciones**
  El manejo de excepciones es una técnica de desarrollo que permite al desarrollador controlar los diferentes errores que se pueden ocasionar durante la ejecución de una aplicación. Cuando ocurre cierto tipo de error, el sistema reacciona ejecutando un fragmento de código que resuelve la situación, normalmente, devolviendo un mensaje de error que es personalizado por el programador para así proveer una solución.

3. **Auditoría de seguridad**

   - **Análisis de caja blanca**
     Los análisis de caja blanca son muy eficaces en validación de las decisiones de diseño y los supuestos y en la búsqueda de errores de programación y errores de implementación.

     Los análisis de caja blanca realizan validaciones si la aplicación del código sigue el diseño previsto, para validar la funcionalidad de seguridad implementada, y para descubrir vulnerabilidades que se puedan explotar.
   - **Análisis de caja negra**
     Los análisis de caja negra son métodos que no requieren acceso al código fuente. La persona encargada de realizar las pruebas no tiene el acceso al código de la aplicación. Como consecuencia de ello, los análisis de caja negra se centran en el comportamiento externo del software, tales como los requisitos, especificaciones de protocolos, intentos de ataques etc.

     Dentro de estas pruebas de seguridad, los análisis 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 implementado.

- **Google hacking**
  Google Hacking es una técnica de *hacking* utilizada para filtrar información mediante el buscador de Google, se trata de realizar búsquedas avanzadas mediante el buscador de Google. El uso de esta herramienta se debe a que contiene diversa cantidad de comandos conocidos [16] para la filtración de búsquedas.
  Esta herramienta es muy útil de usar, para ponerse en la piel de un atacante y poder realizar diferentes acciones como: obtener fichero de usuarios y contraseñas, obtener bases de datos volcadas y poder hacer modificaciones o la detección de la versión del sistema operativo de un servidor.
- **Metasploit**
  Metasploit es una herramienta de seguridad que está enfocada a auditores de seguridad. Es una herramienta muy completa que tiene dispone muchísimos *exploits* con vulnerabilidades conocidas, junto con los llamados *payloads*, que son los códigos que explotan estas vulnerabilidades. Esta herramienta es de gran utilidad sobretodo para intentar explotar diversas vulnerabilidades que pueda contener el servidor donde se aloje la aplicación desplegada y en el caso de que las tengan, intentar solventarlas con una actualización.

4. **Mantenimiento** Una vez puesto en producción el software, hay que mantenerlo. Para ello, hay que tener unos controles dedicados al buen funcionamiento el sistema una vez puesto en producción. Cada año salen a la luz miles de vulnerabilidades en todos para los sistemas. Con el fin de identificar vulnerabilidades, existe un estándar para identificarlas:

   - **CVE - Common Vulnerabilities and Exposures**
     Es un identificador estándar para las vulnerabilidades, se trata de un código para cada vulnerabilidad existente
   - **CPE - Common Platform Enumeration**
     Es un identificador estándar para cada plataforma, se trata de un código único para cada plataforma existente
   - **Fuente**
     Quien identifica la vulnerabilidad
   - **Descripción**
     Daño que puede provocar la vulnerabilidad
   - **Solución**
     Solución dada por la fuente para proteger nuestros sistemas.

1.8. Pruebas y casos prácticos

En los siguientes casos prácticos se va a demostrar diferentes tipos de ataques informáticos relacionados con un mal desarrollo de software seguro. Se viene a demostrar la importancia de tener que realizar pruebas informáticas y que los sistemas funcionan correctamente antes de entregarlos a un cliente.

1.8.1. Client side security only

Este tipo de problemas se da a la hora de rellenar cualquier formulario y sólo se validan los datos del lado del cliente. Este sería un caso muy común de realizar un mal desarrollo de software seguro, ya que un usuario normal no lo sabría, pero un usuario técnico puede probar diferentes técnicas y así poder explotar la vulnerabilidad.

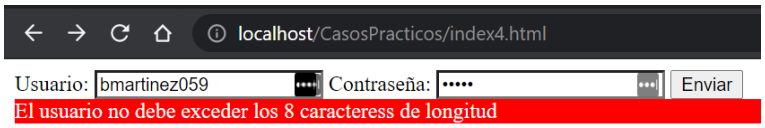


Figura 5: Seguridad del lado del cliente

En la imagen, podemos observar que un usuario ha intentado iniciar sesión y se le ha denegado debido a que el usuario excede los 8 caracteres de longitud. Si analizamos la petición que se realiza cuando no se excede la longitud, podemos recoger la cURL, editarla y enviarla mediante el bash o cmd para ver que ocurre si se excede dicha longitud.

```
  curl 'http://localhost/CasosPracticos/simplepost.php' \
-H 'Connection: keep-alive' \
-H 'Cache-Control: max-age=0' \
-H 'Content-Type: application/x-www-form-urlencoded' \
-H 'Referer: http://localhost/CasosPracticos/index4.html' \
-H 'Accept-Language: es-ES,es;q=0.9' \
--data-raw 'usuario=bmartinez059&pass=password' \
--compressed
```

Una vez cambiado el usuario a uno de mayor longitud y este enviado por *bash*, ocurriría lo siguiente:

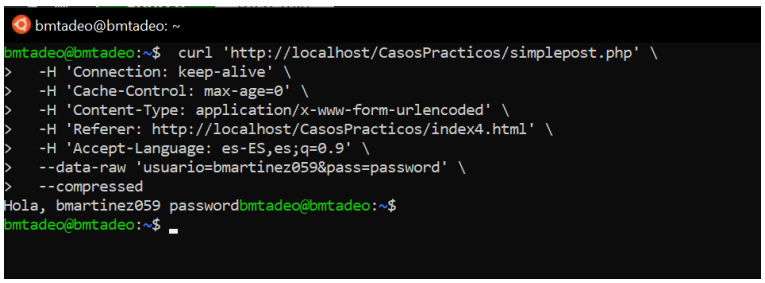


Figura 6: Envío de cURL mediante bash

El servidor, al no validar la longitud del campo del usuario nos muestra el valor enviado, el cual tiene una longitud mayor a 8. El uso más común de estas validaciones es a la hora de crear contraseñas o validarlas. Al usuario siempre se le da el feedback en el lado del cliente, pero el usuario no sabe que es lo que

ocurre más allá de este, por lo que ante posibles ataques siempre hay que validar los datos más de una vez, tanto en el cliente como en el servidor.

- **Cómo solucionar el problema**
  La solución al problema que había que realizar es validar los campos que se envían por el formulario también en el lado del servidor. De esta forma si se realizase la misma operación este no tendría que mostrar el nombre del usuario en la consola bash.

1.8.2. Inyección SQL

Este tipo de problemas se da a la hora de rellenar cualquier formulario web o en una aplicación de escritorio que tenga una conexión a una base de datos.

En esta vulnerabilidad se pueden realizar diferentes acciones, desde añadir una sentencia SQL que nos devuelva todos los usuarios de la base de datos a poder eliminarla, restringir los permisos de la base de datos para añadirnos permisos de administrador entre otros.

En la siguiente imagen, podemos observar cómo en una página web de inicio de sesión, un usuario inserta lo siguiente:

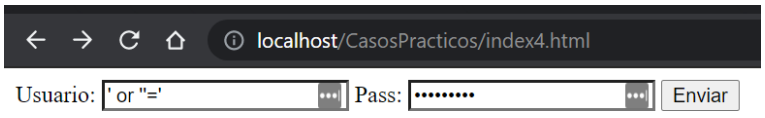


Figura 7: SQL Injection

Con esos dos campos, se podría interpretar que un usuario esta simplemente iniciando sesión, pero en realidad lo que está haciendo es modificar la sentencia SQL que está escrita en el lado del servidor para que este devuelva un resultado no esperado. La sentencia SQL que se estaría ejecutando en el lado del servidor seria la siguiente:

```
SELECT *
FROM usuarios
WHERE user =’’ OR ’’=’’ AND pass = ’’ OR ’’=’’;
```

Esto significa que la sentencia siempre va a devolver un valor, cuando el usuario y la contraseña estén vacíos o cuando se cumpla la siguiente condición, la cual siempre va a ser cierta. Por lo que al usuario que estaría realizando el ataque le devolvería los valores disponibles en la tabla usuarios:

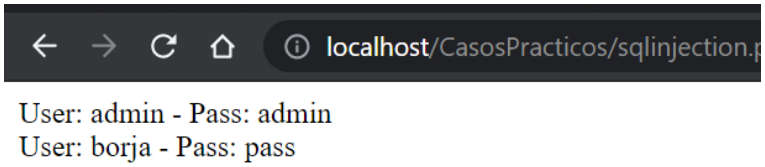


Figura 8: Resultado SQL Injection

- **Cómo solucionar el problema**
  Una de las soluciones para que este tipo de ataques no ocurra, es lo primero tratar los datos que se envían al servidor, es decir, no debemos coger el campo que viene en una petición GET o POST y añadirlo directamente en la sentencia SQL. Para ello, usamos *prepared statements* y *parameterized queries*. Por ejemplo, en una página web PHP se podría de la siguiente manera:

```
$stmt = $dbConnection->prepare('SELECT *
FROM usuarios
WHERE user = ?
AND pass = ?');
$stmt->bind_param('s', $nombre);
$stmt->bind_param('s', $pass);
$stmt->execute();
```

  Además, hay que eliminar caracteres especiales. Para ello, se pueden usar las siguientes funciones:

```
_mysqli_real_escapestring
_mysqli::escapestring
_mysqli::real_escape_string
```

  Estas funciones PHP, recogen el string que va a ser añadido a la sentencia y lo devuelve con los posibles ataques SQL Injection eliminados.

1.8.3. Inclusión de archivos locales

En este caso práctico se quiere mostrar un mal desarrollo de software, mostrando cómo un usuario avanzado puede enviar mediante un formulario HTML a un servidor PHP un *exploit* y éste pueda realizar diferentes acciones. Además, se mostrará cómo se puede solventar este tipo de vulnerabilidad que es muy común en servidores PHP.

En una página web siempre hay formularios donde se nos hace rellenar una serie de campos, como pueden ser los formularios de registro que nos permiten adjuntar ficheros. En ese caso, se nos muestra que debemos insertar un nombre, un apellido y un fichero.

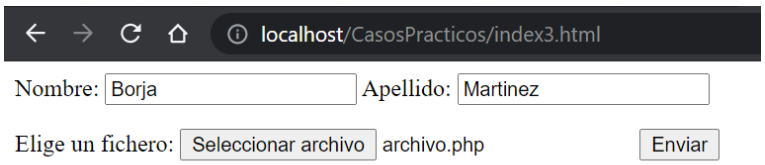


Figura 9: Registro en una pagina web cualquiera

Como podemos observar, estamos adjuntando un fichero PHP llamado *archivo.php*, en un principio parece un fichero inocente que se va a enviar a un servidor y este lo almacenara. Hasta que analizamos el contenido del fichero que contiene la siguiente información:

```
<?php
system($_GET['cmd']);
```

Este pequeño *exploit*, lo que hace es ejecutar el comando *system()* de PHP cuando se le pasa por parámetro GET el comando *cmd* (este comando puede tener cualquier nombre). El comando *system()* lo que hace es ejecutar comandos en el propio servidor, como pueden ser *uname -a* para obtener el sistema operativo del servidor, *ls -al /*, para obtener todos los directorios de la carpeta raíz.

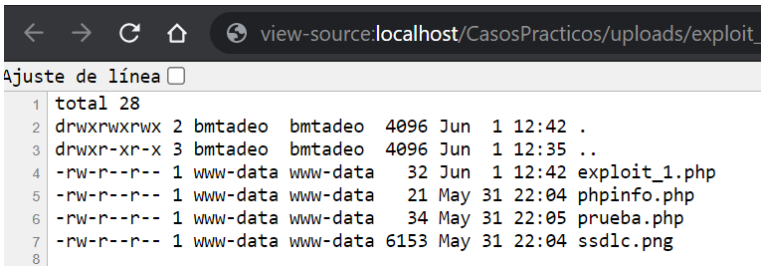


Figura 10: Ejecución del *exploit*

Otro pequeño *exploit*, es adjuntar un fichero PHP con la siguiente información:

```
<?php
phpinfo();
```

El resultado de este fichero será mostrar la mayoría de la información el servidor, desde el sistema operativo a las carpetas de configuración del servidor

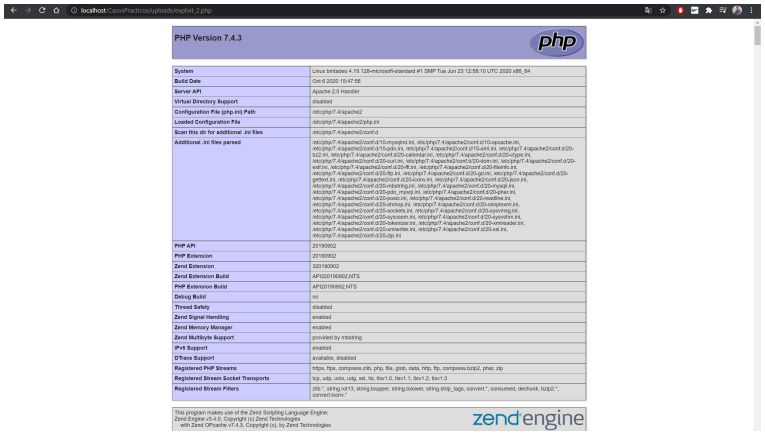


Figura 11: Ejecución del segundo *exploit*

- **Cómo solucionar el problema**
  La solución al problema es muy sencilla, solo que muchos desarrolladores no la aplican en sus desarrollos. Para ello existen diferentes formas para reforzar la seguridad:

  - No permitir que nos envíen cualquier tipo de ficheros, esto se realiza comprobando el *MIME type* del fichero.
  - Comprobar el basename del fichero a subir.
  - Revisar los permisos de la carpeta donde se suben los diferentes archivos.
  - Protección para la no ejecución de PHP en carpetas y evitando el indexado de ficheros, de tal forma que no se muestren a ningún fichero. Esto se realiza creando el fichero *.htaccess* con las siguientes opciones:

```
Options -Indexes
RemoveHandler .php .phtml .php3
RemoveType .php .phtml .php3
php_flag engine off
```

2. Metodología

En este apartado, se va a explicar la metodología seguía en el desarrollo del trabajo. Primero se realizará un diagrama de planificación de tareas, posteriormente se describirán las tareas nombradas en él (descripción, tareas precedentes y salidas) y la duración de cada una de ellas. Tras describir las diferentes tareas existentes durante el proyecto, se trasladarán al diagrama de Gantt, donde se mostrara la duración de las tareas de una forma más gráfica.

2.1. Planificación de las tareas

Respecto a las tareas, se ha desarrollado una Planificación del Trabajo (Figura 12), en la cual se organizan las tareas en distintos campos.

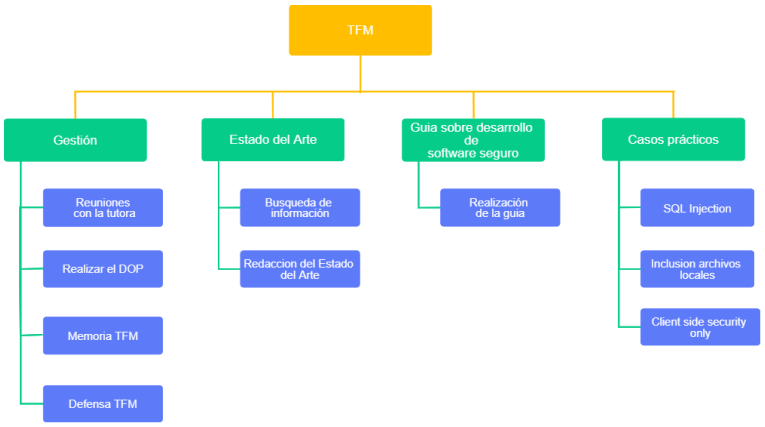


Figura 12: Planificación del Trabajo

- **Gestión:** Consistirá en aclarar todas las dudas posibles acerca del proyecto, además de realizar la documentación necesaria para la memoria del TFM y la defensa de éste.
- **Estado del Arte:** Se explicará la situación actual del desarrollo del software seguro, así como distintos tipos de ataques.
- **Guía sobre desarrollo de software seguro:** Realización de la guía sobre desarrollo de software seguro.
- **Casos prácticos:** Definición de distintos casos prácticos en los que se muestra diferentes ataques en un mal desarrollo de software.

2.2. Paquetes de trabajo

El proyecto se agrupará en distintos paquetes de trabajo, que a su vez estarán constituidos de tareas específicas que permitirán segmentar el proyecto y facilitar el control de éste. La planificación del proyecto es muy importante para para llevar el control del proyecto. A continuación, se detallan los diferentes paquetes de trabajo:

1. **Gestión**

1.1. Tarea: Reuniones con la tutora

Duración	5 horas
Descripción	Reuniones con la tutora del TFM. Sirven para aclarar dudas y finalidades del trabajo.
Tareas precedentes	Ninguna.
Herramientas necesarias	Un ordenador
Entradas	Ninguna
Salidas	Ninguna.

Tabla 5: Tarea: Reuniones con la tutora

1.2. Tarea: Realizar la planificación del proyecto

Duración	24 horas.
Descripción	Realizar la planificación. Documento en el que se encuentran recogidas las intenciones y objetivos del proyecto.
Tareas precedentes	Ninguna
Herramientas necesarias	Un ordenador y Overleaf
Entradas	Ninguna
Salidas	Planificación del proyecto.

Tabla 6: Tarea: Realizar la planificación del proyecto

1.3. Tarea: Memoria TFM

Duración	30 horas.
Descripción	Realización de la memoria del Trabajo de Fin de Máster.
Tareas precedentes	Ninguna
Herramientas necesarias	Un ordenador y Overleaf
Entradas	Ninguna
Salidas	Memoria del TFM.

Tabla 7: Tarea: Memoria del TFM

1.4. Tarea: Defensa del TFM

Duración	30 horas.
Descripción	Realización de la defensa del Trabajo de Fin de Máster.
Tareas precedentes	Memoria del TFM
Herramientas necesarias	Un ordenador
Entradas	Ninguna
Salidas	Defensa del TFM.

Tabla 8: Tarea: Defensa del TFM

2. **Estado del arte**

2.1. Tarea: Búsqueda de información

Duración	30 horas.
Descripción	Búsqueda de la información sobre el estado del arte
Tareas precedentes	Ninguna
Herramientas necesarias	Un ordenador
Entradas	Ninguna
Salidas	Información.

Tabla 9: Tarea: Búsqueda de información

2.2. Tarea: Realizar el Estado del Arte

Duración	30 horas.
Descripción	Realización del Estado del Arte. Documento en el que se encuentra el estado actual del desarrollo de software seguro.
Tareas precedentes	Búsqueda de Información
Herramientas necesarias	Un ordenador y Overleaf
Entradas	Información
Salidas	El Documento de Estado del Arte.

Tabla 10: Tarea: Realizar el Estado del Arte

3. **Guía sobre desarrollo de software seguro**

3.1. Tarea: Realización de la guía

Duración	25 horas.
Descripción	Realización de la guía sobre desarrollo de software seguro
Tareas precedentes	Realizar el Estado del Arte
Herramientas necesarias	Un ordenador y Overleaf
Entradas	El Documento de Estado del Arte.
Salidas	Guía de desarrollo de software.

Tabla 11: Tarea: Realización de la guía

4. **Casos prácticos**

4.1. Tarea: SQL Injection

Duración	25 horas.
Descripción	Realización del caso práctico sobre SQL Injection
Tareas precedentes	Ninguna
Herramientas necesarias	Un ordenador
Entradas	Ninguna
Salidas	Primer caso práctico.

Tabla 12: Tarea: SQL Injection

4.2. Tarea: Inclusión de archivos locales

Duración	25 horas.
Descripción	Realización del caso práctico de inclusión de archivos locales.
Tareas precedentes	Ninguna
Herramientas necesarias	Un ordenador
Entradas	Ninguna
Salidas	Segundo caso práctico.

Tabla 13: Tarea: Inclusión de archivos locales

Duración	25 horas.
Descripción	Realización del caso práctico de Client-side security only.
Tareas precedentes	Ninguna
Herramientas necesarias	Un ordenador
Entradas	Ninguna
Salidas	Tercer caso práctico.

Tabla 14: Tarea: Client-side security only

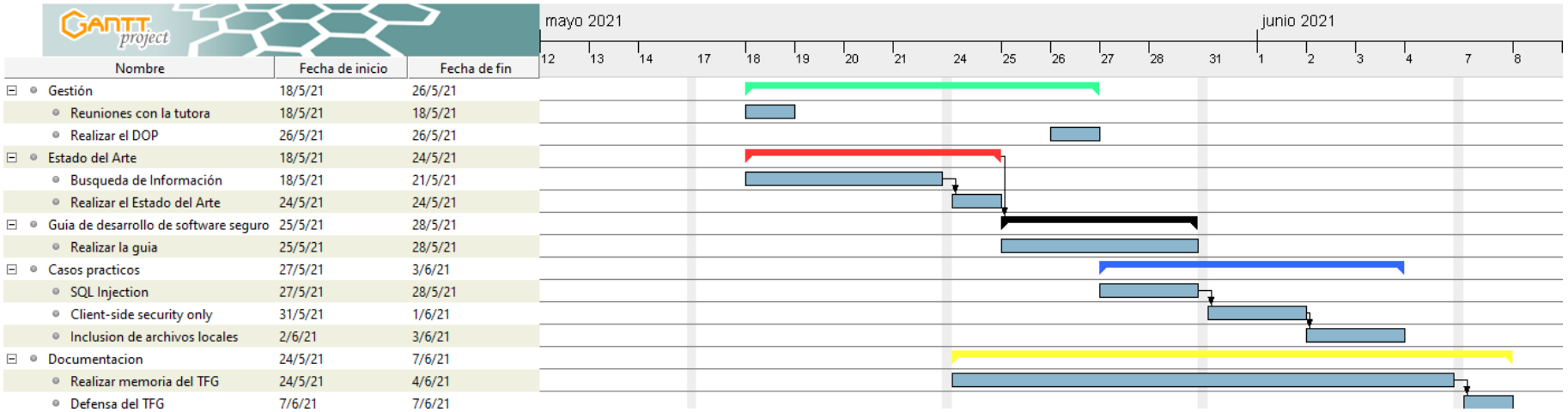
2.3. Planificación temporal

Tras decidir y explicar las tareas que se van a realizar durante el proyecto, antes de organizar los tiempos, indicamos los tiempos en la siguiente tabla para cada tarea.

Tarea	**Estimación**
1. Gestión	**95 horas**
1.1. Reuniones con la tutora	5 horas
1.2. Realizar la planificación	5 horas
1.3. Realizar memoria del TFM	80 horas
1.4. Defensa del TFM	5 horas
2. Estado del Arte	**60 horas**
2.1. Búsqueda de información	30 horas
2.2. Realizar el Estado del Arte	30 horas
3. Guía de desarrollo de software seguro	**25 horas**
3.1. Realización de la guía	25 horas
4. Casos prácticos	**45 horas**
4.1. SQL Injection	15 horas
4.2. Inclusión de archivos locales	15 horas
4.3. Client-side security only	15 horas
Horas totales	**225 horas**

Tabla 15: Planificación temporal de las tareas

2.3.1. Diagrama de Gantt



Nombre	Fecha de inicio	Fecha de fin
Gestión	18/5/21	26/5/21
Reuniones con la tutora	18/5/21	18/5/21
Realizar el DOP	26/5/21	26/5/21
Estado del Arte	18/5/21	24/5/21
Busqueda de Información	18/5/21	21/5/21
Realizar el Estado del Arte	24/5/21	24/5/21
Guia de desarrollo de software seguro	25/5/21	28/5/21
Realizar la guia	25/5/21	28/5/21
Casos practicos	27/5/21	3/6/21
SQL Injection	27/5/21	28/5/21
Client-side security only	31/5/21	1/6/21
Inclusion de archivos locales	2/6/21	3/6/21
Documentacion	24/5/21	7/6/21
Realizar memoria del TFG	24/5/21	4/6/21
Defensa del TFG	7/6/21	7/6/21

Figura 13: Diagrama de Gantt

3. Aspectos económicos

A la hora de realizar un proyecto, es importante ver el impacto económico que este supondrá y estimar lo que va a costar su desarrollo, así como la amortización del mismo. Para calcular el coste total del desarrolla del proyecto, se calculara en diferentes apartados. Primero se calculara los gastos como el software y el hardware. Por último, el gasto de la mano de obra en función de las horas totales invertidas en el proyecto.

3.1. Hardware

Respecto al hardware, para desarrollar el proyecto se ha utilizado un MacBook Pro Retina de 13" del año 2014, cuyo precio inicial fue de 1.256,00€. En la siguiente tabla, se muestra el precio y la vida útil, para así, después poder calcular la amortización mensual del equipo respecto a su vida útil y posteriormente calcular la amortización durante el proyecto.

Equipo	**Vida útil estimada**	**Precio del equipo**
MacBook Pro Retina de 13"	96 meses	1.256,00€

Tabla 16: Precio y vida útil del hardware

Tras haber realizado los diferentes cálculos, la amortización mensual y la amortización durante el proyecto es la siguiente:

- **Amortización mensual:** 13,08€/mes
- **Amortización durante el proyecto:** 117,72€

3.2. Software

Por otra parte, la mayor parte del software utilizado en la realización de este proyecto o es de código abierto o de uso gratuito. El sistema operativo del ordenador es macOS, por lo que el precio de su licencia ya viene incluida en el precio del equipo. El editor de LaTeX con el que se esta escribiendo la memoria es Overleaf, de uso gratuito.

3.3. Recursos humanos

En cuanto a personal, teniendo en cuenta en base al BOE-A-2019-14977 del 18 de octubre del 2019 sobre el convenio colectivo del sector de Empresas de Ingeniería y Oficinas de Estudios Técnicos [14], el salario mínimo anual de dos analistas es de 23.973,88€. Un analista es el encargado de realizar las 225 horas del proyecto y el otro analista es el encargado de la supervisión de este durante 10 horas del proyecto.

El salario se divide en 14 pagas, donde cada paga es de 1.712,42€/mes. Sabiendo que una jornada laboral en una semana son 40 horas y mensualmente son 160 horas, el precio por hora de un trabajador es de 10,70€/hora.

Para el calculo del coste de los recursos humanos del primer analista, multiplicamos el coste a la hora por las 225 horas del proyecto, obteniendo que los recursos humanos del primer analista son 2.407,50€. Para el segundo analista, multiplicamos el coste a la hora por las 10 horas dedicadas al proyecto, obteniendo que los recursos humanos dedicados al proyecto son 107€/es.

3.4. Gastos totales

Concepto	**Coste**
Hardware	117,72€
Software	0€
Mano de obra	2.514,50€
Total	**2.632,22€**

Tabla 17: Gastos totales del proyecto

4. Conclusiones y trabajo futuro

En este último apartado se tratarán las diferentes conclusiones obtenidas tras realizar el proyecto, se mostrará la verificación de los objetivos que se establecieron al comienzo del mismo y finalmente se hablará de las distintas mejoras que podría tener este proyecto en el futuro.

4.1. ¿Se han cumplido los objetivos establecidos?

A la hora de planificar el proyecto se establecieron distintos objetivos que se pretendían obtener tras la realización del proyecto, a continuación, se analizara si los siguientes objetivos se han cumplido:

- **Elaboración de una guía en base a las metodologías de seguridad existentes**
  Se ha elaborado una guía que puede permitir a un desarrollador de software tener una metodología a seguir a la hora de desarrollar un producto seguro. Esta guía recoge diferentes técnicas a utilizar desde el análisis de requisitos hasta el mantenimiento futuro del software una vez este se encuentra en producción.

- **Elaboración de diferentes casos prácticos de fallos de seguridad relacionados con un mal desarrollo de software**
  Se han elaborado diferentes casos prácticos que pueden servir a desarrolladores de software a solventar dichos fallos, como un SQL Injection, inclusión de ficheros locales o validar datos solo en el lado del cliente. Estos casos son los más comunes que se pueden encontrar del lado de la seguridad en un desarrollo de software y con estos casos prácticos se da una solución a esta problemática.

4.2. Trabajo futuro

Este proyecto venía a crear una guía de desarrollo de software seguro, de tal forma que a los desarrolladores les sirviese de guía mientras desarrollan diferentes productos.

Una vez creada la guía y teniendo la base de las diferentes problemáticas que existen junto a las distintas guías que hay en la actualidad, una posible ampliación que se podría realizar sería un analizador de código en base a la seguridad. Este analizador de código, podría realizar comprobaciones de distintas vulnerabilidades que hay en la actualidad, como puede ser revisar que se validan los datos, no existan fallos como inyecciones SQL. De esta forma, este analizador podría servir de ayuda a los programadores.

5. Referencias

(1) Comprendiendo la vulnerabilidad XSS (Cross-site Scripting) en sitios web: https://www.welivesecurity.com/la-es/2015/04/29/vulnerabilidad-xss-cross-site-scripting-sitios-web/

(2) OWASP Cross Site Scripting (XSS): https://owasp.org/www-community/attacks/xss/

(3) OWASP SQL Injection: https://owasp.org/www-community/attacks/SQL_Injection

(4) OWASP Local File Inclusion: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/11.1-Testing_for_Local_File_Inclusion/

(5) 2020, año record en ciberataques https://www.elindependiente.com/espana/2021/01/01/2020-ano-record-en-ciberataques/

(6) Secure Software Development Framework https://csrc.nist.gov/Projects/ssdf

(7) ISO 27001 https://www.isotools.org/normas/riesgos-y-seguridad/iso-27001/

(8) Cuanto cuesta un ciberataque a las empresas https://opendatasecurity.io/cuanto-cuesta-un-ciberataque-a-las-empresas/

(9) OWASP-Secure Software Development Life Cycle https://owasp.org/www-pdf-archive/OWASP-LATAMTour-Patagonia-2016-rvfigueroa.pdf

(10) Introduction to the CLASP Process https://us-cert.cisa.gov/bsi/articles/best-practices/requirements-engineering/introduction-to-the-clasp-process

(11) ISO/IEC 27034 https://noticias.cec.es/index.php/2020/03/20/isoiec-27034-estandar-internacional-para-la-seguridad-de-las-aplicaciones/

(12) SAFECode https://safecode.org/

(13) BOE-A-2019-14977 sobre el Convenio colectivo del sector de empresas de ingeniería y oficinas de estudios técnicos: https://www.boe.es/boe/dias/2019/10/18/pdfs/BOE-A-2019-14977.pdf

(14) OWASP: https://owasp.org

(15) OWASP Testing Guide: https://owasp.org/www-project-web-security-testing-guide/

(16) Comandos Google Hacking:
https://byte-mind.net/tecnicas-google-hacking-para-recolectar-informacion-publica/