Universidad de las Ciencias Informáticas

Facultad 1

“Módulo de PHP para la herramienta Auditoría de Código Fuente”

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor Mayvis Rivet Martos

Tutor Ing. Abel Fírvida Donéstevez
Ing. Michel Evaristo Febles Parker

La Habana, junio de 2015

Declaración de Autoría

Declaro ser la autora del presente Trabajo de Diploma y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo.

Para que así conste firmo la presente declaración jurada de autoría en La Habana a los días ____ del mes de _____ del año _____.

Mayvis Rivet Martos

_____

Firma del autor(a)

Ing. Abel Fírvida Donéstevez

_____

Firma del Tutor(a)

Ing. Michel Evaristo Febles Parker

_____

Firma de Tutor(a)

"La confianza en uno mismo y la rápida decisión son el preludio del éxito."

José Martí

Agradecimientos

Quisiera agradecer a todas las personas que de una forma han estado presente cuando los he necesitado.

A mis tutores por toda la ayuda brindada durante la tesis.

A mis compañeros de grupo por todos los momentos compartidos.

A mis amigas de infancia que han estado siempre a mi lado.

A mis amigas de la universidad por compartir conmigo estos años juntas.

A mi novio y a su familia por acompañarme en estos momentos.

A toda mi familia por apoyarme siempre en todas las decisiones que he tomado; los quiero mucho.

Dedicatoria

A mi mamá y a mi papá que siempre me han apoyado en todos los momentos.

A mi hermano por ser un ejemplo a seguir en mi vida.

A mi novio por estar a mi lado.

Resumen

La herramienta Auditoría de Código Fuente fue desarrollada para garantizar seguridad a la distribución cubana GNU/Linux Nova, manteniéndola fuera de ataques y sin puertas traseras. Realiza auditorías sobre aplicaciones que se encuentran en el repositorio de Nova y que están desarrolladas en los lenguajes de programación C/C++, Perl, Python y Bash; sin embargo, actualmente no posee mecanismos de detección de vulnerabilidades en códigos fuente de PHP. El presente Trabajo de Diploma tiene como propósito la realización de un módulo para la herramienta ACF, que sea capaz de detectar las funciones de riesgo que contengan los ficheros de PHP del repositorio de Nova. Para dar cumplimiento a este objetivo se lleva a cabo un análisis de las principales vulnerabilidades de este lenguaje de programación y las herramientas que se dedican a su detección. En la implementación de la propuesta de solución se emplea el Entorno Integrado de Desarrollo Eclipse, el lenguaje de programación C++ y como metodología para guiar el proceso de *software* OpenUP. Además de llevar un control de versiones mediante la herramienta Git, la documentación del código con Doxygen y el modelado con la herramienta Visual Paradigm. Se realizan pruebas de funcionalidad y pruebas de aceptación para verificar la calidad del producto. Finalmente se presenta como resultado fundamental de la investigación, un módulo que permite a la herramienta ACF detectar posibles funciones vulnerables utilizadas en PHP, mediante un análisis estático realizado al código fuente.

Palabras claves: auditoría, código fuente, PHP, repositorio, vulnerabilidades.

Índice

Índice de figuras

Índice de tablas

Introducción

En la actualidad, la informática juega un papel fundamental en todos los ámbitos de la vida, pues vivimos en una sociedad comandada por las nuevas tecnologías. Debido a su rápido crecimiento y expansión, la población es cada vez más dependiente de los beneficios que brindan, destacando el uso de la web.

Para el desarrollo de las aplicaciones que funcionan en la web han surgido los lenguajes de programación, los cuales se originan debido a las tendencias y necesidades de las plataformas; dentro de los más conocidos se pueden encontrar: HTML[1], Javascript, ASP[2], ASP.NET, JSP[3], Ruby, Python, y PHP[4]. [1]

Este último, es un lenguaje de programación originalmente diseñado para el desarrollo de sitios web con contenido dinámico. Fue uno de los primeros lenguajes de programación del lado del servidor que se podía incorporar directamente en el documento HTML, en lugar de llamar a un archivo externo que procese los datos. El código es interpretado por un servidor web con un módulo de intérprete de PHP que genera la página web resultante. [2]

PHP es popular para los desarrolladores web porque se ejecuta en más de 200 millones de sitios, incluyendo *Digg*, *Yahoo*, *Wikipedia* y *Facebook;* lo cual demuestra que es un lenguaje potente que puede servir para pequeñas páginas o para grandes portales. Proyectos como *Wordpress*, *Joomla* y *Drupal* han basado su desarrollo en PHP. [3]

Este lenguaje es utilizado en muchos de los centros de desarrollo de *software* de la Universidad de las Ciencias Informáticas (UCI), tal es el caso de los centros Ideoinformática (CIDI), Gobierno Electrónico (CEGEL), Informatización de Entidades (CEIGE), Telemática, Geoinformática y Señales Digitales (GEYSED), y la Empresa de Tecnologías de la Información para la Defensa (XETID).

EL Centro de Software Libre (CESOL) de la UCI, cuenta con el Departamento de Sistema Operativo, el cual tiene como producto principal la Distribución Cubana de GNU/Linux Nova. Dicho sistema operativo

1 **HTML**(acrónimo en inglés de *HyperText Markup Language*, en español Lenguaje de Marcas Hipertextuales)

2 **ASP**(acrónimo en inglés de *Active Server Pages*)

3 **JSP**(acrónimo en inglés de *Java Server Pages*)

4 **PHP(**acrónimo en inglés de *Hypertext Pre-processor*)

fue creado basándose en los principios de seguridad, soberanía tecnológica, socio-adaptabilidad y sostenibilidad, con el objetivo de alcanzar los niveles de excelencia [4]. Para garantizar el principio de seguridad, se ha desarrollado la herramienta Auditoría de Código Fuente (ACF); la cual tiene como finalidad obtener un sistema seguro de ataques y sin puertas traseras.

Esta herramienta es utilizada para encontrar fallas de seguridad en el código fuente de cualquier aplicación o librería que se encuentre en el repositorio[5] de Nova. Hasta ahora el proyecto ACF no soporta la revisión de código PHP, lo que podría conllevar a la inclusión de vulnerabilidades a las aplicaciones desarrolladas con php que estén presentes en el repositorio. La explotación de dichas vulnerabilidades tendrían consecuencias, tales como: permitir realizar actos sin permiso del administrador del equipo, evitar los mecanismos de protección, elevación de privilegios de usuarios, denegación de servicio, caída de la aplicación o cualquier otra variante para violar la seguridad del sistema.

Teniendo en cuenta lo anteriormente planteado surge el siguiente **problema de investigación**: ¿Cómo detectar vulnerabilidades en códigos fuente PHP del repositorio de Nova utilizando la herramienta ACF? Se establece como **objeto de estudio** el proceso de detección de vulnerabilidades en códigos fuente PHP.

Para dar solución a este problema se propone como **objetivo general** desarrollar un módulo que permita a la herramienta ACF la revisión de códigos fuente PHP en el repositorio de Nova. Considerando como **objetivos específicos** los siguientes**:**

- Realizar un estudio sobre las herramientas de auditoría de código fuente de PHP existentes en la actualidad.
- Diseñar un módulo que permita la detección de vulnerabilidades en el código fuente de PHP.
- Implementar el módulo de PHP para la herramienta ACF.
- Realizar pruebas al sistema para comprobar sus funcionalidades.

El **campo de acción** se centra en la detección de vulnerabilidades de las aplicaciones desarrolladas en

5 **Repositorio:** Sitio centralizado donde se almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos.

PHP del repositorio de Nova. De acuerdo a la investigación se sostiene como **idea a defender** que el desarrollo de un módulo para la revisión de códigos fuente PHP permitirá a la herramienta ACF aumentar la detección de posibles vulnerabilidades en el repositorio Nova.

En relación con los objetivos específicos se presentan las **tareas de investigación** correspondientes**:**

- Revisión de bibliografías asociadas a las herramientas de auditoría de código PHP para comprender sus funcionamientos.
- Definición de los requisitos funcionales y no funcionales para obtener las necesidades del módulo a desarrollar.
- Análisis de la arquitectura de ACF para la vinculación del módulo.
- Diseño del módulo acoplado a la arquitectura de ACF.
- Codificación de las funcionalidades definidas.
- Diseño y ejecución de casos de prueba a las funcionalidades implementadas para la evaluación de la solución.

El **resultado esperado** es que la herramienta ACF sea capaz de detectar vulnerabilidades en códigos PHP.

Métodos científicos

Teóricos:

El método **analítico-sintético** es utilizado para investigar y poder extraer de las bibliografías existentes, los elementos esenciales que debe tener el módulo a desarrollar. Además de permitir realizar un análisis de las herramientas de auditoría de código fuente de PHP y sintetizar las relaciones existentes entre las mismas, de acuerdo a los tipos de vulnerabilidades que detectan.

El método **histórico-lógico** es empleado con el objetivo de reconocer la trayectoria real de las aplicaciones encargadas de detectar las vulnerabilidades del código fuente de PHP y profundizar sobre la evolución, funcionamiento y desarrollo de éstas.

El método **inductivo-deductivo** es aplicado en el estudio del comportamiento de aplicaciones encargadas de la auditoría de código fuente de PHP, para poder obtener las funcionalidades comunes en los sistemas

y el modo en que se efectúan los chequeos.

El método de **modelación** es utilizado para representar el comportamiento, organización y los elementos pertenecientes al sistema. Se realizan diagramas necesarios para obtener una mayor comprensión en el momento del desarrollo del módulo.

Empíricos:

El método de **observación** se utiliza como técnica para la captura de los requisitos funcionales y no funcionales, seleccionando los aspectos necesarios que debe cumplir el sistema a desarrollar.

El documento consta de tres capítulos, los cuales abarcan todo el proceso de investigación y desarrollo del sistema propuesto.

- **Capítulo 1. Fundamentación teórica de las herramientas de auditoría de códigos fuente PHP:** Se enfoca en los conceptos asociados a la investigación, las herramientas relacionadas con la auditoría de código fuente de PHP y la relación entre las mismas; además de exponer las tecnologías a utilizar en el desarrollo del módulo.
- **Capítulo 2. Análisis y diseño del Módulo de PHP:** Aborda los detalles principales relacionados con el análisis y diseño de la solución propuesta. Se obtienen los requisitos tanto funcionales como no funcionales que debe poseer el sistema y los elementos que corresponden a cada una de las fases de la metodología de desarrollo de *software* empleada.
- **Capítulo 3. Implementación y pruebas del Módulo de PHP:** Se hace referencia a los aspectos relacionados con la implementación de la solución y a las pruebas realizadas para verificar la calidad requerida del producto de *software*.

Capítulo 1: Fundamentación teórica de las herramientas de auditoría de códigos fuente PHP.

Introducción

Este capítulo muestra una descripción de los principales conceptos relacionados con la investigación, para poder comprender los temas fundamentales. Se presentan las vulnerabilidades que afectan al código fuente de PHP y en los períodos de tiempo que han sido registradas las funciones[6] en riesgo. Además de realizar un estudio de las herramientas dedicadas a este propósito y las tecnologías a utilizar en el desarrollo del módulo. Se exponen las posibles funciones vulnerables, junto con la descripción y clasificación de cada una de éstas.

1.1 Definiciones de interés

1.1.1 Auditoría de Código Fuente

La auditoría del código fuente se realiza para conocer el nivel de seguridad de las aplicaciones utilizadas en los sistemas. Mediante un examen sistemático, se tiene como finalidad descubrir los errores que puedan conducir a vulnerabilidades que debilitan la integridad de su información. Proporcionando soluciones para poder mejorar la seguridad informática. [5]

1.1.2 Seguridad Informática

Seguridad informática es la disciplina que se ocupa de diseñar las normas, procedimientos, métodos y técnicas, orientados a proveer condiciones seguras y confiables, para el procesamiento de datos en sistemas informáticos [6]. La seguridad se fundamenta en tres principios para lograr sus objetivos, los cuales deben ser cumplidos por todo sistema informático: Confidencialidad, Integridad y Disponibilidad.

Confidencialidad: Se refiere a la privacidad de los elementos de información almacenados y procesados

6 **Funciones:** Conjunto de sentencias e instrucciones que se encuentran definidas dentro de una estructura, con el objetivo de que se realice una acción específica

en un sistema informático. Basándose en este principio, las herramientas de seguridad informática deben proteger el sistema de invasiones y accesos por parte de personas o programas no autorizados [7]. Los objetos de un sistema han de ser accedidos únicamente por elementos autorizados a ello, y que esos elementos autorizados no van a utilizar la información disponible para otras entidades.

Integridad: Se relaciona con la validez y consistencia de los elementos de información almacenados y procesador en un sistema informático. Se debe asegurar que los procesos de actualización estén bien sincronizados y no se dupliquen, de forma que todos los elementos del sistema manipulen adecuadamente los mismos datos [7]. Los objetos sólo pueden ser modificados por elementos autorizados, y de una manera controlada.

Disponibilidad: Está enfocado a la continuidad de acceso a los elementos de información almacenados y procesados en un sistema informático. Este principio indica que los objetos del sistema tienen que permanecer accesibles a elementos autorizados. Es importante en sistemas informáticos cuyo compromiso con el usuario, es prestar servicio permanente. [7]

Al profundizar sobre este aspecto, el Proyecto Abierto de Seguridad en Aplicaciones Web (OWASP por sus siglas en inglés) considera que los dentro de los riesgos críticos en la seguridad de las aplicaciones web se encuentran: Inyección, Secuencia de Comandos en Sitios Cruzados (XSS), Pérdida de Autenticación y Gestión de Sesiones, Referencia Directa Insegura a Objetos, Falsificación de Peticiones en Sitios Cruzados (CSRF), Defectuosa Configuración de Seguridad, Almacenamiento Criptográfico Inseguro, Falla de Restricción de Acceso a URL, Protección Insuficiente en la Capa de Transporte, Redirecciones y reenvíos no validados.

- **Inyección**

Las fallas de inyección, tales como SQL[7], sistema operativo, y LDAP[8], ocurren cuando datos no confiables

7 El lenguaje de consulta estructurado o **SQL** (por sus siglas en inglés *Structured Query Language*) es un lenguaje declarativo de acceso a bases de datos.

8 **LDAP** (Protocolo compacto de acceso a directorios) es un protocolo estándar que permite acceder a bases de información de usuarios de una red.

son enviados a un intérprete como parte de un comando o consulta. Los datos hostiles del atacante pueden engañar al intérprete en ejecutar comandos no intencionados o acceder a datos no autorizados. Una falla de inyección puede resultar en perdida o corrupción de datos, falta de integridad, o negación de acceso y en ocasiones puede llevar a la toma de posesión completa del servidor. [9]

- **XSS**

El XSS es un fallo de seguridad en sistemas de información basados en web, que más que comprometer la seguridad del servidor web compromete la seguridad del cliente. Ocurren cuando una aplicación web permite inyectar código en la página; esto se puede lograr por medio de campos de texto de formularios o por medio de la URL. XSS permite a los atacantes ejecutar secuencias 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. [9]

- **Pérdida de Autenticación y Gestión de Sesiones**

Las funciones de la aplicación relacionadas con la autenticación y gestión de sesiones son frecuentemente implementadas incorrectamente, permitiendo a los atacantes comprometer contraseñas, llaves, token[9] de sesiones, o explotar otras fallas de implementación para asumir la identidad de otros usuarios. [9]

- **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. [9]

- **CSRF**

El CSRF es prácticamente igual que XSS, salvo que este ataque se basa en explotar la confianza que un usuario tiene en un sitio web. Un ataque CSRF obliga al navegador de una víctima autenticada a enviar

9 Un **token de seguridad** es un dispositivo electrónico que se le da a un usuario autorizado de un servicio computarizado para facilitar el proceso de autenticación.

una petición HTTP[10] falsificado, 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 pedidos que la aplicación vulnerable piensa son peticiones legítimas provenientes de la víctima. De ser exitoso, el CSRF puede comprometer la información del usuario y el comportamiento normal del usuario en la aplicación web. En caso de que el usuario sea el administrador de la aplicación web, este tipo de vulnerabilidad puede llegar a comprometer por completo al sistema en cuestión. [9]

- **Defectuosa Configuración de Seguridad**

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. [9]

- **Almacenamiento Criptográfico Inseguro**

Muchas aplicaciones web no protegen adecuadamente los datos sensibles, tales como tarjetas de crédito, NSSs[11] y credenciales de autenticación con mecanismos de cifrado. Atacantes pueden modificar o robar tales datos protegidos inadecuadamente para conducir robos de identidad, fraudes de tarjeta de crédito u otros crímenes. [9]

- **Falla de Restricción de Acceso a URL**

Las aplicaciones web verifican los privilegios de acceso a URLs[12] antes de generar enlaces o botones protegidos. Sin embargo, las aplicaciones necesitan realizar controles similares cada vez que estas

10 **HTTP de** *HyperText Transfer Protocol* (Protocolo de transferencia de hipertexto) es el método más común de intercambio de información en la world wide web, el método mediante el cual se transfieren las páginas web a un ordenador.

11 **NSS** es el acrónimo del inglés *Network Switching Subsystem* o Subsistema de Conmutación de Red. Es el componente que realiza las funciones de portar y administrar las comunicaciones entre teléfonos móviles y la Red Conmutada de Telefonía Pública (PSTN) para una red GSM.

12 **URL** (siglas en inglés de *Uniform Resource Locator*) es un identificador de recursos uniforme (URI) cuyos recursos referidos pueden cambiar, esto es, la dirección puede apuntar a recursos variables en el tiempo.

páginas son accedidas, o los atacantes podrán falsificar URLs para acceder a estas páginas igualmente. [9]

- **Protección Insuficiente en la Capa de Transporte**

Las aplicaciones frecuentemente fallan al autenticar, cifrar, y proteger la confidencialidad e integridad de tráfico de red sensible. Cuando esto ocurre, es debido a la utilización de algoritmos débiles, certificados expirados, inválidos, o sencillamente no utilizados correctamente. [9]

- **Redirecciones y reenvíos no validados**

Las aplicaciones web 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*[13] o *malware*[14], o utilizar reenvíos para acceder páginas no autorizadas. [9]

La identificación de algunos de los riesgos más críticos que enfrentan la seguridad en aplicaciones web, permiten reconocer los daños que podrían ocasionar al sistema al que comprometan. El escenario de amenazas presentan constantemente cambios, los factores clave en esta evolución son provocados debido a los avances hechos por los atacantes, liberación de tecnologías con nuevas debilidades y defensas incorporadas, así como el despliegue de sistemas cada vez más complejos.

1.2 Análisis de vulnerabilidades del lenguaje PHP

PHP no está exento de problemas de seguridad. En informática se llama *exploit* a toda pieza de *software* o secuencia de comandos cuyo objetivo sea provocar un comportamiento no deseado o imprevisto en los programas informáticos, *hardware*, o componente electrónico. En el caso de las páginas *web*, estos *exploits* están destinados a causar comportamientos extraños en la aplicación o extraer información sensible del propio servidor o de los usuarios que visiten la página web. [8]

La seguridad en PHP suele terminar siendo un factor a veces polémico. Cada desarrollador usualmente

13 **Phishing** o suplantación de identidad es un término informático que denomina un modelo de abuso informático.

14 **Malware** es un tipo de software que tiene como objetivo infiltrarse o dañar una computadora o sistema de información sin el consentimiento de su propietario.

tiene sus propios criterios preventivos sujetos a puntos de vista muy específicos, dependiendo de la experiencia personal. Además del carácter cambiante y evolutivo de las brechas y amenazas de seguridad.

A continuación se presentan una gráfica donde se puede observar las vulnerabilidades de PHP encontradas por años, según el sitio oficial de CVE (*Common Vulnerabilities and Exposures*).

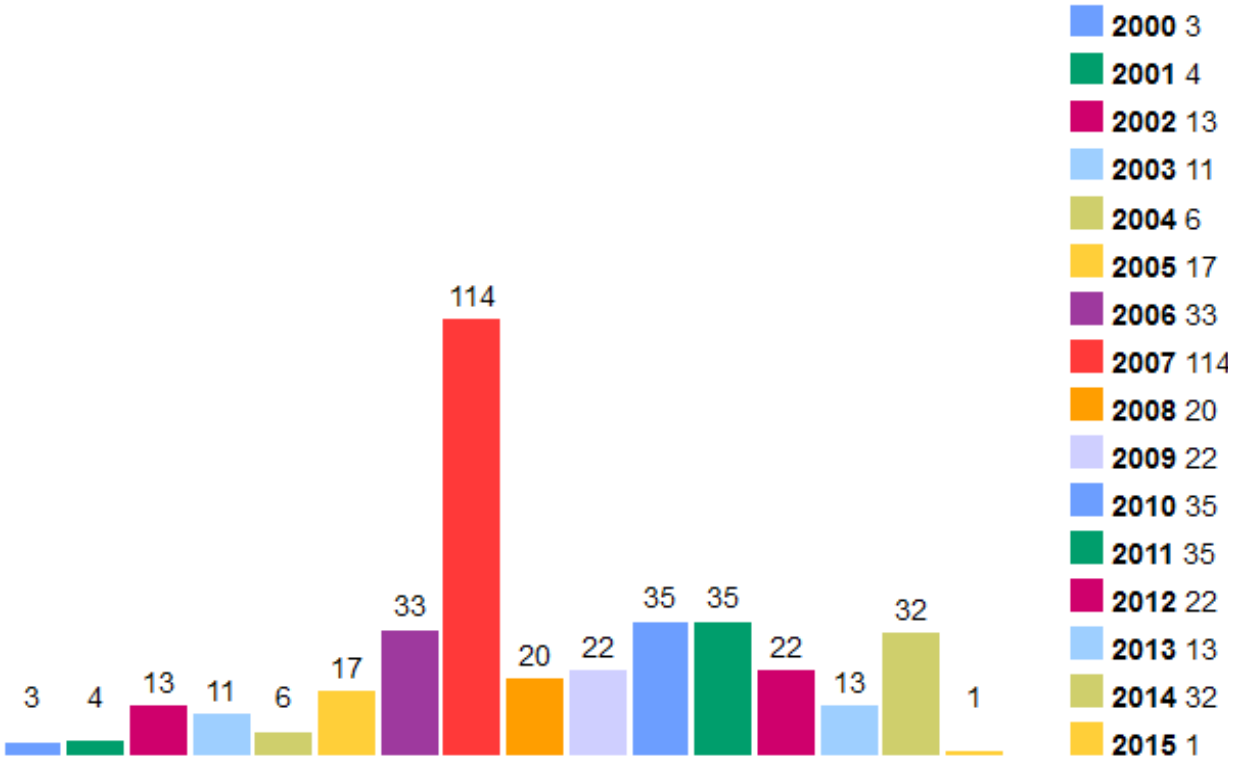

Figura 1: Cantidad de vulnerabilidades por años. [13]

Mediante la representación anterior se demuestra la constante detección de funciones vulnerables, con un total de 381 registradas desde el 2000 hasta principios del 2015 y resaltando el año 2007 con 114.

En la figura 2, se muestran los tipos de vulnerabilidades encontradas con las cantidades de funciones vulnerables que han sido registradas, aportando a la investigación los criterios de búsquedas para localizar riesgos en códigos fuente PHP.

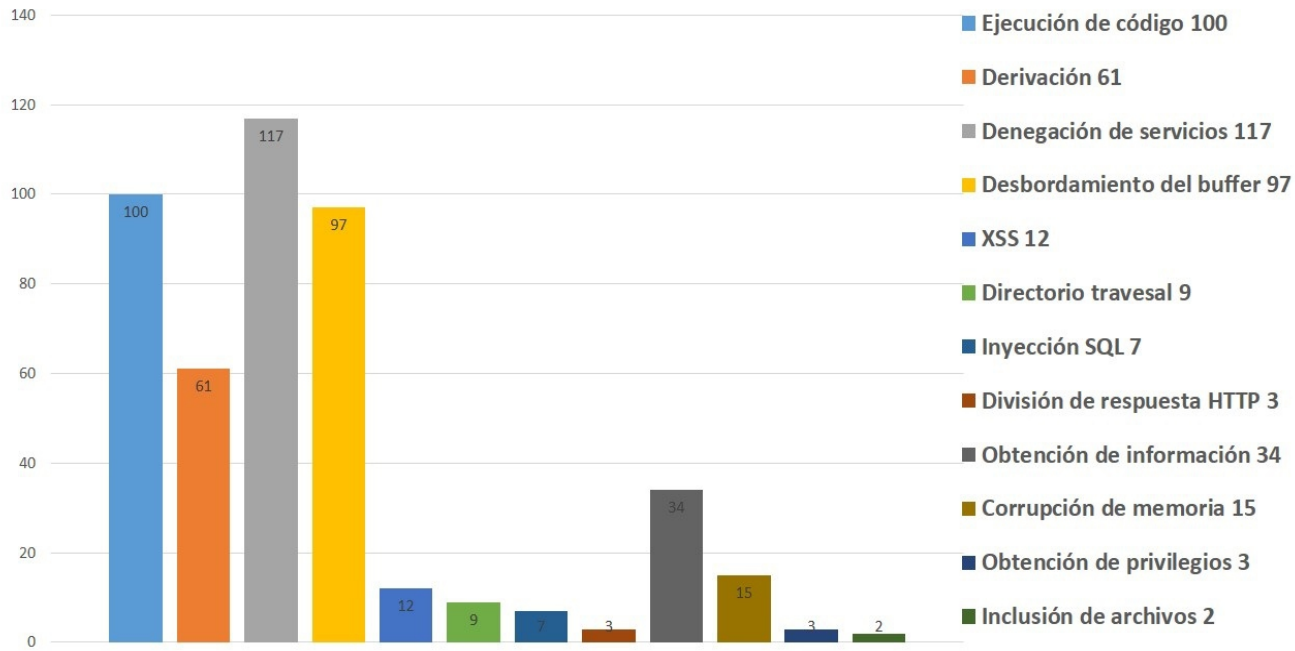

Figura 2: Cantidad de vulnerabilidades por tipos. [13]

Se puede apreciar que dentro de las vulnerabilidades con más funciones registradas se encuentran: denegación de servicio (*DoS*), ejecución de código y desbordamiento de buffer (*overflow*); aunque es recomendable realizar búsquedas de funciones para todo tipo de vulnerabilidad, debido a que todas son perjudiciales para el sistema al que expongan.

1.3 Análisis de las principales herramientas existentes

A continuación se analizan algunas de las herramientas que se encargan de encontrar las vulnerabilidades en el código fuente de PHP, para determinar si son adaptables a la presente investigación. Se realiza una comparación donde se exponen las descripciones de cada una de éstas, junto a los ataques que detectan y tipo de aplicación que representa.

Herramientas	Descripción	Vulnerabilidades que escanea	Tipo
PHP Vulnerability Hunter	Es un *fuzzer*[13] que puede provocar una amplia gama de fallos explotables en las aplicaciones web PHP. Utiliza una combinación de análisis estático y dinámico para asignar automáticamente la aplicación de destino. Debido a que funciona mediante la instrumentación de aplicación, puede detectar entradas que no se hace referencia en los formularios de la página representada. [10]	- Ejecución arbitraria de comandos. - Modificación arbitraria de ficheros (escritura/ cambio/ renombrado/ borrado). - Inclusión de archivos locales (LFI) y lectura arbitraria de ficheros. - Ejecución de PHP. - Inyección SQL. - XSS. - Redirección abierta - Descubrimiento de ruta [10]	Aplicación web
Web Application Protection (WAP)	Con esta herramienta es posible detectar y corregir automáticamente vulnerabilidades de validación de entrada en aplicaciones web escritas en lenguaje PHP (versión 4.0 o superior). Realiza el análisis del flujo de datos para detectar las vulnerabilidades de validación de entrada. El objetivo del análisis es rastrear registros maliciosos insertados por los	- Inyección SQL en MySQL, PostgreSQL y DB2 DBMS. - XSS. - La inclusión de archivos remotos (RFI). - Inclusión de archivos locales (LFI). - Salto de directorio o camino transversal (DT / PT). - Divulgación de Código Fuente (SCD). - Inyección de comandos del	- Herramienta de código abierto, distribuída bajo la licencia GPLv3. - Multiplataforma, puesto que existen versiones para Windows, Linux y Mac OSX (para su ejecución sólo se necesita Java). [12]

13 Un **fuzzer** es un programa que intenta descubrir vulnerabilidades de seguridad enviando una entrada arbitraria a una aplicación.

Herramientas	Descripción	Vulnerabilidades que escanea	Tipo
	puntos de entrada ($ _GET y $ _POST) y verificar funciones de PHP que pueden ser explotadas por alguna entrada maliciosa. WAP está escrito en lenguaje Java y está constituido por tres módulos: Analizador de Código, Detección de falsos positivos y Código corrector. [11]	sistema operativo (OSCI). - Inyección de código PHP. [11]	
RIPS	Es un escáner de seguridad para encontrar vulnerabilidades en aplicaciones PHP a través del análisis estático de código. RIPS es capaz de transformar el código PHP origen en un modelo de programa y detectar los *sinks* sensibles (funciones potencialmente vulnerables) que pueden ser contaminadas por la entrada del usuario durante el flujo de la aplicación (influenciado por un usuario malicioso). RIPS también ofrece un marco de código de auditoría integrada para el análisis manual posterior. [13]	- XSS. - Inyección SQL. - Descubrimiento de ficheros. - Inclusión de archivos remotos (RFI). - Inclusión de archivos locales (LFI). - Ejecución Remota de Código (RCE). [13]	Aplicación web

Herramientas	Descripción	Vulnerabilidades que escanea	Tipo
Damn Vulnerable Web App (DVWA)	Es un reconocido entorno de entrenamiento en explotación de seguridad web escrito en PHP y MySQL cuyo objetivo principal es permitir a programadores y técnicos estudiar e investigar sobre las diferentes temáticas involucradas en dicho campo en un entorno completamente legal. Es posible realizar pruebas sobre los diferentes tipos de ataques web que se pueden llevar a cabo en este tipo de aplicación, y más concretamente sobre páginas web PHP. [8]	- Fuerza Bruta. -Ejecución de comandos. - CSRF. - Inclusión de archivos (locales y remotos). - Inyección SQL. - Subir Archivo. - XSS. [8]	Aplicación web

Tabla 1: Herramientas para detectar vulnerabilidades en el código fuente de PHP.

1.3.1 Resultados de la comparación de las herramientas presentadas

Como se podrá observar en el siguiente esquema, las herramientas estudiadas realizan análisis comunes para el escaneo de vulnerabilidades, tales como: *cross-site scripting* (XSS), inyección SQL, inclusión de archivos remotos (RFI) y la inclusión de archivos locales (LFI). Según estudios previos estos ataques son los que han registrado menos funciones vulnerables a lo largo del tiempo.

No realizan escaneo para detectar funciones que permitan un desbordamiento de buffer, lo cual podría provocar un comportamiento errático del sistema y permitiría la ejecución de código malicioso. Tampoco exploran ataques de tipo de denegación de servicio, obtención de información y corrupción de memoria; por lo que pueden producir perdidas de los datos y causar que los servicios o recursos sean inaccesible a

los usuarios legítimos.

Teniendo en cuenta dichos inconvenientes, se puede llegar a la conclusión de que las herramientas no cumplen en su totalidad con las exigencias de la investigación; sin embargo, para el desarrollo del módulo se propone utilizar los criterios de obtención de vulnerabilidades que realizan e incluirle las investigadas.

Figura 3: Comparación entre las herramientas y el módulo de PHP.

1.4 Funciones vulnerables de PHP

Seguidamente se muestra un estudio de las funciones utilizadas en PHP que podrían ser de riesgo para el sistema, presentando una breve descripción de las mismas y clasificándolas por los tipos de vulnerabilidades antes expuestas.

Vulnerabilidades	Funciones vulnerables	Descripciones
Denegación de servicios	exif_process_unicode	La función “exif_process_unicode” podría permitir a atacantes remotos ejecutar código arbitrario o causar una denegación de servicio (sin inicializar puntero libre y caída de la aplicación). [17]
	php_parserr	La función “php_parserr” podría provocar múltiples desbordamientos de buffer y permite que los servidores DNS remotos provoquen una denegación de servicio (caída de aplicación) o posiblemente ejecutar código arbitrario a través de un registro DNS manipulado, relacionado con la función dns_get_record y la función dn_expand. [18]
	substr_compare	La función “substr_compare” podría causar un error de entrada de validación y una denegación de servicio. [19]
	cdf_read_property_info	La función “cdf_read_property_info” podría permitir a atacantes remotos provocar una denegación de servicio (caída de aplicación) a través un archivo CDF diseñado. [20]
	cdf_count_chain	La función “cdf_count_chain” no valida correctamente los datos del sector de recuento, lo que permite a atacantes remotos provocar una denegación de servicio (caída de aplicación) a través de un archivo CDF diseñado. [21]
	cdf_check_stream_offset	La función “cdf_check_stream_offset” se basa en los datos del sector de tamaño incorrecto, lo que podría permitir a atacantes remotos provocar una

Vulnerabilidades	Funciones vulnerables	Descripciones
		denegación de servicio (caída de aplicación) a través de un desplazamiento en un archivo CDF flujo diseñado. [22]
	mconvert	La función “mconvert” podría conllevar a un desbordamiento de buffer, permitiendo a atacantes remotos provocar una denegación de servicio (caída de aplicación) a través una cadena Pascal hecha a mano en una conversión FILE_PSTRING. [23]
	cdf_read_short_sector	La función “cdf_read_short_sector” pudiera permitir a atacantes remotos provocar una denegación de servicio (error de aserción y salida de la aplicación) a través de un archivo CDF diseñado. [24]
	build_tablename	La función “build_tablename” no valida la extracción token para nombres de tabla, permitiendo a atacantes remotos provocar una denegación de servicio (NULL referencia a un puntero y caída de aplicación). [25]
	phar_rename_archive	La función vulnerable “phar_rename_archive” podría permitir a atacantes remotos provocar una denegación de servicio o posiblemente tener un impacto no especificado a través de vectores que desencadenan un intento de cambio de nombre, posiblemente, de un archivo Phar al nombre de un archivo existente. [26]
	_zip_cdir_new	La función “_zip_cdir_new” podría permitir a atacantes remotos provocar una denegación de

Vulnerabilidades	Funciones vulnerables	Descripciones
		servicio (caída de aplicación) o ejecutar código arbitrario a través de un archivo ZIP que contiene muchas entradas, dando lugar a un desbordamiento de buffer. [27]
	cdf_unpack_summary_info	La función 'cdf_unpack_summary_info' podría permitir a atacantes remotos provocar una denegación de servicio (degradación del rendimiento) mediante la activación de muchas llamadas file_printf.
Desbordamiento de buffer	date_from_ISO8601	La función “date_from_ISO8601” podría conllevar a un desbordamiento de buffer. La ejecución de mkgmtime permite a atacantes remotos provocar una denegación de servicio (caída de aplicación) a través de (1) un primer argumento diseñado a la función xmlrpc_set_type o (2) un argumento diseñado a la función xmlrpc_decode. [28]
	wordwrap	La función “wordwrap” lo cual podría causar un error de desbordamiento de buffer de datos, ya que no valida en forma correcta los datos user-supplied, por lo que podrían ser explotados por un atacante remoto o un usuario local empleando la función afectada. [29]
	array_fill	La función “array_fill” no manipula adecuadamente un dato numérico excesivamente grande, esto podría ser usado por el atacante para agotar la memoria del sistema sobrepasando los parámetros de memory_limit. [30]
Derivación	php-wrapper.fcgi	La función “php-wrapper.fcgi” no maneja

Vulnerabilidades	Funciones vulnerables	Descripciones
		adecuadamente los argumentos de línea de comandos, lo que permite a atacantes remotos evitar un mecanismo de protección en PHP 5.3.12 y 5.4.2 y ejecutar código arbitrario. [31]
	crypt	La función “crypt” en PHP 5.3.7 cuando se utiliza el tipo de hash MD5, devuelve el valor del argumento de salida en lugar de la cadena hash, lo que podría permitir a atacantes remotos evitar la autenticación a través de una contraseña arbitraria. [32]
Obtención de información	gdImageCrop	La función “gdImageCrop” podría permitir a atacantes remotos provocar una denegación de servicio (caída de aplicación) u obtener información sensible a través de una llamada a la función imagecrop con un negativo valor para la (1) x o (2) la dimensión. [33]
Corrupción de memoria	exif_ifd_make_value	Función vulnerable “exif_ifd_make_value” opera en matrices de punto flotante de forma incorrecta, lo que permite a atacantes remotos provocar una denegación de servicio (corrupción de memoria montón y caída de la aplicación) o posiblemente ejecutar código arbitrario a través de una imagen manipulada JPEG con datos de miniatura TIFF que se manejan de forma incorrecta por la función exif_thumbnail. [34]
	asn1_time_to_time_t	La función “asn1_time_to_time_t” no analiza correctamente (1) notBefore y (2) notAfter marcas de tiempo en X 0.509 certificados, lo que permite

Vulnerabilidades	Funciones vulnerables	Descripciones
		a atacantes remotos ejecutar código arbitrario o causar una denegación de servicio (corrupción de memoria) a través de un certificado diseñado que no se maneja correctamente por la función openssl_x509_parse.asn1_time_to_time_t. [35]
	openssl_x509_parse	La función “openssl_x509_parse” pudiera permitir a atacantes falsificar servidores SSL de su elección mediante un certificado elaborado emitido por una Autoridad de Certificación legítima. [36]
Obtención de privilegios	crack_opendict	La función “crack_opendict” podría permitir a usuarios locales conseguir privilegios. [37]
División de respuesta HTTP	sapi_header_op	La función “sapi_header_op” permite a atacantes remotos evitar un mecanismo de protección de la respuesta de división de HTTP a través de una URL manipulada, relacionado con la interacción inadecuada entre la función de cabecera PHP y algunos navegadores, como se ha demostrado a través de Internet Explorer y Google. [38]

Tabla 2: Funciones vulnerables que detecta el módulo de PHP.

En la Tabla 3 se encuentran funciones vulnerables de PHP que detectan las herramientas estudiadas, las cuales serán incorporadas al módulo a desarrollar.

Vulnerabilidades	Funciones	Descripciones
Inyección	mail	La función “mail” se conoce como “*Headers Mail Injection*”, consiste en aprovecharse de formularios de sitios web que utilicen esta función para el envío de los datos ingresados por el

Vulnerabilidades	Funciones	Descripciones
		visitante y que, además, estos no validen los datos en forma correcta para evitar este tipo de abusos. [39]
	pg_query	Estas funciones podrían causar vulnerabilidades de Inyección SQL.[40]
	pg_send_query	
	db2_exec	
	mysqli_stmt::execute	
	mysqli::query	
	mysqli::multi_query	
	mysqli::real_query	
	mysql_query	
	mysql_unbuffered_query	
Ejecución de código	process_nested_data	La función “process_nested_dat” permite a atacantes remotos ejecutar código arbitrario a través de un llamada unserialize diseñada que aprovecha el manejo inadecuado de las teclas numéricas duplicadas dentro de las propiedades de un objeto serializado. [41]
	object_custom	La función “object_custom” permite a atacantes remotos provocar una denegación de servicio (caída de aplicación) o posiblemente ejecutar código arbitrario a través de un argumento a la función unserialize que desencadena el cálculo de un valor de gran longitud. [42]
	com_print_typeinfo	La función “com_print_typeinf”. En PHP 5.4.3 y versiones anteriores para Windows esta función permite a atacantes remotos ejecutar código arbitrario a través de argumentos artesanales que

Vulnerabilidades	Funciones	Descripciones
		desencadenan una manipulación incorrecta de tipos VARIANT de objetos COM. [43]
Inclusión de archivos Directorio travesal	fopen	La función “fopen” en PHP 5.2.0 no controla correctamente manipuladores URL no válidos, el cual permite a atacantes dependientes del contexto evitar las restricciones safe_mode y leer archivos de su elección a través de una ruta de archivo especificado con una URL inválida. [44]
	file_get_contents	El uso de la función “file_get_contents” puede causar inclusión de archivos, permite a atacantes remotos leer archivos de su elección a través del parámetro de ruta. [45]
	copy	La vulnerabilidad reside en un error en la función de PHP “copy”. Un atacante remoto podría saltarse las restricciones de seguridad impuestas por el modo seguro (*safe mode*) y acceder a ficheros fuera de "open_basedir" mediante el envoltorio "compress.zlib://". [46]
	symlink	La función “symlink” en PHP 5.1.6 y anteriores permite a usuarios locales evitar la restricción open_basedir mediante el uso de una combinación de las funciones symlink, mkdir, y unlink, para cambiar la ruta del archivo después de la verificación open_basedir y antes de que el archivo se abra por el sistema subyacente, para después desvincular el enlace simbólico resultante. [47]

Vulnerabilidades	Funciones	Descripciones
	require	Estas funciones son utilizadas para incluir en una misma página otras páginas a la vez. Su uso podría provocar que las peticiones procesadas por el servidor web permitan a un atacante ejecutar código remoto o tomar control del equipo vulnerable. [48]
	require_once	
	includ	
	include_once	
	move_uploaded_file	La función “move_uploaded_file” en versiones de PHP anteriores de 5.4.39, 5.5.x antes de 5.5.23, y 5.6.x antes de 5.6.7, trunca un nombre de camino al encontrar un carácter \x00, lo que permite a atacantes remotos evitar restricciones de extensión prevista y crear archivos con nombres inesperados a través de un segundo argumento diseñado. [49]
XSS	utf8_decode	La función “utf8_decode” en PHP antes de 5.3.4, no controla correctamente la forma de codificación UTF-8 y subsecuencias mal formadas en datos UTF-8, que hace que sea más fácil para los atacantes remotos evitar *cross-site scripting* (XSS) y los mecanismos de protección de la inyección SQL a través de una cadena hecha a mano. [50]

Tabla 3: Funciones vulnerables de las herramientas estudiadas que se incluyen al módulo.

1.5 Métricas y clasificación de las vulnerabilidades

EL sistema de puntaje utilizado para cuantificar la severidad que puede presentar las vulnerabilidades es *Common Vulnerability Scoring System (CVSS).* Se encuentra bajo la custodia de *Forum of Incident Response and Security Teams (FIRST)*, pero se trata de un estándar completamente abierto, por lo que

puede ser utilizado libremente.

Es aplicado en bases de datos de vulnerabilidades públicamente conocidas como *National Vulnerability Database (NVDB)*, *Common Vulnerabilities and Exposures (CVE)* u *Open Source Vulnerability Database (OSVDB).*

Para determinar el impacto que representa una vulnerabilidad se utiliza una escala que va del **0 al 10.** La severidad se considera **baja** si el puntaje obtenido luego de aplicar la fórmula CVSS resulta entre 0.0 y 3.9. El impacto es **medio** si el resultado se ubica entre 4.0 y 6.9. Se considera **alto** cuando el puntaje cae dentro del rango 7.0 y 10.0.

Para calcular un puntaje asociado a una vulnerabilidad, CVSS utiliza tres **grupos de métricas: base**, **temporal** y de **entorno**, cada una se conforma a su vez de un conjunto de otras métricas. [51]

Para determinar las puntuaciones de las vulnerabilidades se tuvo en cuenta solo las métricas de **base**, ya que según la Guía de CVSS versión 2.0, con esta métrica es suficiente para una correcta identificación de las clasificaciones necesarias, se plantea que las demás son opcionales.

Las métricas de base engloban las características de una vulnerabilidad que permanece constante, entre estas características se encuentran:

1. Métricas de Explotabilidad

- **Vector de acceso:** Local, Remoto.
- **Complejidad de ataque: B**aja, Alta.
- **Nivel de autenticación requerida:** No requerida, Requerida.

2. Métricas de Impacto

- **Impacto en la confidencialidad:** Ninguna, Parcial, Completa.
- **Impacto en la integridad:** Ninguna, Parcial, Completa.
- **Impacto en la disponibilidad:** Ninguna, Parcial, Completa.
- **Mayor impacto en:** Confidencialidad, Integridad, Disponibilidad.

Mediante la siguiente ecuación definida por CVSS se realizó el proceso de puntuación.

Métrica Base = round_to_1_decimal (((0.6 * Impacto) + (0.4 * Explotabilidad) – 1.5) * f (Impacto))

Explotabilidad = 20 * Vector_de_acceso * Complejidad_de_acceso * Autenticación

Impacto = 10.41 * (1 – (1 – Confidencialidad) * (1 – Integridad) * (1 – Disponibilidad))

La aplicación de este estándar arrojó las siguientes clasificaciones:

- **Advertencia** para las vulnerabilidades de tipo: denegación de servicios, ejecución de código, desbordamiento de buffer, obtención de información, XSS, directorio travesal, inyección SQL y división de respuesta HTTP.
- **Información** para las vulnerabilidades de tipo: derivación, corrupción de memoria, obtención de privilegios e inclusión de archivos.

1.6 Breve revisión de ACF

La herramienta ACF permite la detección de vulnerabilidades en el código fuente de las aplicaciones desarrolladas en los lenguajes de programación C/C++, Perl, Python y Bash; que se encuentran en el repositorio de Nova. También incluye un módulo especial para la búsqueda de posibles ataques políticos hacia Cuba.

Para el uso de ACF no es necesario poseer conocimientos profundos del *software*, pero debe garantizarse un nivel de instrucción informático aceptable que permita la rápida asimilación del mismo. Brinda soporte para el idioma español e inglés. [52]

1.7 Tecnologías a utilizar en el desarrollo de la herramienta.

Para la realización de la herramienta Auditoría de Código Fuente (ACF), fue empleada la metodología de desarrollo de *software* OpenUP y el lenguaje de programación C++; por lo que para seguir con la misma línea de desarrollo, son escogidas para el proceso de elaboración del Módulo de PHP.

OpenUP es un proceso unificado que aplica enfoques iterativos e incrementales dentro de un ciclo de vida estructurado. Utiliza una filosofía ágil que contiene el conjunto mínimo de prácticas para ayudar al equipo de desarrollo de *software* a realizar un producto de alta calidad y de una forma eficiente. Esta metodología va dirigida a equipos pequeños que trabajan juntos en la misma ubicación. De igual manera, permite disminuir las posibilidades de riesgo y descubrir errores tempranos. [53] Es un modelo de desarrollo de *software* y parte del Framework de modelo de proceso de Eclipse (*Eclipse Process Framework*),

desarrollado por la fundación Eclipse. [54]

C ++ fue diseñado para apoyar la abstracción de datos, programación orientada a objetos y programación genérica, además de las técnicas de programación C tradicionales en virtud de esas limitaciones [55]. Se define con un nivel intermedio, pues contiene características tanto de alto como de bajo nivel. Es uno de los lenguajes más populares en programación por su carácter híbrido, sirviendo para programación estructurada y para la orientada a objetos. [56]

Es utilizado el lenguaje de etiquetas **XML**, el cual es una adaptación del SGML (*Standard Generalized Markup Language*), que permite la organización y el etiquetado de documentos. [57] Los expertos señalan varias ventajas que derivan de la utilización del XML, como por ejemplo que: es extensible (se pueden añadir nuevas etiquetas tras el diseño del documento); su analizador es estándar (no requiere de cambios para cada versión del metalenguaje); facilita el análisis y el procesamiento de los documentos XML creados por terceros. [58]

El Lenguaje Unificado de Modelado **UML 2.0** es empleado para especificar, construir, visualizar y documentar los artefactos del sistema de *software*. Este lenguaje es utilizado mediante la herramienta CASE: **Visual Paradigm para UML v8.0**, la cual brinda soporte al ciclo de vida completo del desarrollo de *software*. Este es un producto galardonado que facilita la diagramación visual y el diseño de sistema. Emplea una rápida respuesta con poca memoria utilizando moderadamente los tiempos del procesador, lo que le permite manejar grandes y complicadas estructuras de un proyecto en una forma muy eficiente. [61]

Para la implementación del sistema se utiliza el Entorno Integrado de Desarrollo (IDE): **Eclipse v3.8**, debido a que es una potente y completa plataforma de programación, desarrollo y compilación de elementos tan variados como sitios web, programas en C++ o aplicaciones Java. Es un entorno de desarrollo integrado, multiplataforma y de código abierto basado en Java. [59] Su concepto de trabajo está determinado por las perspectivas, que no es otra cosa que una preconfiguración de ventanas y editores, relacionadas entre sí, y que permiten trabajar en un determinado entorno de forma óptima. [60]

Es empleada la herramienta **Doxygen v1.8.6** para la documentación del código, producto a que realiza un análisis en busca de directivas en forma de comentarios y los transforma en documentación. También

acepta multitud de lenguajes, incluyendo C/C++. Una de sus grandes cualidades es que el usuario tiene una variedad de formatos de documentación para elegir, entre ellos: HTML, PDF y RTF. [62]

Para realizar el control de las versiones y registrar los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, se utiliza el sistema **Git** en su **versión 1.7.9.5**.

Los sistemas de control de versiones permiten revertir archivos o proyectos enteros a un estado anterior, comparar cambios, observar quién modificó por última vez algo que puede estar causando un problema, quién introdujo un error y cuándo. Git es un sistema 100% distribuido, rápido y no depende de acceso a la red o un repositorio central. Se enfoca en la velocidad, uso práctico y manejo de proyectos grandes. [63]

Conclusiones parciales

En este capítulo se abordaron los conceptos fundamentales, los cuales otorgan conocimientos relacionados con la investigación. Se realizó un estudio sobre las herramientas dedicadas al escaneo de fallas de seguridad en el código fuente de PHP, permitiendo determinar los tipos de vulnerabilidades en que se enfatizará la búsqueda del sistema a desarrollar. Se llevó a cabo un análisis de las posibles funciones vulnerables que se incorporarán al sistema, lo que permitirá realizar la detección de vulnerabilidades. Además se analizaron las diferentes tecnologías, herramientas y metodologías utilizadas en la actualidad, seleccionando los lenguajes de programación XML y C++, junto al IDE Eclipse para la implementación de las funcionalidades necesarias. La metodología OpenUP ayudará para guiar el proceso de desarrollo del *software*, mientras que la herramienta CASE Visual Paradigm será empleada para realizar el modelado de los diagramas.

Capítulo 2: Análisis y diseño del Módulo de PHP.

Introducción

En el capítulo se presenta una propuesta de solución al problema de investigación planteado. Para conocer las características y funcionalidades que debe proporcionar el *software*, se definen los requisitos funcionales y no funcionales. Además de exponer la arquitectura y los patrones de diseño utilizados en el sistema. De igual manera, se muestran los diagramas que sirven de guía para el proceso de implementación.

2.1 Propuesta de solución

Para darle solución al problema existente en la detección de fallas de seguridad del código fuente de PHP del repositorio de Nova, se propone desarrollar un módulo para la herramienta ACF. Mediante un análisis estático del código será capaz de realizar búsquedas de vulnerabilidades para evitar ser víctimas de ataques y permitir mayor seguridad en las aplicaciones o librerías del repositorio. Las funciones vulnerables serán presentadas en un reporte resumen, clasificándolas en advertencias o informaciones, según el nivel de afectación que pueda causar al sistema. Además generará un reporte detallado con las vulnerabilidades detectadas, la descripción de los daños que puedan ocasionar y la dirección del fichero donde se encuentra; con el objetivo de que el mantenedor de paquetes o el programador sean los encargados de contrarrestar el riesgo.

2.2 Fases de OpenUP

La metodología OpenUP consta de cuatro fases que guían el proceso de desarrollo del *software*: inicio, elaboración, construcción y transición.

Fase de inicio: En esta fase las necesidades de cada participante del proyecto son tomadas en cuenta y plasmadas en objetivos del proyecto. Se definen para el proyecto: el ámbito, los límites, el criterio de aceptación, los casos de uso críticos, una estimación inicial del coste y un boceto de la planificación. [64]

Fase de elaboración: Se realizan tareas de análisis del dominio y definición de la arquitectura del sistema. Se debe elaborar un plan de proyecto, estableciendo unos requisitos y una arquitectura estables. Por otro lado, el proceso de desarrollo, las herramientas, la infraestructura a utilizar y el entorno de desarrollo también se especifican en detalle en esta fase. Al final de la fase se debe tener una definición clara y precisa de los casos de uso, los actores, la arquitectura del sistema y un prototipo ejecutable de la misma. [64]

Fase de construcción: Son realizados, probados e integrados todos los componentes y funcionalidades del sistema que falten por implementar. Los resultados obtenidos en forma de incrementos ejecutables deben ser desarrollados de la forma más rápida posible teniendo en cuenta la calidad de lo desarrollado. [64]

Fase de transición: La fase de la transición consta de las subfases de pruebas de versiones beta, pilotaje y capacitación de los usuarios finales y de los encargados del mantenimiento del sistema. En función de la respuesta obtenida por los usuarios puede ser necesario realizar cambios en las entregas finales o implementar alguna funcionalidad más. [64]

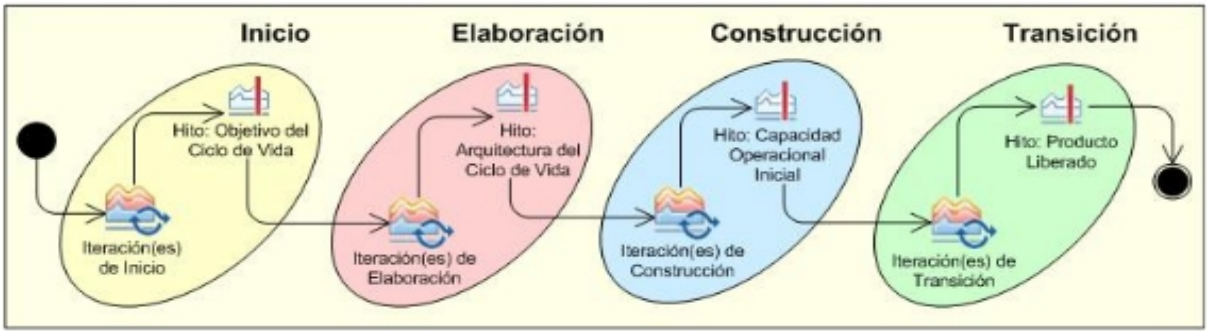

Figura 4: Ciclo de vida de OpenUP. [64]

2.3 Requisitos funcionales y no funcionales del sistema

Mediante la identificación de requisitos funcionales y no funcionales que a continuación se presentan, se podrá obtener las necesidades que debe cumplir el módulo a desarrollar.

2.3.1 Requisitos funcionales

R1 -Detectar vulnerabilidades en el código fuente de PHP.

- **R1.1-** Definir extensiones de los archivos de código fuente PHP.
- **R1.2-** Detectar vulnerabilidades que conlleven a una denegación de servicio.
- **R1.3-** Detectar vulnerabilidades que provoquen ejecución de código.
- **R1.4-** Detectar vulnerabilidades que provocan desbordamiento de buffer.
- **R1.5-** Detectar vulnerabilidades correspondientes a derivación.
- **R1.6**- Detectar vulnerabilidades que impliquen la obtención de información.
- **R1.7-** Detectar vulnerabilidades que ocasionen la corrupción de memoria.
- **R1.8-** Detectar vulnerabilidades de tipo XSS.
- **R1.9-** Detectar vulnerabilidades de tipo directorio travesal.
- **R1.10-** Detectar vulnerabilidades de tipo que puedan causar inyección SQL.
- **R1.11-** Detectar vulnerabilidades que conlleven a la obtención de privilegios.
- **R1.12-** Detectar vulnerabilidades de tipo división de respuesta HTTP.
- **R1.13-** Detectar vulnerabilidades de tipo que permita la inclusión de archivos.
- **R1.14-** Generar reporte de vulnerabilidades encontradas.

2.3.2 Requisitos no funcionales

Usabilidad

- El módulo podrá ser utilizado por todo aquel usuario que desee verificar el código fuente PHP de cualquier aplicación o librería que se encuentre en el repositorio de Nova.

Apariencia o interfaz externa

- El módulo no presenta una interfaz externa, debido a que la herramienta ACF será la encargada de proporcionar la misma.

Software

- Sistema operativo GNU/Linux Nova 2013.

Restricciones de diseño e implementación

- Utilizar como lenguaje de programación C++.
- El estilo de programación debe ser el especificado en la Guía de estilo para lenguaje C++.

2.4 Definición del actor y casos de uso del sistema

2.4.1 Definición del actor del sistema

Mediante esta tabla se define el actor del sistema y la descripción de las actividades que realiza.

Actores	Justificación
Usuario	Es la persona que tiene como necesidad detectar vulnerabilidades en el código fuente de PHP de una aplicación.

Tabla 4: Actor del sistema.

2.4.2 Vista Caso de uso

Diagrama de Casos de Uso del Sistema

Figura 5: Diagrama de casos de uso.

2.4.3 Descripciones de los casos de uso del sistema

A continuación se realizará una descripción textual de los casos de usos del sistema que han sido representados en el diagrama anterior; obteniendo el esclarecimiento de los componentes, acciones y reacciones del sistema.

Caso de uso # 1	Detectar vulnerabilidades en el código fuente de PHP
Actores	Usuario
Resumen	El caso de uso inicia cuando el usuario decide detectar las vulnerabilidades que existen en el código fuente PHP de uno o varios ficheros. Finaliza cuando son detectadas y reportadas correctamente.
Precondiciones	
Referencias	R1
Complejidad	Alta
Prioridad	Alta
Postcondiciones	Detecta vulnerabilidades en el código fuente de PHP.

Flujo normal de eventos		
#	**Acción del Actor**	**Respuesta del Sistema**
1	Establece la dirección donde se encuentran los ficheros de código fuente de PHP y el lugar donde se almacenará el reporte.	3- El módulo de PHP obtiene los ficheros para detectar sus vulnerabilidades.
2	Solicita la revisión de los ficheros para detectar vulnerabilidades en el código fuente de PHP mediante el botón “Auditar”.	4- Comprueba que los ficheros corresponden al código fuente PHP.
		5- Se ejecuta el Módulo de PHP.
		6- Realiza la revisión de las funciones vulnerables en el código fuente que puedan conllevar a las siguientes vulnerabilidades: - Denegación de servicio. - Ejecución de código. - Desbordamiento de buffer. - Derivación. - Obtención de información.

		- Corrupción de memoria. - XSS. - Directorio travesal. - Inyección SQL. - Obtención de privilegios. - División de respuesta HTTP. - Inclusión de archivos.
		7- Genera un reporte resumido con las cantidades de vulnerabilidades encontradas en los ficheros, clasificándolas en advertencias o informaciones.
		8- Genera un reporte detallado con la función vulnerable detectada, su descripción, la dirección del fichero y la línea de código donde se encuentra.
Flujo alterno 1		
	Acción del Actor	**Respuesta del Sistema**
		4 a 1. Si el código fuente no pertenece a PHP, el módulo no realiza ninguna detección de vulnerabilidades sobre dicho archivo.

Tabla 5: Descripción textual del caso de uso “Detectar vulnerabilidades en el código fuente de PHP”.

2.5 Patrones de diseño

En la terminología de objetos, el patrón es una descripción de un problema y su solución, que recibe un nombre y que puede emplearse en otros contextos. Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular. El uso de patrones ayuda a obtener un *software* de calidad (reutilización y extensibilidad). [66]

2.5.1 Patrones GRASP

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a

objetos, expresados en forma de patrones. GRASP es un acrónimo que significa *General Responsibility Asignment Software Patterns* (patrones generales de *software* para asignar responsabilidades). [67]

Seguidamente se presentan los patrones GRASP utilizados en el Módulo de PHP:

Creador

El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento. [67]

Este patrón de diseño se evidencia mediante el siguiente fragmento de código de la aplicación, donde la clase interfaz acf-php es considerada la idónea para asumir la responsabilidad de crear objetos de otras clases en el desarrollo de sus funciones.

```
Php* php;
php = new Php(TipoPlugin(1), destiny, param);
php->Execute(globalValues);
```

Alta Cohesión

La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. [67]

Beneficios:

- Mejoran la claridad y facilidad con que se entiende el diseño.
- Se simplifica el mantenimiento y las mejoras de funcionalidad.
- A menudo se genera un bajo acoplamiento.
- Soporta mayor capacidad de reutilización.

Este patrón se evidencia en la clase **Php**, la cual tiene implementada las funciones necesarias para realizar su trabajo y colabora con otras para llevar a cabo sus tareas.

Figura 6: Representación del patrón Alta Cohesión.

Bajo Acoplamiento

El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, que conoce y recurre a ellas. Acoplamiento bajo significa que una clase no depende de muchas clases. [67]

Beneficios:

- No se afectan por cambios de otros componentes.
- Fáciles de entender por separado.
- Fáciles de reutilizar.

Este patrón se utilizó con el objetivo de tener las clases menos relacionadas entre sí, como es el caso de la clase interfaz **acf-php**, la cual no depende de otras clases para realizar sus responsabilidades.

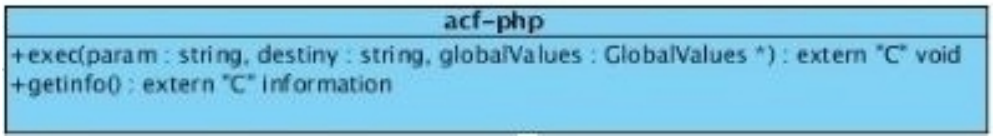

Figura 7: Representación del patrón Bajo Acoplamiento.

2.6 Arquitectura de software

ACF utiliza una arquitectura de *software* **n-capas,** permitiendo una mayor flexibilidad en las soluciones, pues se pueden añadir nuevos módulos para dotar al sistema de funcionalidades. La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido sin tener que revisar entre código mezclado.

Las capas que presenta son: **Vista**, **Lógica del negocio** y **Acceso a datos**.

La **Vista** es la que presenta el sistema, donde se notifica y se captura la información del usuario en un mínimo de proceso. Esta capa se comunica únicamente con la capa de negocio.

En **Lógica del negocio** es donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de Vista, para recibir las solicitudes y presentar los resultados, y con la capa Acceso a datos, para solicitar al gestor de base de datos almacenar o recuperar datos de él. En esta capa se le incluye la **arquitectura basada en Componentes**, por lo que el Módulo de PHP se acopla a la misma formando parte de estos componentes.

En la capa de **Acceso a datos** es donde se encuentran los datos y es la encargada de acceder a los mismos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

Arquitectura basada en Componente

La arquitectura basada en componentes se utiliza para descomponer la herramienta en módulos funcionales, para así dar origen al principio de reutilización del *software* y permitir su utilización sobre diferentes sistemas en el futuro. Esta arquitectura se agrupa en el paquete Plugin_Subsystem, almacenando los paquetes de código fuente de los componentes de auditoría en la carpeta LanguageSource.

2.7 Vistas arquitectónicas de OpenUP

OpenUP propone una arquitectura basada en 3 vistas: Lógico, Operacional y Caso de uso, las cuales se emplearán para conducir el desarrollo de la solución del *software*.

Lógico: En esta vista se describe la estructura y el comportamiento de las partes arquitectónicamente significativos del sistema. Esto podría incluir la estructura de paquetes, interfaces críticas, clases y subsistemas importantes, y las relaciones entre estos elementos. También incluye vistas físicas y lógicas de datos persistentes, si la persistencia se construirá en el sistema. Este es un subconjunto documentado del diseño.

Operacional: Describe los nodos físicos del sistema y de los procesos, hilos, y componentes que se ejecutan en los nodos físicos. Este punto de vista no es necesario si el sistema se ejecuta en un proceso único e hilo.

Caso de uso: Una lista o diagrama de los casos de uso que contienen requisitos arquitectónicamente

significativos.

2.8 Vista Lógica

2.8.1 Diagrama de paquetes

Los diagramas de paquetes UML muestran la organización de los paquetes, perfiles y espacios de nombres. Los diagramas de paquetes se suelen usar para mostrar la organización de alto nivel de un proyecto de *software*. Los paquetes y diagramas de paquetes suelen venir muy bien a la hora de organizar y documentar subproyectos reutilizables. [68]

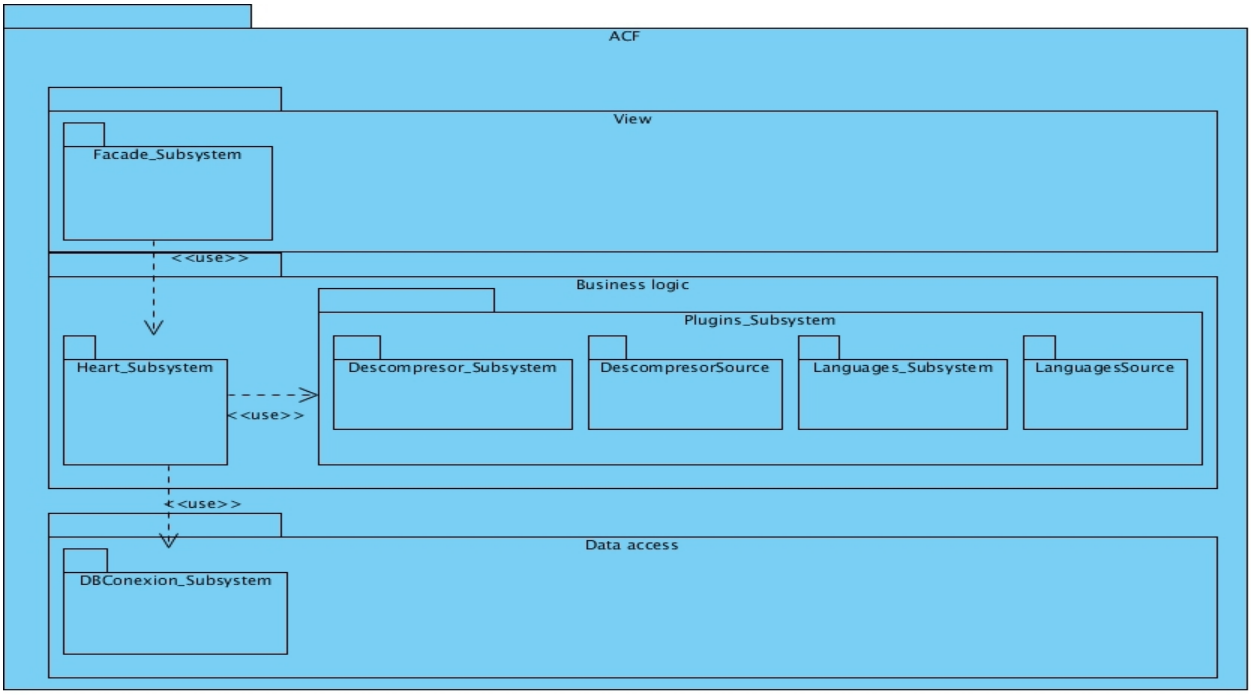

Figura 8: Diagrama de paquetes.

Capa de Vista: Esta capa contiene la carpeta Facade_Subsystem, la cual provee la interfaz de la herramienta.

Capa de Lógica del Negocio: Proporciona los componentes del *software*, donde el Módulo de PHP se

encuentra implementado formando parte de la carpeta **LanguagesSource**. Este módulo cuenta con la interfaz **acf-php.cpp**, la cual es la encargada de vincularse con la herramienta. Posee la clase **Php.cpp** que detecta las posibles vulnerabilidades del código, utilizando las declaraciones de constantes, variables y funciones declaradas en el archivo de encabezado **php.h**.

Capa de Acceso a Datos: Presenta la carpeta DBConexion_Subsystem, donde se encuentra el trabajo con las bases de datos y el repositorio.

2.9 Diagrama de clases

El propósito de este diagrama es el de representar los objetos fundamentales del sistema, es decir los que percibe el usuario y con los que espera tratar para completar su tarea en vez de objetos del sistema o de un modelo de programación. [69]

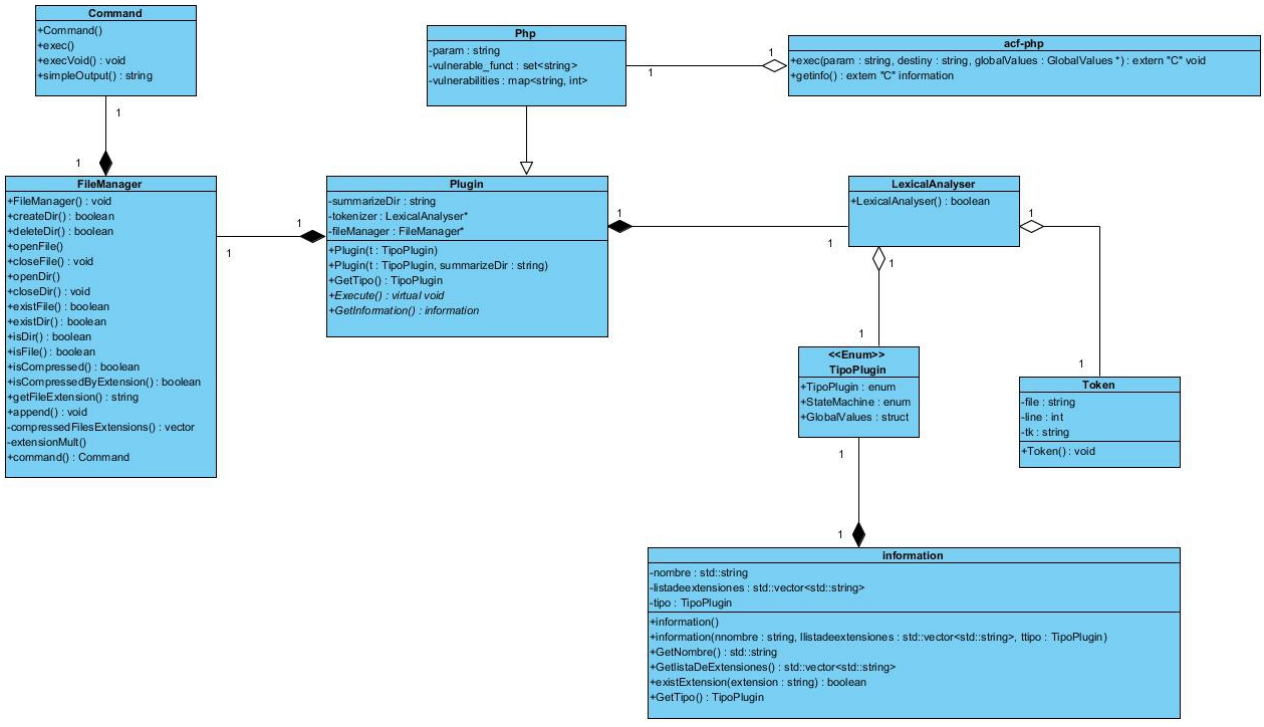

Figura 9: Diagrama de clases.

2.10 Diagramas de interacción

Los diagramas de interacción son algunos de los artefactos que se preparan en el análisis y diseño orientados a objetos. Existen dos tipos de diagramas de interacción: el Diagrama de Colaboración y el Diagrama de Secuencia.

2.10.1 Diagramas de secuencia

Los diagramas de secuencia son más adecuados para observar la perspectiva cronológica de las interacciones. Permite diseñar el comportamiento del *software* y sirve, entre otras cosas, para diseñar casos de uso. [70]

El diagrama correspondiente al caso de uso “Detectar vulnerabilidades en el código fuente de PHP”, se puede consultar en el Anexo 11.

Conclusiones parciales

Luego de haber propuesto el desarrollo de un módulo dedicado a la detección de vulnerabilidades en códigos fuente PHP para ACF, se pudieron definir los requisitos funcionales y no funcionales, los cuales permitieron establecer las necesidades a alcanzar. Fueron abordados y detallados los casos de uso del sistema, lo que permitió explicar paso a paso las acciones del actor y las respuestas que ofrece el sistema. Mediante los diagramas realizados se pudo obtener una representación detallada de la organización, comportamiento y funcionalidades del *software* a implementar.

Capítulo 3: Implementación y pruebas del Módulo de PHP.

Introducción

Este capítulo es dedicado a la implementación del módulo propuesto. Se describen los estándares de codificación que permiten la organización y comprensión de las líneas de códigos. Además de enfocarse en las pruebas que se realizan para comprobar el correcto funcionamiento del sistema desarrollado.

3.1 Diagrama de componentes

Los diagramas de componentes muestran los componentes del *software*, ya sea las tecnologías o simplemente secciones del sistema claramente distintas y los artilugios de que está compuesto, como los archivos de código fuente, las librerías o las tablas de una base de datos. [72]

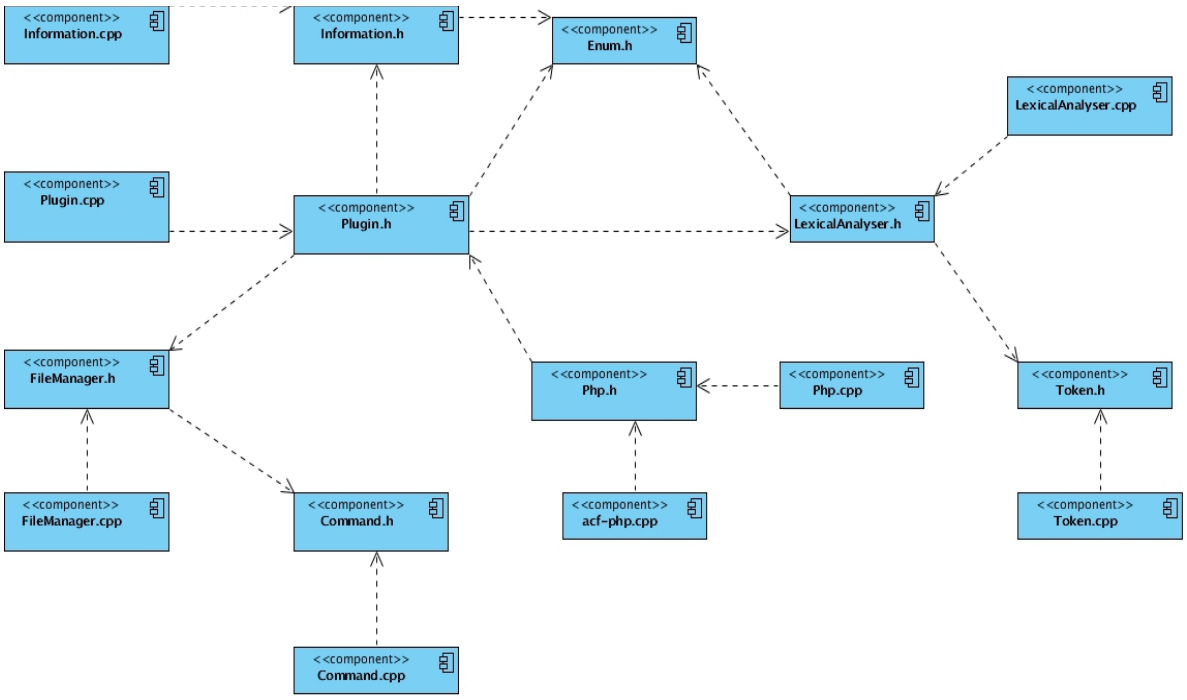

Figura 10: Diagrama de componentes.

3.2 Justificación del estándar de codificación

Los estándares de codificación son pautas de programación que permiten entender de manera rápida, fácil y sencilla, el código empleado en el desarrollo del *software*. El uso de estos estándares tiene innumerables ventajas, entre ellas:

1. Asegurar la legibilidad del código entre distintos programadores, facilitando la depuración del mismo.
2. Proveer una guía para el encargado de mantenimiento/actualización del sistema, con código claro y bien documentado.
3. Facilitar la portabilidad entre plataformas y aplicaciones.[73]

Variables o atributos

- Comenzarán con letra minúscula.
- Si están formados por varias palabras, la primera letra de cada palabra se pondrá en mayúsculas excepto la primera.
- No se utilizarán números ni '_'.

Indentación

- La indentación es de cuatro espacios, sin tabulador, para que cualquier editor de texto reconozca correctamente.

Ejemplo:

```
if(info > 0 || warning > 0)
    {
    ....fileManager->append(sumarizedir, "%s\n", param.c_str());
        fileManager->append(sumarizedir, "-> ADVERTENCIAS = %d\n", warning);
        fileManager->append(sumarizedir, "-> INFORMACIONES = %d\n\n", info);
    }
```

Estructuras de control

- Las llaves "{" que abren el cuerpo de un bucle, ya sea *if* o *while*, se situarán debajo de la sentencia

y con un sangrado adicional de forma que queden alineadas con el código que depende de estas sentencias. La llave que cierra el bloque se situará en la línea siguiente a la última línea del bloque y con un nivel de sangrado igual al de las sentencias que encierra.

Ejemplo:

```
while ((token = tokenizer->nextToken()) != NULL)
      {
            if (vulnerable_funct.count(token->getTK()) != 0)
            {
```

Comentarios o información del código

- Se utilizan los estilos de C (/* */) y C++ (//) para comentar el código.
- Los comentarios no deben salirse de un ancho de 80 columnas, que es normal en la mayor parte de las pantallas.
- Al escribir un comentario se deja un espacio entre el * de apertura y el texto, y otro entre el texto y el * de cierre.
- El cajetín del fichero contendrá como mínimo la siguiente información:

  -Nombre del fichero

  -Autores

  - Fecha de creación

Ejemplo del cajetín sencillo:

```
/*
 * Php.cpp
 *
 *  Created on: 05/03/2015
 *      Author: mayvis rivet martos
 */
```

Otras normas

- Después de cada signo de puntuación, "," o ";" se deja un espacio, pero nunca antes.

- No se deja un espacio después de abrir un paréntesis ni antes de cerrarlo.

3.3 Pruebas de software

Las pruebas son técnicas de comprobación dinámica, siempre implican la ejecución del programa. Permiten evaluar la calidad de un producto y mejorarlo identificando los problemas que presenta. [74]

3.3.1 Niveles

A la hora de evaluar dinámicamente un sistema se ha de seguir estrategias, las cuales deben permitir comenzar por los componentes más simples y más pequeños e ir avanzando progresivamente hasta probar todo el *software* en su conjunto. Las pruebas se llevan a cabo por los siguientes niveles:

1. **Pruebas Unitarias:** Comienzan con la prueba de cada módulo.
2. **Pruebas de Integración:** A partir del esquema del diseño, los módulos probados se vuelven a probar combinados para probar sus interfaces.
3. **Prueba del Sistema:** El *software* ensamblado totalmente con cualquier componente hardware que requiere se prueba para comprobar que se cumplen los requisitos funcionales.
4. **Pruebas de Aceptación.** El cliente comprueba que el *software* funciona según sus expectativas. [74]

Al Módulo de PHP se le realizarán **Pruebas del Sistema y de Aceptación.** En el nivel de Sistema se tiene como propósito ejercitar profundamente el programa para verificar que se han integrado adecuadamente todos los elementos y que realizan las funciones adecuadas. El *software* debe integrarse con los componentes de *hardware* correspondientes y se ha de comprobar el funcionamiento del sistema completo acorde a los requisitos. [75]

Finalmente, se realizarán las Pruebas de Aceptación sobre el sistema completo. A la hora de realizarlas, el producto está listo para implantarse en el entorno del cliente. El usuario debe ser el que realice las pruebas, ayudado por personas del equipo. El objetivo de este tipo de prueba es comprobar si el cliente está satisfecho con el producto desarrollado y si este producto cumple con sus expectativas, en términos de los errores que genera y de la funcionalidad que suministra. [75]

3.3.2 Métodos

Las técnicas de evaluación dinámica o prueba, proporcionan distintos criterios para generar casos de prueba que provoquen fallos en los programas. Existen dos métodos para probar un sistema construido:

- **Método de caja blanca o estructurales:** Se basan en un minucioso examen de los detalles procedimentales del código a evaluar, por lo que es necesario conocer la lógica del programa. [75]
- **Método de caja negra o funcionales:** Se realizan pruebas sobre la interfaz del sistema a probar, entendiendo por interfaz las entradas y salidas del mismo. No es necesario conocer la lógica del sistema, únicamente la funcionalidad que debe realizar. [75]

Las pruebas que se llevarán a cabo serán las de **caja negra,** las cuales se aplican a la interfaz del *software*. Se examinarán los aspectos funcionales del sistema, buscando asegurar que se ha ingresado toda clase de entrada y que la salida observada sea igual a la esperada. Estas pruebas se basan en la especificación del programa o componente a ser probado para elaborar los casos de prueba.

Seguidamente se seleccionará la técnica para realizar estas pruebas de caja negra y así examinar el funcionamiento del sistema.

3.3.3 Técnicas

Para desarrollar la prueba de Caja Negra existen varias técnicas, entre ellas están:

- Técnica de la Partición de Equivalencia.
- Técnica del Análisis de Valores Límite.
- Técnica de Grafos de Causa-Efecto.

Dentro del método de Caja Negra la técnica de la **Partición de Equivalencia** es una de las más efectivas, pues permite examinar los valores válidos e inválidos de las entradas existentes en el *software*, descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de pruebas que descubran clases de errores, reduciendo así en número de clases de prueba que hay que desarrollar. [75]

Esta técnica de partición de equivalencia será empleada mediante la realización del diseño de caso de

prueba siguiente, para poder comprobar el comportamiento del sistema.

3.3.4 Diseño de Casos de Pruebas

Escenario	Descripción	Dir. Código fuente	Dir. Reporte	Respuesta del sistema	Flujo Central
EC1.1 Se detectan la extensiones de los ficheros .php	1- El usuario añade el directorio donde se encuentran ficheros de código fuente PHP. 2-El usuario añade la dirección donde se almacenará el reporte de vulnerabilidades detectadas. 3- Presiona el botón “Auditar”.	V /temp/Symfony	V /media/Datos	1- El sistema reconoce los ficheros de código fuente PHP. 2- Muestra el mensaje: “Buscando vulnerabilidades”. Ver Anexo 1.	1. El usuario añade el directorio donde se encuentra el código fuente PHP. 2- El usuario añade el directorio donde se almacenará el reporte de vulnerabilidades. 3- El usuario presionará el botón “Auditar”.
EC1.2 Realizando auditoría dejando el campo del directorio de código fuente vacío.	1- El usuario añade la dirección donde se almacenará el reporte de vulnerabilidades detectadas. 2- Presiona el botón “Auditar”.	I Vacío	V /media/Datos	1- El sistema muestra el mensaje de error: “El campo dirección está vacío”. Ver Anexo 2.	
EC1.3 Realizando	1- El usuario añade el directorio donde se	V /temp/Symfony	I Vacío	1- El sistema muestra el mensaje	

Escenario	Descripción	Dir. Código fuente	Dir. Reporte	Respuesta del sistema	Flujo Central
auditoría dejando el campo del directorio del reporte vacío.	encuentran ficheros de código fuente PHP. 2- Presiona el botón “Auditar”.			de error: “El campo dirección está vacío”. Ver Anexo 3.	
EC1.4 Realizando auditoría dejando los campos vacíos.	1- El usuario presiona el botón “Auditar”.	I Vacío	I Vacío	1- El sistema muestra el mensaje de error: “EL campo de dirección está vacío”. Ver Anexo 4	
EC1.5 El sistema audita ficheros de código fuente PHP y detecta vulnerabilidades.	1- El usuario añade el directorio donde se encuentran ficheros de código fuente PHP. 2-El usuario añade la dirección donde se almacenará el reporte de vulnerabilidades detectadas. 3- Presiona el botón “Auditar”.	V /temp/Symfony	V /media/Datos	1- El sistema audita los ficheros de código fuente PHP. 2- Muestra la dirección donde se encuentra el fichero vulnerable, y clasifica las funciones en Advertencias o Informaciones. Ver Anexo 5. 3- Genera dos reportes en la dirección dada por el usuario, uno	

Escenario	Descripción	Dir. Código fuente	Dir. Reporte	Respuesta del sistema	Flujo Central
				detallado y el otro resumido.	
EC1.6 El sistema audita ficheros de código fuente PHP y no detecta vulnerabilidades.	1- El usuario añade el directorio donde se encuentran ficheros de código fuente PHP. 2-El usuario añade la dirección donde se almacenará el reporte de vulnerabilidades detectadas. 3- Presiona el botón “Auditar”.	V /temp/Symfony	V /media/Datos	1- El sistema audita los ficheros. 2- Muestra el mensaje: “No se detectaron vulnerabilidades”. Ver Anexo 6. 3- Genera el reporte detallado y el resumido.	
EC1.7 Realizando auditoría con campos incorrectos.	1- El usuario añade el directorio incorrectamente. 2- Presiona el botón “Auditar”.	I Incorrecto/	V /media/Datos	1- El sistema muestra el mensaje de error: “El campo dirección es erróneo”. Ver Anexo 7	

Tabla 6: Caso de Prueba- Detectar vulnerabilidades en el código fuente de PHP.

3.4 Resultados de las pruebas funcionales

Para evaluar la calidad del sistema y mejorarlo según los errores detectados, se realizaron pruebas a nivel de sistema. Haciendo uso del método de caja negra se lleva a cabo la técnica de partición equivalente, permitiendo encontrar no conformidades mediante los casos de pruebas diseñados.

En la siguiente gráfica quedan concretadas las 3 iteraciones de pruebas funcionales realizadas, especificando las no conformidades detectadas, resueltas y las que permanecen pendientes para su corrección. Se obtuvo un total 10 no conformidades, donde los principales errores se relacionaban con la ortografía y concordancia en las descripciones de las funciones vulnerables.

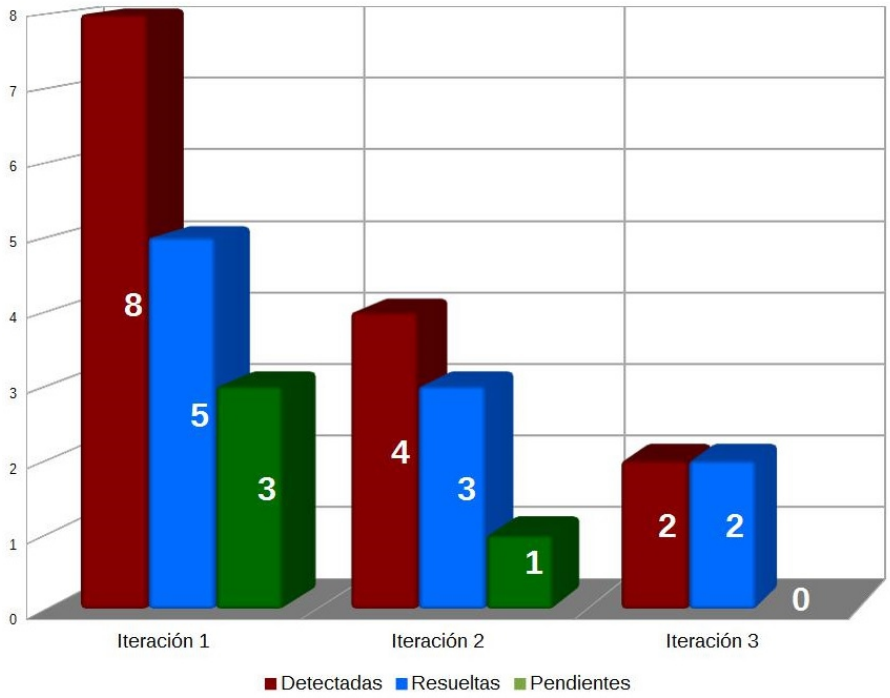

Figura 11: Resultados de las pruebas funcionales.

Se efectuaron auditorías reales al paquete de Symfony Standard Vendors 2.5.0, obteniendo como resultado un total de 561 vulnerabilidades, de las cuales 557 clasificadas en advertencias debido a su alto nivel de riesgo y 4 informaciones por daños menores. Las funciones vulnerables que detectó el módulo fueron comprobadas en los ficheros .php donde pertenecían, lo cual demostró que no se estaba en presencia de falsos positivos.

La siguiente gráfica fue generada por la herramienta ACF para representar los resultados de la auditoría realizadas.

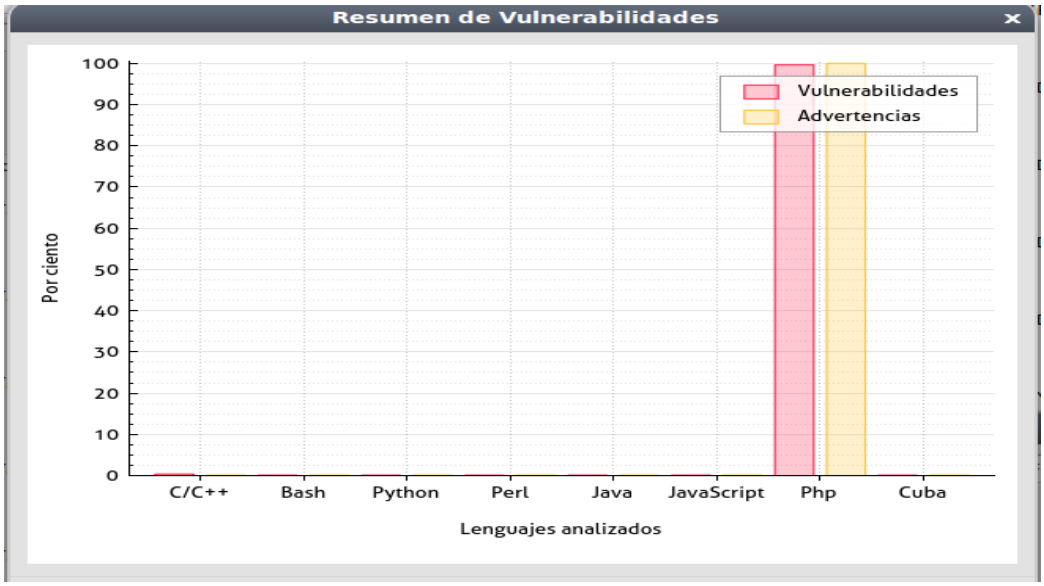

Figura 12: Resultado de la auditoría a Symfony.

A continuación se presenta ejemplos de la auditoría al paquete de Symfony, el reporte generado por el sistema y la función vulnerable dentro del código PHP.

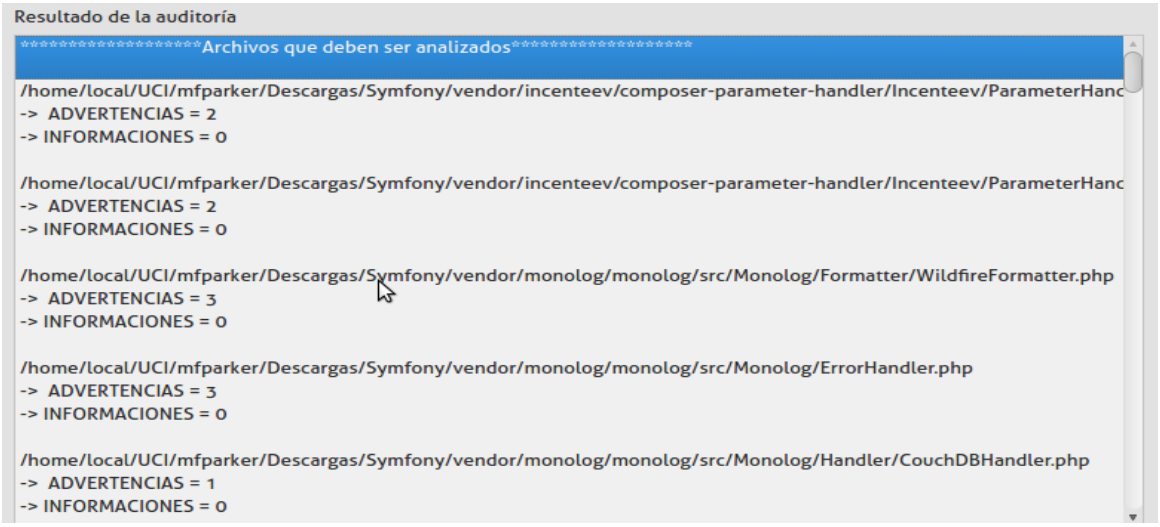

Figura 13: Auditoría a el paquete de Symfony.

En la figura 14 se puede observar el resultado de la auditoría, donde se muestra la dirección de los archivos analizados y la cantidad de vulnerabilidades según su clasificación. Mediante la figura 15 se puede obtener un reporte más detallado de la auditoría, encontrando la función vulnerable junto a su descripción, la dirección del código fuente php y la línea a que pertenece dicha función.

```
1 *----------------| /home/local/UCI/mfparker/Descargas/Symfony/vendor/incenteev/composer-parameter-handler/Incenteev/ParameterHandler/Tests/ProcessorTest.php
  ----------------*
2 ADVERTENCIA [line: 129] Se ha detectado el uso de la función 'copy', la cual puede causar una inclusión de archivos.
3 ADVERTENCIA [line: 132] Se ha detectado el uso de la función 'copy', la cual puede causar una inclusión de archivos.
4 ->  ADVERTENCIAS = 2
5 -> INFORMACIONES = 0
6 *----------------| analizado | ----------------*
7
8 *----------------| /home/local/UCI/mfparker/Descargas/Symfony/vendor/incenteev/composer-parameter-handler/Incenteev/ParameterHandler/Tests/ProcessorTest.php
  ----------------*
9 ->  ADVERTENCIAS = 0
10 -> INFORMACIONES = 0
11 *----------------| analizado | ----------------*
12
```

Figura 14: Reporte de vulnerabilidades.

La figura 16 representa un fragmento al fichero ProcessorTest.php del paquete de Symfony, al cual se le detectó la función vulnerable descrita anteriormente y en las líneas de código pertenecientes.

```
128
129         $fs->copy($dataDir.'/dist.yml', $workingDir.'/'. $testCase['dist-file']);
130
131         if ($exists = file_exists($dataDir.'/existing.yml')) {
132             $fs->copy($dataDir.'/existing.yml', $workingDir.'/'.$testCase['config']['file']);
133         }
```

Figura 15: Ejemplo de la función vulnerable en el código de Symfony.

Como se muestra en la siguiente gráfica, se efectuaron auditorías a códigos fuente de diferentes lenguajes de programación, evidenciando la integración del Módulo de PHP a la herramienta ACF y el alto nivel de detección de vulnerabilidades.

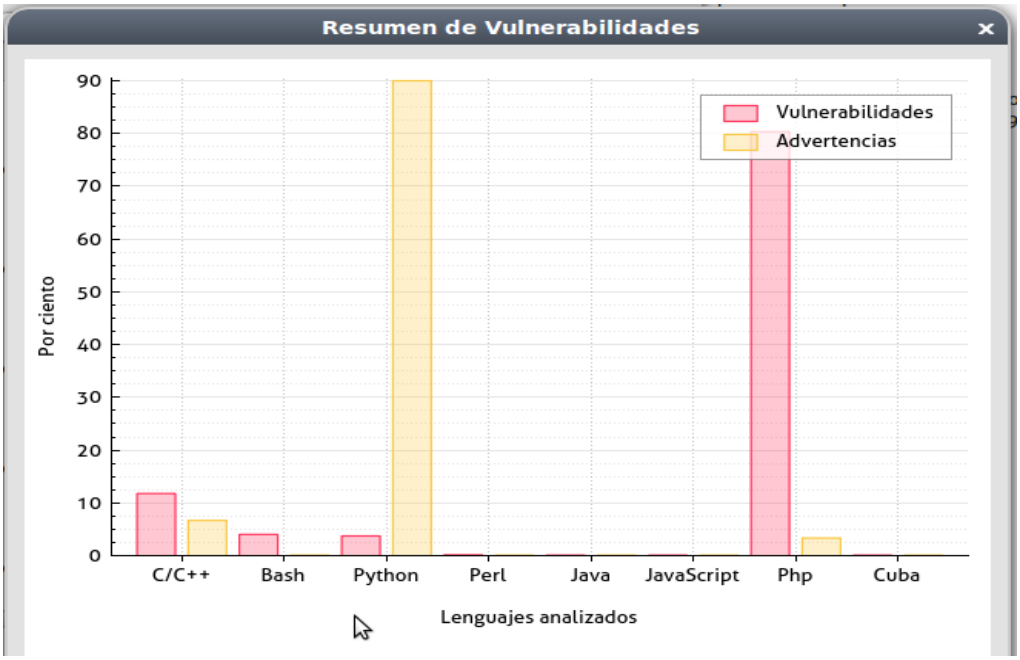

Figura 16: Integración del módulo de PHP con ACF

Como método de validación se realizaron las pruebas de aceptación, donde el cliente comprobó el correcto funcionamiento del sistema y emitió un aval de conformidad.

Conclusiones parciales

En este capítulo se definió los estándares de codificación utilizados para la implementación del sistema, lo que permitió mantener uniformidad y claridad en las líneas de código. Se realizaron pruebas a nivel de sistema, utilizando el método caja negra y como técnica asociada a ésta la partición de equivalencia, arrojando no conformidades mediante el uso de los diseños de casos de pruebas y permitiendo comprobar el funcionamiento del módulo. Mediante las pruebas de aceptación se emitió un aval de conformidad, lo que contribuyó a evidenciar la satisfacción del cliente con el producto.

Conclusiones generales

Una vez culminado el Módulo de PHP para ACF y cumpliendo con los objetivos específicos trazados en la investigación, se puede arribar a las siguientes conclusiones:

- El estudio de las herramientas dedicadas a la auditoría de código fuente de PHP permitió determinar funciones vulnerables para incorporar al módulo.
- El diseño del Módulo de PHP contribuyó a una mejor compresión del funcionamiento del sistema.
- El uso del Módulo de PHP mediante ACF permitió aumentar la detección de vulnerabilidades del repositorio de Nova.
- La realización de las pruebas de *software* permitieron comprobar el adecuado funcionamiento del sistema y la satisfacción del cliente, obteniendo un módulo capaz de detectar funciones vulnerables en las aplicaciones implementadas en PHP.

Recomendaciones

Para seguir con el perfeccionamiento del sistema, se recomienda:

- Realizar otras búsquedas de funciones vulnerables que afecten al código fuente de PHP, debido a la constante evolución de las brechas de seguridad en los sistemas informáticos.
- Aumentar el reconocimiento de caracteres en el autómata para la detección de funciones.

Referencias bibliográficas

1. PÉREZ, Damián. Los diferentes lenguajes de programación para la web. Maestros del Web [En línea]. 2 Noviembre 2007. [Accedido 5 Diciembre 2014]. Disponible en: http://www.maestrosdelweb.com/los-diferentes-lenguajes-de-programacion-para-la-web/

2. BRUNO, Luis Gabriel. Aplicaciones Informáticas. PHP. [En línea]. 30 Octubre 2014. [Accedido 4 Noviembre 2014]. Disponible en: https://sextoaplicacionesinformaticas.wordpress.com/2014/10/30/php/

3. Grandes webs que utilizan Php | TuFunción. [En línea]. [Accedido 4 Noviembre 2014]. Disponible en: http://xombit.com/2014/01/7-lenguajes-de-programacion-deberias-aprender.

4. PIERRA, Allan. Conceptualización y Reestructuración Estratégica de la Distribución Cubana de GNU/Linux “Nova”. [En línea]. Tesis de maestría. UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS, 2013. [Accedido 4 Diciembre 2014]. Disponible en: http://repositorio_institucional.uci.cu/jspui/handle/ident/800442

5. Auditoría de Código Fuente. [En línea]. 2014. [Accedido 2 Febrero 2015]. Disponible en: http://www.websec.mx/auditoria-de-codigo-fuente

6. RÍOS, Julio. Seguridad Informática. Monografías [En línea]. [Accedido 12 Febrero 2015]. Disponible en: http://www.monografias.com/trabajos82/la-seguridad-informatica/la-seguridad-informatica.shtml#ixzz3RFLPW2jY

7. HUERTA V, Antonio. Seguridad en Unix y redes. 2002. [En línea]. [Accedido 29 Mayo 2015]. Disponible en: https://www.rediris.es/cert/doc/unixsec/unixsec.pdf

8. WAISEN, Javier. PÉREZ, Francisco Javier. Web Vulnerable DVWA [En línea]. Máster en Administración, Comunicaciones y Seguridad Informática. Universidad de Almería, 2011. [Accedido

11 Febrero 2015]. Disponible en: www.adminso.es/recursos/Proyectos/PFM/2011_12/PFM_DVWA.pdf

9. PROYECTO ABIERTO DE SEGURIDAD EN APLICACIONES WEB (OWASP). OWASP Top 10 - 2010 Los diez riesgos más importantes en Aplicaciones Web. 2010. P. 22.

10. LEITCH, John. PHP Vulnerability Hunter. CodePlex [En línea]. 2 Junio 2013. [Accedido 17 Noviembre 2014]. Disponible en: http://phpvulnhunter.codeplex.com/Wikipage?ProjectName=phpvulnhunter

11. PAZ, Álvaro. Detectar y corregir automáticamente vulnerabilidades de validación de entrada en aplicaciones Web. [En línea]. 5 Noviembre 2014. [Accedido 18 Noviembre 2014]. Disponible en: http://tecnicaquilmes.fullblog.com.ar/detectar-y-corregir-automaticamente-vulnerabilidades-de-validacion-de.html

12. RÚA, Juan Alberto. Auditando la seguridad de nuestras aplicaciones PHP con WAP. [En línea]. 4 Febrero 2015. [Accedido 5 Marzo 2015]. Disponible en: http://juanchorua.blogspot.com/2015/02/auditando-la-seguridad-de-nuestras.html

13. MOTOS, Vicente. RIPS: un analizador de código estático de PHP : hackplayers. [En línea]. 3 Julio 2012. [Accedido 18 Noviembre 2014]. Disponible en: http://www.hackplayers.com/2012/07/rips-un-analizador-de-codigo-estatico.html

14. PHP PHP : CVE security vulnerabilities, versions and detailed reports. CVE Details. [En línea]. [Accedido 28 Febrero 2015]. Disponible en: http://www.cvedetails.com/product/128/PHP-PHP.html?vendor_id=74

15. IDE, Andy. PHP just grows & grows | Netcraft. Netcraft. [En línea]. 31 Enero 2013. Disponible en: http://news.netcraft.com/archives/2013/01/31/php-just-grows-grows.html

16. FRANCO, David A., PEREA, Jorge L. and TOVAR, Luis C. Herramienta para la Detección de Vulnerabilidades basada en la Identificación de Servicios. Información tecnológica. Febrero 2013. Vol. 24, no. 5, p. 13–22. DOI 10.4067/S0718-07642013000500003.

17. CVE Details. Vulnerability Details: CVE-2015-0232. [En línea]. 17 Abril 2015. [Accedido 26 Mayo 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2015-0232/

18. CVE Details. Vulnerability Details: CVE- 2014-3670. [En línea]. 13 Mayo 2015. [Accedido 26 Mayo 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2014-3670/

19. CVE Details. Vulnerability Details: CVE- 2006-1991. [En línea]. 13 Junio 2011. [Accedido 26 Mayo 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2006-1991/

20. CVE Details. Vulnerability Details: CVE- 2014-3487. [En línea]. 13 Abril 2015. [Accedido 1 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2014-3487/

21. CVE Details. Vulnerability Details: CVE-2014-3480. [En línea]. 13 Abril 2015. [Accedido 1 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2014-3480/

22. CVE Details. Vulnerability Details: CVE-2014-3479. [En línea]. 13 Abril 2015. [Accedido 1 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2014-3479/

23. CVE Details. Vulnerability Details: CVE-2014-9652. [En línea]. 1 Abril 2015. [Accedido 1 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2014-9652/

24. CVE Details. Vulnerability Details: CVE- 2014-0207. [En línea]. 13 Abril 2015. [Accedido 1 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2014-0207/

25. CVE Details. Vulnerability Details: CVE- 2015-1352. [En línea]. 21 Mayo 2015. [Accedido 1 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2015-1352/

26. CVE Details. Vulnerability Details: CVE- 2015-2301. [En línea]. 21 Mayo 2015. [Accedido 1 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2015-2301/

27. CVE Details. Vulnerability Details: CVE- 2015-2331. [En línea]. 13 Abril 2015. [Accedido 1 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2015-2331/

28. CVE Details. Vulnerability Details: CVE-2014-3668. [En línea]. 13 Mayo 2015. [Accedido 1 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2014-3668/

29. CVE Details. Vulnerability Details: CVE-2002-1396. [En línea]. 9 Mayo 2008. [Accedido 1 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2002-1396/

30. CVE Details. Vulnerability Details: CVE- 2006-7205. [En línea]. 9 Mayo 2008. [Accedido 2 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2006-7205/

31. CVE Details. Vulnerability Details: CVE-2015-0232. [En línea]. 23 Julio 2013. [Accedido 2 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2012-2335/

32. CVE Details. Vulnerability Details: CVE-2011-3268. [En línea]. 2 Febrero 2012. [Accedido 2 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2011-3268/

33. CVE Details. Vulnerability Details: CVE- 2013-7226. [En línea]. 13 Marzo 2014. [Accedido 2 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2013-7226/

34. CVE Details. Vulnerability Details: CVE- 2014-3670. [En línea]. 13 Mayo 2015. [Accedido 2 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2014-3670/

35. CVE Details. Vulnerability Details: CVE- 2013-6420. [En línea]. 18 Mayo 2015. [Accedido 2 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2013-6420/

36. CVE Details. Vulnerability Details: CVE- 2013-4248. [En línea]. 12 Febrero 2014. [Accedido 2 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2013-4248/

37. CVE Details. Vulnerability Details: CVE-2007-1401. [En línea]. 9 Mayo 2008. [Accedido 2 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2007-1401/

38. CVE Details. Vulnerability Details: CVE- 2012-4388. [En línea]. 11 Noviembre 2013. [Accedido 2 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2012-4388/

39. Tutoriales Au.cl. Vulnerabilidad de la función mail () de PHP. [En línea]. 2 Septiembre 2012. [Accedido 2 Junio 2015]. Disponible en: http://archive-cl.com/cl/a/au.cl/2013-08-14_2620241_15/Vulnerabilidad_de_la_funci%C3%B3n_mail_de_PHP_Tutoriales_Au_cl/

40. Web Application Protection. WAP Manual. [En línea]. [Accedido 2 Junio 2015]. Disponible en: http://awap.sourceforge.net/support.html

41. CVE Details. Vulnerability Details: CVE- 2014-8142. [En línea]. 17 Marzo 2015. [Accedido 2 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2014-8142/

42. CVE Details. Vulnerability Details: CVE- 2014-3669. [En línea]. 17 Abril 2015. [Accedido 3 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2014-3669/

43. CVE Details. Vulnerability Details: CVE-2012-2376 (1 public exploit). [En línea]. 16 Agosto 2012. [Accedido 3 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2012-2376/

44. CVE Details. Vulnerability Details: CVE- 2007-0448. [En línea]. 10 Noviembre 2008. [Accedido 3 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2007-0448/

45. CVE Details. Vulnerability Details: CVE- 2014-5368. [En línea]. 27 Agosto 2014. [Accedido 3 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2014-5368/

46. CN-CERT. Vulnerability Bulletins. Múltiples vulnerabilidades en PHP. [En línea]. [Accedido 26 Mayo 2015]. Disponible en: https://www.ccn-cert.cni.es/index.php?option=com_vulnerabilidades&task=view&id=2310&Itemid=96&lang=gl

47. CVE Details. Vulnerability Details: CVE- 2006-5178. [En línea]. 15 Septiembre 2010. [Accedido 3 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2006-5178/

48. EL hacker.net. Remote File Inclusion. [En línea]. [Accedido 3 Junio 2015]. Disponible en: http://wiki.elhacker.net/bugs-y-exploits/nivel-web/rfi

49. CVE Details. Vulnerability Details: CVE- 2015-2348. [En línea]. 13 Abril 2015. [Accedido 3 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2015-2348/

50. CVE Details. Vulnerability Details: CVE- 2010-3870. [En línea]. 23 Marzo 2011. [Accedido 3 Junio 2015]. Disponible en: http://www.cvedetails.com/cve/CVE-2010-3870/

51. MENDOZA, Miguel Angel. WELIVESECURITY. Vulnerabilidades: ¿qué es CVSS y cómo utilizarlo. [En línea]. 4 Agosto 2014. [Accedido 6 Junio 2015]. Disponible en: http://www.welivesecurity.com/la-es/2014/08/04/vulnerabilidades-que-es-cvss-como-utilizarlo/

52. Manual de usuario Auditoría de código fuente (ACF) v1.0. Centro de Software Libre y Código Abierto (CESOL), Universidad de las Ciencias Informáticas (UCI): La Habana, 2014.

53. HERNÁNDEZ, Carmelo José. Metodología open up ágil y tradicional. [En línea]. Universidad de Córdoba. 14:24:47 UTC. [Accedido 14 Marzo 2015]. Disponible en: http://es.slideshare.net/carmeloh2/metodologa-open-up-39321348

54. GIMSON, Loraine. Metodologías ágiles y desarrollo basado en conocimiento. UNIVERSIDAD NACIONAL DE LA PLATA. FACULTAD DE INFORMÁTICA. Junio 2012.

55. STROUSTRUP, Bjarne. A Tour of C++. The C++ Programming Language. [En línea]. [Accedido 4 Junio 2015]. Disponible en: http://www.stroustrup.com/3rd_tour.pdf

56. COTTY, Enmanuel. Mi lenguaje de programación de preferencia: C++. [En línea]. Universidad Interamericana de Puerto Rico. 15 Abril 2012. [Accedido 14 Marzo 2015]. Disponible en:http://es.slideshare.net/bastard1/mi-lenguaje-de-programacin-de-preferencia-c

57. Exes. XML ¿QUÉ ES? [En línea]. Universidad a Distancia de Madrid, 2014. [Accedido 14 Junio 2015]. Disponible en: http://www.mundolinux.info/que-es-xml.htm

58. Definición de XML - Qué es, Significado y Concepto. [En línea]. 2015. [Accedido 14 Junio 2015]. Disponible en: http://definicion.de/xml/#ixzz3dF7Ts68x

59. GÓNZALEZ, Marvin. IDE- Eclipse. [En línea]. 1 Agosto 2013. [Accedido 14 Marzo 2015]. Disponible en: http://es.slideshare.net/MagaLasic/presentacion-eclipse-grupo-6

60. GENBETA:dev. Eclipse IDE. [En línea]. 10 Enero 2014. [Accedido 20 Marzo 2015]. Disponible en: http://www.genvetadev.com

61. SIERRA, Daniel. Herramientas automatizadas. Exposición herramienta CASE. [En línea]. Universidad Interamericana de Puerto Rico. 16 Noviembre 2007. [Accedido 14 Marzo 2015]. Available from: http://es.slideshare.net/vanquishdarkenigma/visual-paradigm-for-uml

62. Doxygen. Marco de Desarrollo de la Junta de Andalucía. [Accedido 14 Marzo 2015]. Disponible en: http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/110

63. SALDAÑA, Gabriel. Introducción a Git: Un Sistema de control de versiones...bien hecho. [Accedido 11 Mayo 2015]. Disponible en: http://www.gabrielsaldana.org/platica_git.pdf

64. RÍOS, Santiago. HINOJOSA, Cecilia. DELGADO, Ramiro. Aplicación de la Metodología OpenUP en el desarrollo de Sistemas de Difusión de Gestión del Conocimiento de la ESPE. [En línea]. Escuela Politécnica del Ejército, Ecuador. 2013. [Accedido 12 Mayo 2015]. Disponible en: http://repositorio.espe.edu.ec/bitstream/21000/6316/1/AC-SISTEMAS-ESPE-047042.pdf

65. Elementos de UML. Diagrama de casos de uso. [En línea]. [Accedido 22 Marzo 2015]. Disponible en: https://docs.kde.org/stable/es/kdesdk/umbrello/uml-elements.html

66. GOÑI, Alfredo. Reutilización del Software. Patrones de Diseño. [En línea]. [Accedido 11 Mayo 2015]. Disponible en: https://www.siul02.si.ehu.es/~alfredo/iso/06Patrones.pdf. p- 82

67. VISCONTI, Marcello. ATUDILLO, Hernán. Fundamentos de Ingeniería de Software. Departamento de Informática Universidad Técnica Federico Santa María.

68. ALTOVA. Diagramas de paquetes UML. [En línea]. [Accedido 26 Marzo 2015]. Disponible en: http://www.altova.com/es/umodel/uml-package-diagrams.html

69. Programación Orientada a Objetos (POO). Tag Archives: Diagrama de clases. [En línea]. Universidad Nacional Experimental del Táchira – Departamento de Informática – Computación II, 2 Noviembre 2013. [Accedido 22 Marzo 2015]. Disponible: https://compu2poo.wordpress.com/tag/diagrama-de-clases/

70. GONZÁLEZ, Javier MCs. UML Y el proceso unificado de desarrollo trabajando con objetos. [En línea]. Departamento de Tecnologías de Información ITESM, campus Guadalajara.2009 [Accedido 22 Marzo 2015]. Disponible en: http://es.slideshare.net/javiergs/200505-modelado-de-software-con-uml

71. SARMIENTO, Johana. UML: Diagrama de Despliegue. [En línea]. 12 Abril 2013. [Accedido 26 Marzo 2015]. Disponible en : http://umldiagramadespliegue.blogspot.com/2013/04/vision-general-de-los-diagramas-de.html

72. Elementos de UML. Diagrama de componentes. [En línea]. [Accedido 23 Marzo 2015]. Disponible en: https://docs.kde.org/stable/es/kdesdk/umbrello/uml-elements.html#state-diagram

73. Estándares de codificación de sistemas. Dirección General de Gobierno Digital. Gobierno de Chubut. [En línea]. [Accedido 11 Mayo 2015]. Disponible en: http://www.chubut.gov.ar/informatica/docs/EstandaresCodificacion.pdf

74. BLANCO, Carlos. Construcción y pruebas de software. Ingeniería de Software II. Universidad de Cantabria, 2015.

75. JURISTO, Natalia. MORENO, Ana M. Técnicas de evaluación de software. Versión 12.0. [En línea]. 17 de octubre de 2006. [Accedido 22 Marzo 2015]. Disponible en: http://www.grise.upm.es/sites/extras/12/pdf/Documentacion_Evaluacion_7.pdf

Bibliografía

1. SERAFÍN, Adrián. “Herramienta para la detección de vulnerabilidades en el código fuente del repositorio de GNU/Linux Nova“. [En línea]. Universidad de las Ciencias Informáticas, 2012. Available from: http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_05171_12

2. SOMMERVILLE, Ian. Cap 6: Requisitos de Software. [En línea]. 8va Edición. Disponible en: http://eva.uci.cu/mod/resource/view.php?id=8501&subdir=/Ediciones_del_Sommerville

3. SOMMERVILLE, Ian. Requerimientos del Software. Ingeniería de Software. Séptima Edición. Madrid, 2005. P 216.

4. PRESSMAN, Roger S. Ingeniería de Software: un enfoque práctico. 6ta Edición. Editorial McGraw-Hill. Nueva York, E.U.A, 2007. Capítulo 13.

5. PRESSMAN, Roger S. Ingeniería de Software: Técnicas de Prueba del Software. Parte 1. Editorial McGraw-Hill. Nueva York, E.U.A, 2007. Capítulo 14. P 418– 440.

6. PRESSMAN, Roger S. Ingeniería de Software: Técnicas de Prueba del Software. Parte 2. Editorial McGraw-Hill. Nueva York, E.U.A, 2007. Capítulo 14. P 441– 461.

7. LARMAN, Craig. UML y Patrones. Introducción al análisis y diseño orientado a objetos.2003. P-507.

8. OpenUP-EPF Wiki -Eclipse. Introduction to OpenUP. [En línea]. [Accedido 12 Mayo 2015]. Disponible en: http://www.epf.eclipse.org/wikis/openup/

9. SAN, Jose A. Scribd. Arquitectura Basada en Componentes. [En línea]. 28 Abril 2009. [Accedido 22 Marzo 2015]. Disponible en: http://www.es.scribd.com/doc/14704374/Arquitectura-Basada-en Componentes#scribd

Glosarios de términos

CASE

Las herramientas CASE (*computer aided software engineering*, ingeniería de *software* asistida por computadora) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de *software*, reduciendo costo y tiempo de las mismas.

Código Fuente

El código fuente de un programa informático (o *software*) es un conjunto de líneas de texto que son instrucciones escritas en algún lenguaje de programación de computadoras, hechas para ser leídas y transformadas por alguna herramienta de *software* (compilador, intérprete, ensamblador) en lenguaje de máquina o instrucciones ejecutables en la máquina.

CVE

El código CVE (vulnerabilidades y amenazas comunes) es un identificador que se asigna a cada vulnerabilidad que se conoce públicamente con el fin de que pueda ser identificada de forma unívoca. Este código fue creado por la corporación MITRE y permite que los usuarios puedan conocer de forma objetiva las vulnerabilidades de un sistema computacional. Los identificadores CVE se presentan en el formato CVE-AÑO-NÚMERO y están acompañados de una breve descripción de la vulnerabilidad o amenaza y un grupo de referencias pertinentes.

Web

Web es un vocablo inglés que significa “red”, “telaraña” o “malla”. El concepto se utiliza en el ámbito tecnológico para nombrar a una red informática y, en general, a Internet (en este caso, suele escribirse como Web, con la W mayúscula).

Anexos

Anexo 1

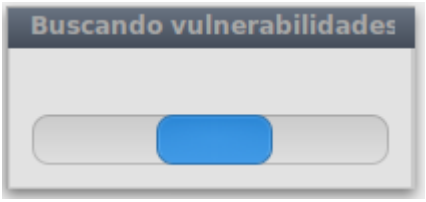

Figura 17: Ejemplo del EC 1.1 de los diseños de casos de pruebas.

Anexo 2

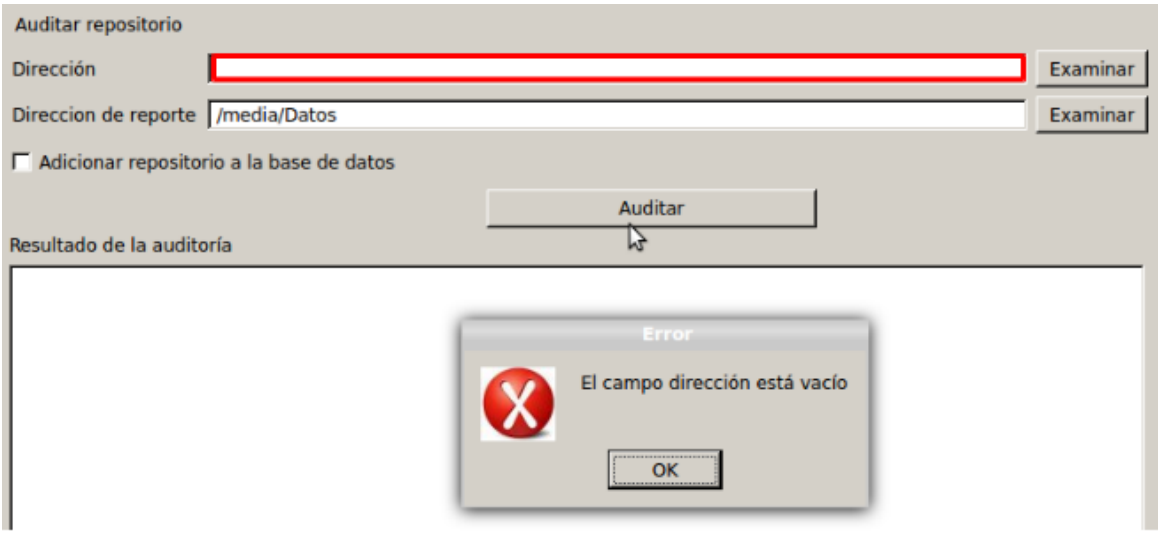

Figura 18: Ejemplo del EC 1.2 de los diseños de casos de pruebas.

Anexo 3

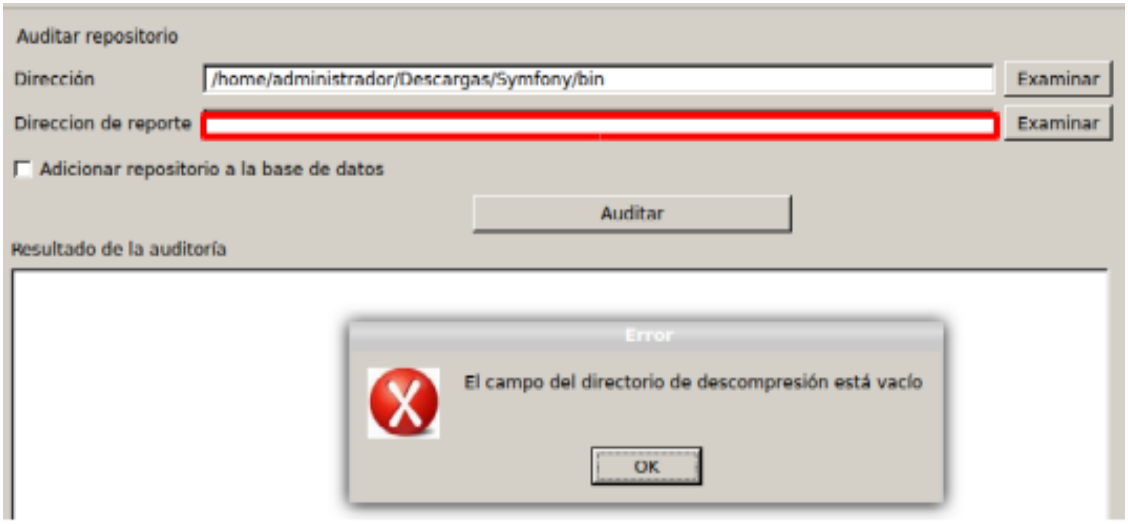

Figura 19: Ejemplo del EC 1.3 de los diseños de casos de pruebas.

Anexo 4

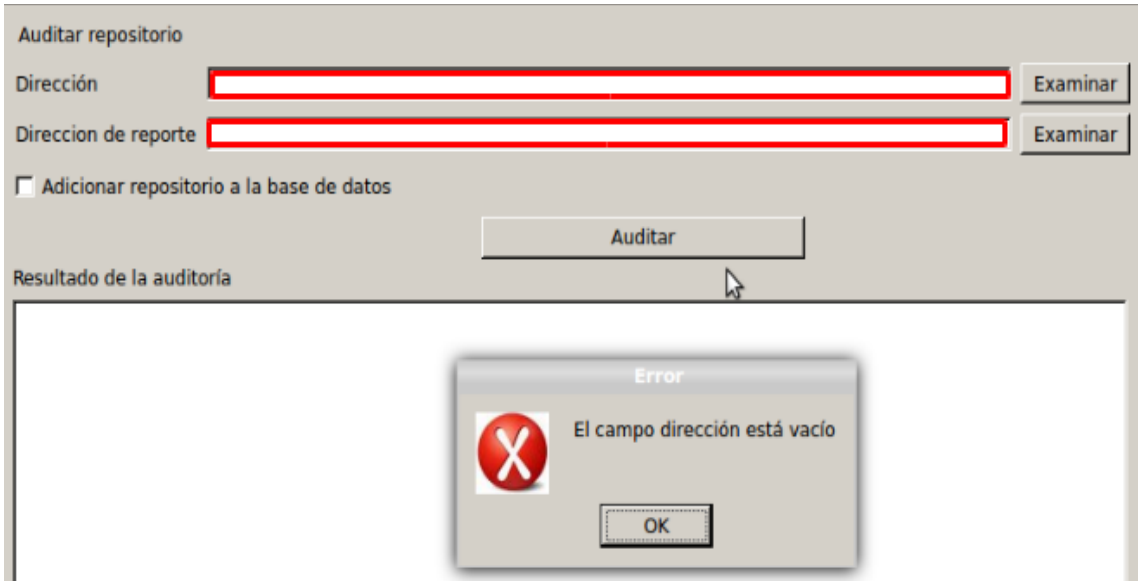

Figura 20: Ejemplo del EC 1.4 de los diseños de casos de pruebas.

Anexo 5

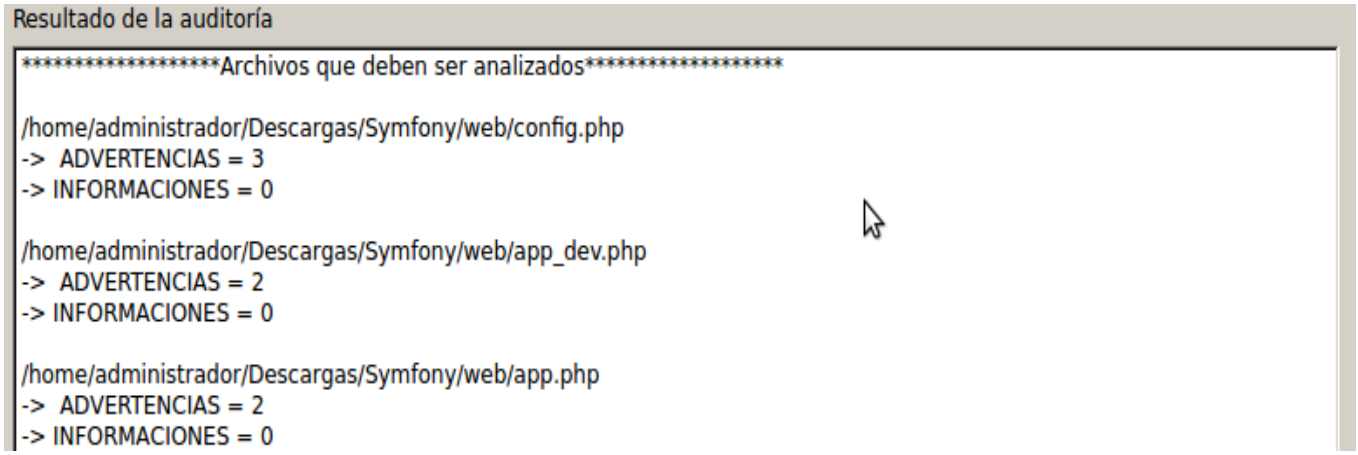

Figura 21: Ejemplo del EC 1.5 de los diseños de casos de pruebas.

Anexo 6

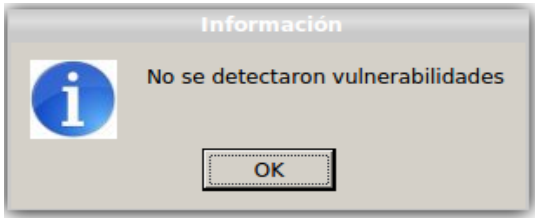

Figura 22: Ejemplo del EC 1.6 de los diseños de casos de pruebas.

Anexo 7

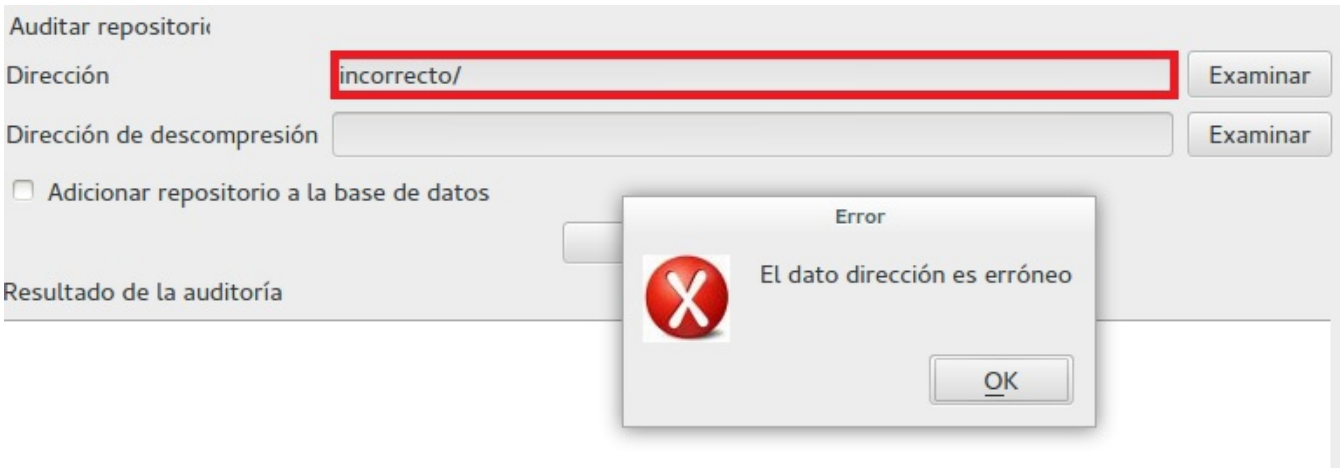

Figura 23: Ejemplo del EC 1.7 de los diseños de casos de pruebas.

Anexo 8

Vulnerabilidad	Confidencialidad		Integridad		Disponibilidad		Resultado de la ecuación de Impacto.
	Tipo	Valor	Tipo	Valor	Tipo	Valor	
Denegación de servicio.	Completa	0.66	Completa	0.66	Completa	0.66	10,00084536
Ejecución de código.	Completa	0.66	Completa	0.66	Completa	0.66	10,00084536
Desbordamiento de buffer.	Completa	0.66	Completa	0.66	Completa	0.66	10,00084536
Derivación.	Parcial	0.275	Parcial	0.275	Parcial	0.275	6,442976719
Obtención de información.	Completa	0.66	Completa	0.66	Completa	0.66	10,00084536
Corrupción de memoria.	Parcial	0.275	Parcial	0.275	Parcial	0.275	6,442976719
XSS.	Parcial	0.275	Parcial	0.275	Parcial	0.275	6,442976719
Directorio travesal.	Parcial	0.275	Ninguna	0,0	Ninguna	0,0	2,86275
Inyección SQL.	Parcial	0.275	Parcial	0.275	Parcial	0.275	6,442976719
Obtención de privilegios.	Completa	0.66	Completa	0.66	Completa	0.66	10,00084536
División de respuesta HTTP.	Completa	0.66	Completa	0.66	Completa	0.66	10,00084536
Inclusión de archivos.	Parcial	0.275	Parcial	0.275	Parcial	0.275	6,442976719

Tabla 7: Métrica de Impacto

Anexo 9

Vulnerabilidad	Vector de acceso		Complejidad de acceso		Autenticación		Resultado de la ecuación de Explotabilidad.
	Tipo	Valor	Tipo	Valor	Tipo	Valor	
Denegación de servicio.	Remoto	1	Alta	0.35	No requerida	0.704	4,928
Ejecución de código.	Remoto	1	Alta	0.35	No requerida	0.704	4,928
Desbordamiento de buffer.	Remoto	1	Alta	0.35	No requerida	0.704	4,928
Derivación.	Remoto	1	Baja	0.35	No requerida	0.704	4,928
Obtención de información.	Remoto	1	Alta	0.35	No requerida	0.704	4,928
Corrupción de memoria.	Remoto	1	Alta	0.35	No requerida	0.704	4,928
XSS.	Remoto	1	Baja	0.71	No requerida	0.704	10,011
Directorio travesal.	Remoto	1	Baja	0.71	No requerida	0.704	10,011
Inyección SQL.	Remoto	1	Baja	0.71	No requerida	0.704	10,011
Obtención de privilegios.	Local	0.395	Alta	0.35	No requerida	0.704	1,949325
División de respuesta HTTP.	Remoto	1	Baja	0.71	No requerida	0.704	10,011
Inclusión de archivos.	Remoto	1	Alta	0.35	No requerida	0.704	4,928

Tabla 8: Métrica de Explotabilidad

Anexo 10

Vulnerabilidad	Impacto	Explotabilidad	Resultado de la Métrica Base	Clasificación Advertencia >= 6 5<Alerta<6 3<Información<5
Denegación de servicio.	4,928	10,00084536	6,417594457	Advertencia
Ejecución de código.	4,928	10,00084536	6,417594457	Advertencia
Desbordamiento de buffer.	4,928	10,00084536	6,417594457	Advertencia
Derivación.	4,928	6,442976719	4,743973049	Información
Obtención de información.	4,928	10,00084536	6,417594457	Advertencia
Corrupción de memoria.	4,928	6,442976719	4,743973049	Información
XSS.	10,011	6,442976719	8,330537849	Advertencia
Directorio travesal.	10,011	2,86275	6,6463992	Advertencia
Inyección SQL.	10,011	6,442976719	8,330537849	Advertencia
Obtención de privilegios.	1,949325	10,00084536	4,315841377	Información
División de respuesta HTTP.	10,011	10,00084536	10,004159257	Advertencia
Inclusión de archivos.	4,928	6,442976719	4,727837086	Información

Tabla 9: Métrica Base

Anexo 11

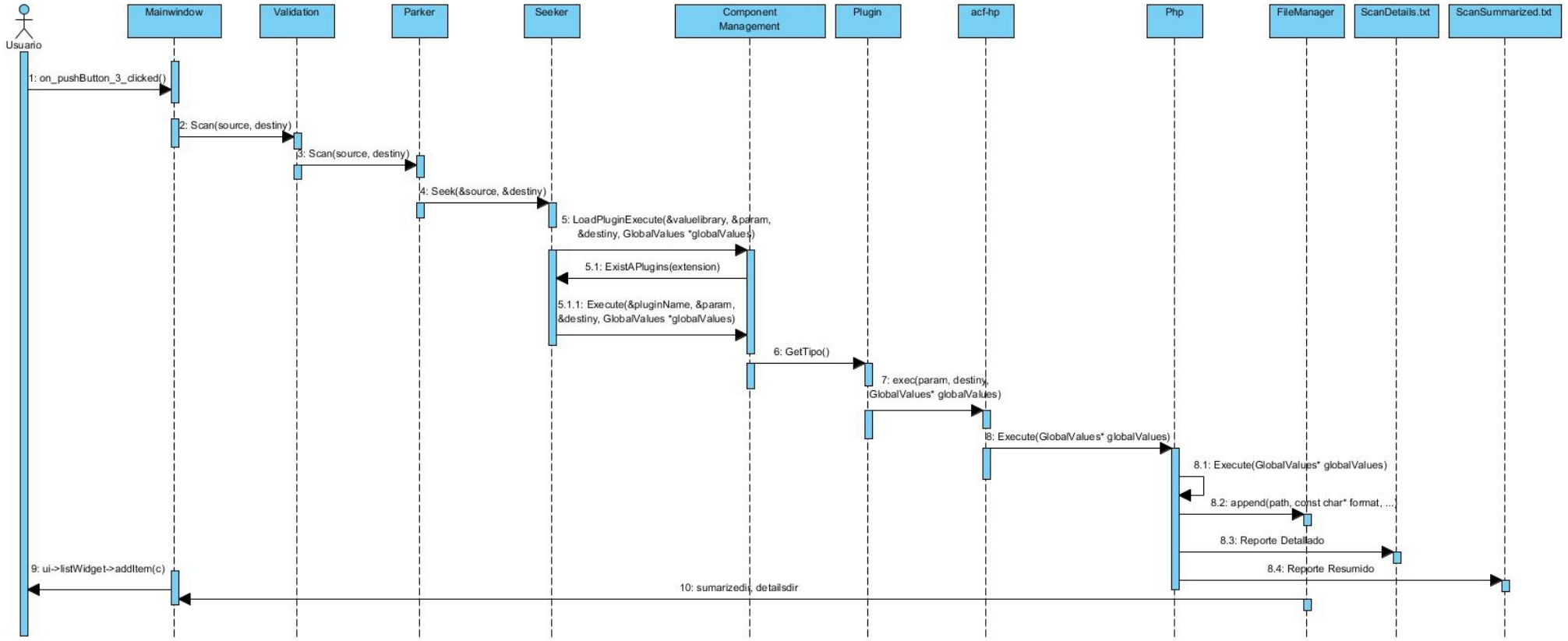

Figura 24: Diagrama de secuencia.