



UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
FACULTAD 2

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

Sistema de Análisis Estático de Vulnerabilidades. Módulo PHP.

Autores:

Adrián Hernández Yeja.

Yosbany Tejas de la Cruz.

Tutor:

Ing. Enrique Félix Agudo Veliz.

Co-Tutora:

Ing. Imirys Rovira Prieto.

Ciudad de La Habana, junio del 2010

“Año 52 de la Revolución”

Declaración de Autoría

Declaramos que Adrián Hernández Yeja y Yosbany Tejas de la Cruz somos los únicos autores de este trabajo y autorizamos a la Facultad 2 de la Universidad de las Ciencias Informáticas (UCI) a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ____ días del mes de _____ del 2010.

Firma del Autor

Adrián Hernández Yeja

Firma del Autor

Yosbany Tejas de la Cruz

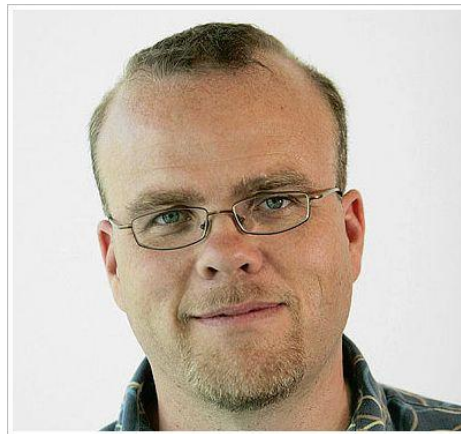
Firma del Tutor

Ing. Enrique Félix Agudo Veliz

Firma del Co-Tutor

Ing. Imirys Rovira Prieto

Pensamiento



“Security is not easy. People have to understand their systems well to know where security issues are likely to appear, and they have to remember to actually check. Like a small hole in a balloon, one missed security check will burst their application.”

Rasmus Lerdorf

Agradecimientos

De Adrián:

Quiero agradecer a todas las personas que han hecho posible la realización de esta tesis por su apoyo espiritual y ayuda en los momentos difíciles. Ellos son:

- Mi familia, en especial, mi mamá, mi papá y mi hermana, los que me han brindado fuerzas para seguir adelante en todo momento.*
- El tutor Ing. Enrique Félix Agudo, por su preocupación y ayuda.*
- Las comunidades de Foros del Web y KumbiaPHP, las que han permitido tener una referencia constante en el conocimiento y han demostrado la fuerza de la socialización del conocimiento.*
- Mis profesores, por la enseñanza y conocimiento que me han brindado para toda la vida.*
- Nuestro líder de proyecto Ing. Rogfel Thompson, por confiar en que podíamos lograr un producto como el que se presenta.*
- La Universidad de Ciencias Informáticas, la cual ha sido el centro donde pude hacer realidad los sueños de ser informático.*
- La Revolución cubana, madre de todos los cubanos y cubanas, que ha hecho posible que un estudiante humilde haya cumplido sus metas.*
- A Fidel Castro, guía y padre de la Revolución, por sus enseñanzas y ejemplo.*

¡A todos muchas gracias!

De Yosbany:

Quisiera agradecer esta tesis a todos aquellos que hicieron posible la realización de la misma, en especial:

- A los seres que más amo en este mundo: mis padres y mi hermana, por ser la fuente de mi inspiración y motivación para superarme cada día más y así poder luchar para que la vida me entregue un futuro mejor.*
- A la Revolución cubana y a nuestro Comandante en Jefe Fidel Castro por confiar en los jóvenes de hoy, y prepararnos día a día para la continuidad de la Revolución.*
- A la Universidad de las Ciencias Informáticas, por ser la universidad que hizo mis sueños realidad como Ingeniero en las Ciencias Informáticas.*
- A la FEU y a su secretariado por confiar en mí y prepararme como dirigente, en especial al presidente Yidian Yosbel Castellano.*
- Gracias a todos mis amigos que estuvieron conmigo y compartimos tantas aventuras, experiencias, desveladas y triunfos. Gracias a Manuel Cabrera Murillo por apoyarme en cada una de mis decisiones y por hacer que mi estancia en la Universidad fuera súper divertida.*
- Quisiera hacer un especial agradecimiento a mi prima Yusmila de la Cruz por el cariño y apoyo brindado durante los 5 años de la carrera, y en general a todos mis familiares.*
- Gracias a mis amigos del aula y del laboratorio, por hacer que cada pedazo de tiempo fuera ameno. No voy a olvidar sus consejos, enseñanzas y ayuda durante el lapso de mi tesis.*
- A nuestro Líder de Proyecto, al tutor por su ayuda, a la comunidad de desarrollo de KumbiaPHP y a todos mis profesores por hacer de mí un profesional. *¡A todos muchas Gracias!**

Dedicatoria

De Adrián:

*A mis padres, mi hermana, mi sobrina, la Revolución
y a los que confían que un futuro mejor es posible.*

De Yosbany:

*A mis padres, mi hermana, a la Revolución cubana
y a todos los que confiaron en mí.*

Resumen

La Universidad de las Ciencias Informáticas tiene como uno de sus objetivos principales la excelencia en la calidad de los productos que oferta. Uno de los requisitos esenciales para el cumplimiento de esta meta lo representa la seguridad de las aplicaciones, sobre todo las del tipo *Web*, mediante oportunas técnicas de validación como es el análisis dinámico y estático, este último importante en fases tempranas del ciclo de desarrollo del *software* por los beneficios que ofrece en la detección de errores de desarrollo. Es por ello que, un lenguaje tan utilizado en la universidad como lo representa *PHP*, por las facilidades y flexibilidad que ofrece a los desarrolladores, requiere del análisis estático de código como una alternativa en la búsqueda de la calidad del producto. La tesis que se presenta, **Sistema de Análisis Estático de Vulnerabilidades, Módulo *PHP***, permite ofrecer una herramienta de validación de código para los desarrolladores y la comunidad universitaria en sentido general.

En la elaboración del *software* se aprovecha las ventajas de los ambientes basados en la *Web*, con el motor de búsqueda de vulnerabilidades del analizador estático *Pixy*, este último con actualizaciones a versiones recientes de *PHP*, principalmente en temas de seguridad, lo que permite ofrecer un producto fácil de usar, rápido, sencillo y con un bajo nivel de detección de falsos positivos.

El sistema brinda la posibilidad de mostrar estadísticas de los análisis realizados, filtrado de reportes generados, así como configuración del analizador de código estático, lo que permite flexibilidad para el usuario.

Una vez desarrollada la solución propuesta se dispone de una herramienta de detección de problemas de seguridad en aplicaciones *PHP* en la Universidad de las Ciencias Informáticas.

Palabras claves: *PHP*, calidad, seguridad.

Índice

INTRODUCCIÓN	1
CAPÍTULO I	6
1.1. Seguridad Informática.....	6
1.2. Programación Segura.....	7
1.3. Seguridad en Aplicaciones <i>Web</i>	8
1.4. Seguridad en Aplicaciones <i>PHP</i>	10
1.5. Tipos de análisis a aplicaciones <i>Web</i>	19
1.6. Tipos de análisis de código estático.....	21
1.7. Características deseables de los analizadores de código estático.....	21
1.8. Analizadores de código estático para <i>PHP</i>	22
1.8.1. Analizadores propietarios.....	22
1.8.1.1. Code Secure.....	23
1.8.1.2. CodeScan.....	23
1.8.2. Analizadores de Código Libre.....	23
1.8.2.1. PHP-SAT.....	23
1.8.2.2. SWAAT (<i>Securitycompass Web Application Auditing Tool</i>).....	24
1.8.2.3. RATS (<i>Rough Auditing Tool for Security</i>).....	24
1.8.2.4. YASCA (<i>Yet Another Source Code Analyzer</i>).....	25
1.8.2.5. Pixy.....	25

1.9.	Propuesta de Desarrollo del sistema.	28
1.9.1.	Plataforma de Desarrollo Linux.	28
1.9.2.	XML (Extensible Markup Language).	29
1.9.3.	Software de Visualización Gráfica Graphviz.	29
1.9.4.	Lenguaje de Modelado <i>UML</i>	29
1.9.5.	Metodología de desarrollo <i>RUP</i>	30
1.9.6.	Herramienta CASE Visual Paradigm.	30
1.9.7.	Entorno de Desarrollo Eclipse.	31
1.9.8.	Servidor Web Apache.	31
1.9.9.	Servidor FTP VSFTPD (<i>Very Secure FTP Daemon</i>).	32
1.9.10.	Gestor de Base de Datos PostgreSQL.	32
1.9.11.	Lenguaje de Desarrollo PHP (Hypertext Preprocessor).	33
1.9.12.	Biblioteca de Javascript <i>JQuery</i>	33
1.9.13.	Framework de desarrollo para <i>PHP KumbiaPHP</i>	34
1.10.	Conclusiones.	34
CAPÍTULO II	35
2.1.	Objeto de Estudio.	35
2.1.1.	Problema y situación problemática.	35
2.1.2.	Objeto de automatización.	36
2.1.3.	Información que se maneja.	36
2.2.	Propuesta del Sistema.	36
2.3.	Modelo de Dominio.	37
2.4.	Especificación de los requisitos de Software.	39
2.4.1.	Requisitos funcionales.	39

2.4.2.	Requisitos no funcionales.....	41
2.5.	Modelo de casos de uso del sistema.....	45
2.5.1.	Definición de los actores del sistema a automatizar.....	45
2.5.2.	Diagrama de casos de uso del sistema a automatizar.....	45
2.5.3.	Descripción de los casos de uso.....	46
2.6.	Conclusiones.....	46
CAPÍTULO III	47
3.1.	Análisis.....	47
3.1.1.	Modelo de Análisis.....	47
3.1.2.	Diagramas de clases del Análisis.....	47
3.2.	Diseño del Sistema.....	51
3.2.1.	Patrones que se utilizan.....	51
3.2.2.	Funcionamiento general de una aplicación en <i>KumbiaPHP 1.0 Spirit</i>	54
3.2.3.	Arquitectura del sistema.....	56
3.2.4.	Diagramas de Interacción.....	57
3.2.5.	Diagramas de clases del Diseño.....	58
3.3.	Modelo de Datos.....	63
3.3.1.	Modelo lógico de datos (diagrama de clases persistentes).....	63
3.3.2.	Modelo físico de datos (Modelo de Datos).....	64
3.4.	Conclusiones.....	65
CAPÍTULO IV	66
4.1	Diagrama de Despliegue.....	66
4.2	Estándares de codificación.....	68
4.3	Diagrama de Componentes.....	69

4.4	Conclusiones.	72
CAPÍTULO V		73
5.1.	Modificaciones en el <i>parser</i> de <i>Pixy PHPParser</i>	73
5.2.	Generación de reportes en formato <i>XML</i>	77
5.3.	Análisis de vulnerabilidades del tipo inyección de código.	78
5.4.	Nuevas funciones de repercusión en la seguridad de aplicaciones <i>PHP</i> en <i>Pixy</i>	78
5.5.	Otras modificaciones.	79
5.6.	Recomendaciones para el correcto funcionamiento de <i>Pixy</i>	82
5.7.	Conclusiones.	83
CAPÍTULO VI		85
6.1.	Método de estimación Puntos por Casos de Uso.....	85
6.1.1.	Cálculo de Puntos de Casos de Uso sin ajustar.....	85
6.1.2.	Cálculo de Puntos de Casos de Uso ajustados.....	88
6.1.3.	Cálculo del Esfuerzo.....	93
6.1.4.	Distribución del Esfuerzo entre las diferentes actividades del módulo.	94
6.2.	Beneficios tangibles e intangibles.	94
6.3.	Análisis de costos y beneficios.	95
6.4.	Conclusiones.	96
CONCLUSIONES GENERALES		97
RECOMENDACIONES.....		99
REFERENCIAS BIBLIOGRÁFICAS.....		100
BIBLIOGRAFÍA.....		103

ANEXOS	107
DESCRIPCIÓN DE LOS CASOS DE USO DEL SISTEMA	107
DIAGRAMAS DE SECUENCIA (DS) DE CADA CASO DE USO (CU)	130
DIAGRAMA DE CLASES DEL ANALIZADOR DE CÓDIGO ESTÁTICO <i>PIXY</i>	138
REGLAS ADICIONADAS EN JFLEX Y JCUP	139
FUNCIONES AGREGADAS EN <i>PIXY</i>	151
DECLARACIONES DE DESARROLLADORES DEL PROYECTOS <i>PIXY</i> Y <i>YASCA</i>	157
CÁLCULO DE NIVEL DE RIESGO DEL ANÁLISIS DE FICHEROS.....	159
GLOSARIO DE TÉRMINOS	161

Índice de Figuras

Fig. 1 Esquema general de un modelo común de vulnerabilidades <i>Web</i>	10
Fig. 2 Modelo de Dominio	38
Fig. 3 Diagrama de Casos de Uso del Sistema	46
Fig. 4 DCA Subir Fuente	49
Fig. 5 DCA Generar Reporte	49
Fig. 6 DCA Gestionar Reportes	50
Fig. 7 DCA Mostrar Estadísticas	50
Fig. 8 Estructura común de una aplicación <i>KumbiaPHP</i>	54
Fig. 9 Funcionamiento de clases controladoras en <i>KumbiaPHP</i>	55
Fig. 10 Funcionamiento de los modelos en <i>KumbiaPHP</i>	56
Fig. 11 Arquitectura del sistema	57
Fig. 12 DCD Subir Fuente	61
Fig. 13 DCD Generar Reporte	62
Fig. 14 DCD Gestionar Reportes	62
Fig. 15 DCD Mostrar Estadísticas	63
Fig. 16 Diagrama de clases persistentes	64
Fig. 17 Modelo de Datos	65
Fig. 18 Diagrama de Despliegue	67

Fig. 19 Diagrama de Componentes, Base de Datos69

Fig. 20 Diagrama de Componentes, código fuente70

Fig. 21 Diagrama de componentes, componentes Web o código ejecutable.....71

Fig. 22 DS CU Subir Fuente. Sección Subir Fichero 130

Fig. 23 DS CU Subir Fuente. Sección Subir Código..... 131

Fig. 24 DS CU Generar Reporte 132

Fig. 25 DS CU Gestionar Reportes 132

Fig. 26 DS CU Gestionar Reportes. Sección Eliminar 133

Fig. 27 DS CU Gestionar Reportes. Sección Descargar 133

Fig. 28 DS CU Mostrar Estadísticas. Sección Estadísticas por usuario..... 134

Fig. 29 DS CU Mostrar Estadísticas. Sección Estadísticas por proyecto..... 135

Fig. 30 DS CU Mostrar Estadísticas. Sección Estadísticas generales..... 136

Fig. 31 DS CU Mostrar Estadísticas. Sección Archivos Subidos 137

Fig. 32 Diagrama de clases del analizador de código estático Pixy 138

Índice de Tablas

Tabla 1 Actores del Sistema	45
Tabla 2 Estereotipos de clases UML.....	48
Tabla 3 Estereotipos Web de UML	59
Tabla 4 Estereotipos de relaciones entre clases	60
Tabla 5 Cálculo del factor de peso de los actores sin ajustar	86
Tabla 6 Cantidad de transacciones por casos de uso.	87
Tabla 7 Cálculo del factor de peso de los casos de uso sin ajustar.....	87
Tabla 8 Cálculo del Factor de Complejidad Técnica	91
Tabla 9 Cálculo del Factor Ambiente	92
Tabla 10 Distribución del esfuerzo estimado entre los flujos de trabajo de RUP.	94
Tabla 11 Descripción textual CU Subir Fuente.....	110
Tabla 12 Descripción textual CU Generar Reporte	117
Tabla 13 Descripción textual CU Gestionar Reportes	122
Tabla 14 Descripción textual CU Mostrar Estadísticas.....	129
Tabla 15 Funciones agregadas en Pixy	156

INTRODUCCIÓN

Hoy en día son cada vez más complejos los entornos de desarrollo *Web*, con un gran volumen de usuarios conectados, muchos de ellos considerados “*hackers*”, los que provocan problemas de seguridad en su interacción con *Internet*. Estos problemas, en la mayoría de los casos, se deben a errores de desarrollo de las aplicaciones, por lo que se hace necesaria la detección de los mismos en fases tempranas del ciclo de desarrollo del *software*. Una de las técnicas que se utilizan con ese fin es el análisis estático de código, que consiste en la revisión de las aplicaciones sin llegar a ejecutarlas. En este sentido, muchos autores comparten el criterio de que una de las mejores formas de detectar las debilidades de una aplicación es analizando su código fuente, mediante la utilización de herramientas automáticas, con lo cual se minimiza el posible efecto que puede ocasionar determinada vulnerabilidad.

Con respecto a *PHP*, lenguaje dinámico y débilmente tipado, el análisis estático de vulnerabilidades se convierte en una práctica imprescindible que debe considerarse desde la concepción de una aplicación *Web*. Para ello, existen algunas herramientas utilizadas en el análisis estático de vulnerabilidades en *PHP*, unas libres, otras propietarias, las que de una forma u otra detectan problemas de vulnerabilidad presentes en las aplicaciones *Web*, siguiendo determinados patrones y técnicas que han sido definidos. Sin embargo, el hecho de que algunas sean propietarias supone un gran problema para adquirirlas debido al monto monetario que exige su adquisición, además, son limitadas en su funcionamiento, ya sea por el tiempo en que pueden ser usadas o por las pocas funcionalidades que brindan al usuario para su uso, existen grandes restricciones por las licencias necesarias para adquirirlas. Las que son libres, por su parte, presentan como dificultad un incompleto desarrollo de *software*, ya sea por un deficiente entorno gráfico o insuficiencia en la búsqueda de vulnerabilidades producto a la no actualización a versiones actuales de *PHP* o excesiva presencia de falsos positivos o negativos en las aplicaciones que analizan. De igual forma, estas herramientas carecen de un control de las vulnerabilidades detectadas, su uso es en esencia informativa y local en la computadora en que se ejecutan, no permiten la persistencia de los

defectos encontrados para posteriores análisis. También, el resultado del análisis de estas herramientas produce algunos falsos positivos, es decir, detección de vulnerabilidades que no representan en realidad un problema o peor aún, falsos negativos, o sea, la no detección de vulnerabilidades cuando están presentes en el código.

Los problemas anteriormente planteados han provocado que el análisis estático de código no sea tenido en cuenta en la validación de la seguridad de proyectos realizados en *PHP*, como es el caso de la Universidad de las Ciencias Informáticas (*UCI*), en donde no existe un sistema que permita a los desarrolladores de *software* en el lenguaje de programación *PHP*, el control de calidad sobre las vulnerabilidades existentes en el código de las aplicaciones desarrolladas en este lenguaje, lo que no permite la garantía del cliente final en el producto que se le ofrece y ganancia de prestigio del centro de altos estudios en la producción de *software*. El análisis que se realiza a las aplicaciones *Web* en la universidad está basado en el análisis dinámico, es decir, se analizan las aplicaciones cuando se encuentran en producción, que si de cierto modo ayuda en la detección de problemas en el despliegue del producto, no se tiene en cuenta elementos como la localización exacta de problemas, posibles entradas y salidas de las aplicaciones y escaneo completo del código fuente. Esto repercute de forma negativa en la calidad del producto que ofrece la universidad, debido a que la seguridad de las aplicaciones, sobre todo las del tipo *Web*, es un requisito indispensable y exigido por los usuarios. De igual forma, en este centro universitario, según el cuarto estudio *Webmétrico* realizado [1], la extensión *PHP* representa el 99,16% de presencia en el mismo, es decir, más de 15 millones 697 mil 127 ficheros tienen esta extensión.

Teniendo en cuenta lo anteriormente planteado, surge **problema científico**: ¿cómo validar la seguridad del código *PHP* en la Universidad de las Ciencias Informáticas mediante un análisis estático?

Se plantea como **objeto de estudio**: análisis estático de código.

El **campo de acción**, como parte del objeto que se abstrae para su transformación, estará enfocado en: análisis estático de código para *PHP* en la Universidad de las Ciencias Informáticas.

El **objetivo general**, que como propósito se aborda, en este sentido, es: elaborar un sistema de análisis estático de código para la detección de vulnerabilidades en *PHP*.

Para cumplir con el objetivo trazado se desarrollan las siguientes **tareas investigativas**:

- Fundamentación teórica de las vulnerabilidades presentes en las aplicaciones *Web* realizadas en *PHP*.
- Diagnóstico de las ventajas y desventajas de herramientas similares de detección de vulnerabilidades en códigos *PHP*.
- Definición del diseño de un sistema para el tratamiento de vulnerabilidades en códigos *PHP*.
- Incorporación a la herramienta de análisis estático para *PHP* (*Pixy*) nuevas características, de forma que posibilite la búsqueda de vulnerabilidades en aplicaciones *PHP* de versiones recientes y brinde reportes detallados de los análisis realizados.

Como **idea a defender** se plantea: la utilización de un sistema de análisis estático de código para *PHP*, contribuirá a garantizar la seguridad y estabilidad de los sistemas desarrollados en este lenguaje en la *UCI*.

Teniendo en cuenta el cumplimiento de las tareas, los **posibles resultados** o **aportes prácticos** son: sistema para la detección de vulnerabilidades en códigos *PHP* en la *UCI* y actualización en cuanto a seguridad y nuevas funcionalidades del analizador estático *Pixy*.

Se utilizaron como **métodos de investigación** en el desarrollo del trabajo, los siguientes:

Teóricos:

- **Histórico – Lógico**: se hizo uso de él para conocer la evolución de los sistemas de detección de código estático, determinando igualmente las tendencias actuales en este sentido.

- **Analítico – Sintético:** se hizo uso de este método para tener una mejor comprensión del funcionamiento de los sistemas de detección de vulnerabilidades de código estático, sus características más relevantes, ventajas, desventajas, lo que permite evaluar con mayor claridad el posible desarrollo de un sistema lo más completo posible. Todo este análisis fue realizado con la previa revisión de la bibliografía consultada.
- **Modelación:** este método se usa porque permite crear abstracciones con el objetivo de explicar la realidad y además posibilita conocer la respuesta de los procesos sin tener que ejecutar los mismos en el mundo real.

Empíricos:

- **Observación:** se utilizó este método durante el proceso de investigación para comprobar las características y particularidades de algunos analizadores estáticos en cuanto al nivel de detección de vulnerabilidades.
- **Entrevista:** la utilización de este método empírico está relacionado con el conocimiento por parte de los creadores de herramientas como *Pixy* y *Yasca*, del avance de las mismas en su desarrollo, sus ventajas, desventajas y posibles mejoras que se pueden realizar en las mismas.

La novedad del trabajo radica en que la herramienta que se presenta brinda un servicio a los desarrolladores de *PHP* en la *UCI* para la detección de vulnerabilidades en aplicaciones realizadas en este lenguaje de programación, permitiendo un análisis y estudio de las mismas.

El trabajo ha sido estructurado en seis capítulos:

Capítulo 1: fundamentación teórica, se exponen conceptos, técnicas, tecnologías y tendencias de la propuesta presentada.

Capítulo 2: características del sistema, se ofrece una visión práctica del sistema, los requisitos funcionales y no funcionales, además de una propuesta del sistema.

Capítulo 3: análisis y diseño del sistema, se presenta una vista interna del sistema, se muestran los diagramas de clases del análisis, así como el diagrama de clases del diseño.

Capítulo 4: implementación del sistema, se muestran los elementos utilizados en la implementación del sistema, con todos los artefactos generados.

Capítulo 5: modificaciones al analizador de código estático Pixy, se presentan una serie de cambios realizados al analizador de código estático *Pixy*, así como algunas características que permiten su vinculación a un sistema *Web*.

Capítulo 6: factibilidad del sistema, se determina el costo del sistema, así como el tiempo que se necesita invertir para la realización del mismo.

CAPÍTULO I

FUNDAMENTACIÓN TEÓRICA

En el presente capítulo, se establece una valoración de los conceptos fundamentales del objeto de estudio y campo de acción de la investigación como tal. Se consideran, como elementos fundamentales los relacionados con la Seguridad Informática en *PHP*, analizadores de código para *PHP*, así como la propuesta de desarrollo del futuro sistema.

1.1. Seguridad Informática.

El término de Seguridad Informática está relacionado con las prácticas que se llevan adelante respecto a un determinado sistema de computación, con el objetivo de proteger y resguardar su funcionamiento, además de la información en él contenida.

La Seguridad Informática se rige por determinados principios que la sustentan y permiten su estudio y comprensión. A continuación se presentan estos principios [2]:

- El intruso al sistema intentará cualquier artilugio que haga más fácil su acceso y posterior ataque: todo atacante dirige su ataque hacia el punto débil de toda aplicación informática.
- Los datos confidenciales deben protegerse sólo hasta que ese secreto pierda su valor como tal: este caso se constata en los grandes sistemas que proporcionan muchos datos procesados, quizás éstos tengan valor para el usuario sólo hasta un determinado momento, luego van perdiendo importancia.

- Las medidas de control se implementan para que tengan un comportamiento efectivo, eficiente, sean fáciles de usar y apropiadas al medio: estas medidas deben funcionar en el momento oportuno y deben ser actualizados periódicamente, de lo contrario pasaría a ser un gasto inútil.

Un sistema seguro debe ser íntegro (la información es modificable sólo por personas autorizadas), confidencial (los datos tienen que ser legibles únicamente para los usuarios autorizados), irrefutable (el usuario no debe poder negar las acciones que realiza) y con buena disponibilidad (debe ser estable y disponible cuando se necesite).

1.2. Programación Segura.

Los elementos abordados en el epígrafe anterior referido a la Seguridad Informática, son ampliamente aplicables al concepto de Programación Segura, la cual se define como *la arquitectura, diseño y desarrollo de programas que siguen determinados patrones con el objetivo de reducir la existencia de problemas de seguridad en su código y garantizar los principios básicos de seguridad.* [3]

La Programación Segura como rama de la Seguridad Informática y la programación, define patrones que deben ser seguidos por los desarrolladores para lograr sistemas robustos, fiables y libres de errores, la misma hace uso de técnicas de testeado de software, utilización de funciones seguras, control del flujo de la información de un programa, entre otras, las que propician estabilidad y certeza del código escrito.

Idealmente, la concepción de elementos de Programación Segura debería ser parte inherente en el proceso de desarrollo de *software*, incluida como parte inicial del diseño de las aplicaciones. Sin embargo, en la mayoría de los casos, existen compromisos bajos en el tema de la seguridad que aumentan las posibilidades de crear aplicaciones vulnerables, lo cual no es la alternativa correcta por el efecto que presupone este criterio.

Se hace necesario tomar conciencia de la importancia de la codificación segura de las aplicaciones, mediante la identificación de puntos críticos, control del flujo de ejecución de los programas, aplicación de criterios conservativos (medidas que permiten la respuesta del sistema ante situaciones inesperadas), aplicación de elementos de la Ingeniería de Software al marco de la Programación Segura (por ejemplo, la programación por parejas que estimula la Programación Extrema). Estos elementos permiten la creación de un producto más completo y robusto, además de que minimizan los posibles ataques a que están expuestos los sistemas informáticos.

1.3. Seguridad en Aplicaciones Web.

Las aplicaciones *Web* han experimentado un acelerado crecimiento en los últimos años, dado, principalmente, a que se han convertido en el canal de comunicación fundamental entre clientes y proveedores de servicios; vinculado con este gran desarrollo, existe un negativo impacto en la seguridad de las aplicaciones, lo cual compromete la disponibilidad e integridad de la información con la cual se interactúa, lo que conduce a que se constaten enormes pérdidas materiales y a que se generen molestias en los usuarios en sentido general.

Según OWASP (Open *Web* Application Security Project) en su artículo “*The Ten Most Critical Web Application Security Risks -2010*” [4], los cinco problemas más serios, por orden de incidencia, hasta el año 2010 en las aplicaciones *Web* están enmarcados en:

- Inyección: ocurre cuando existen datos inseguros que se envían a un intérprete como parte de un comando o consulta.
- Cross site scripting (XSS): ocurre cuando una aplicación toma datos no seguros y los envía a un navegador *Web* sin la validación y escape necesarios, lo que trae como consecuencia la ejecución de scripts en el navegador de la víctima que producen daños terribles para el cliente.
- Autenticación y administración de sesión destruida: se relaciona con la autenticación y administración de sesiones, las cuales no son correctamente implementadas y permiten a los

atacantes comprometer contraseñas, llaves, etc., todo con el objetivo de asumir identidades falsas.

- Referencias inseguras de objetos directos: se producen cuando un desarrollador expone una referencia de una implementación interna de un objeto como un fichero, directorio o llave de una Base de Datos, lo cual provoca que los atacantes manipulen las referencias para acceder a datos no autorizados.
- Cross site request forgery (CSRF): se fuerza al navegador *Web* validado de una víctima a enviar una petición a una aplicación *Web* vulnerable, la cual entonces realiza la acción elegida a través de la víctima. Mediante *CSRF* se explota la confianza que un sitio tiene en un usuario en particular.

Un modelo común de vulnerabilidades en las aplicaciones *Web* lo representa el comprometimiento de la integridad, la cual es la causa fundamental de que se afecte la confidencialidad y disponibilidad de estas aplicaciones. Los atacantes utilizan datos que no son seguros para construir salidas seguras que no son validadas, lo que provoca violaciones de la integridad de los datos. Esta violación permite la escala de los derechos de acceso, lo cual puede representar infracciones en la disponibilidad y confidencialidad de la información. La explicación anterior es mostrada gráficamente a continuación: [5]

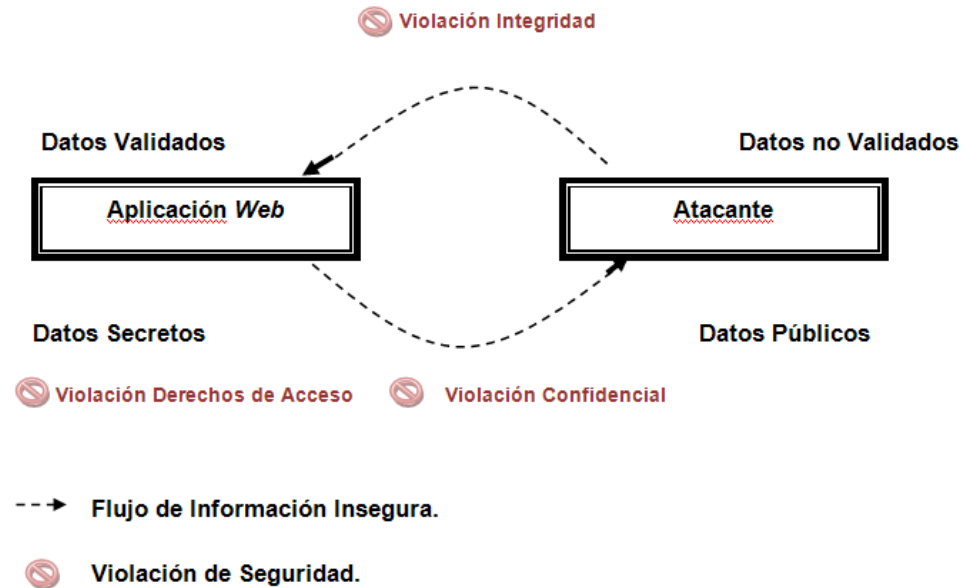


Fig. 1 Esquema general de un modelo común de vulnerabilidades *Web*.

1.4. Seguridad en Aplicaciones *PHP*.

La popularidad que ha adquirido *PHP* a nivel mundial y los crecientes niveles de ataques que se producen en la red de redes, constituyen aspectos de interés en el estudio de la seguridad de este lenguaje, en el que se observan características de seguridad muy especiales y es importante tenerlas en cuenta por la vitalidad que representa el tema.

Las facilidades y gran flexibilidad en la escritura de código que brinda el lenguaje *PHP* a los desarrolladores, representa en ocasiones un problema. Ello es debido a que para los atacantes no es muy difícil detectar algunas vulnerabilidades de los programadores en sus aplicaciones, en tanto que conocen las fallas de seguridad más usuales y es ahí hacia donde dirigen sus ataques. La seguridad de *PHP* incluye la minimización de los errores de programación, y la colocación del código apropiado en su lugar para eliminar toda posible vulnerabilidad.

A continuación se presentan aspectos importantes para la seguridad en el lenguaje *PHP*: [6]

- Register Globals.

Esta es una directiva a través de la cual se pueden escribir aplicaciones simples y convenientes para el desarrollador donde las variables tienen un ámbito global, sin embargo, esto puede causar un riesgo de seguridad potencial. La misma no está disponible por defecto en la versión 4.2.0 y superiores de *PHP*.

Esta propiedad se localiza en el archivo de configuración de *PHP*, el cual, por lo regular lleva por nombre *php.ini* y puede ser, tanto activado, como desactivado. Al activarlo, se permite que los usuarios no verificados inyecten variables en una aplicación para ganar acceso administrativo a la *Web*. La mayoría de los expertos en *PHP* recomiendan desactivar *register_globals*.

En el siguiente código se presenta el peligro que representa la activación de esta directiva:

```
if (isset ($POST_['$user']) && $POST_['password']) == "12345"){  
  
    $admin = true;  
  
}
```

En el ejemplo expuesto, un usuario podría añadir el final de la *URL* de una página *?admin=1* para básicamente forzar la entrada a áreas administrativas que normalmente requerirían una contraseña. En este caso la variable *\$admin* tiene un ámbito global en la aplicación si se tiene *register_globals* activado.

- Reportes de Error.

Con esta herramienta se permite el diagnóstico de errores de una aplicación *Web* realizada en *PHP*, lo que permite que la corrección sea rápida y bastante sencilla. Sin embargo esto supone un problema de seguridad. Si el error es visible a personas no autorizadas, se pueden revelar posibles agujeros de

seguridad en el código de los cuales los *hackers* pueden utilizar sin problemas y elaborar acciones negativas en este sentido.

Las directivas más importantes en este sentido que se encuentran en el archivo de configuración *php.ini* son:

- Display_errors: controla la forma en que los errores en *PHP* se envían a la pantalla.
 - Error_reporting: controla cuáles errores deben reportarse.
 - Log_errors: se especifica si los errores que se encuentran deben ser guardados en un archivo.
 - Error_log: es la localización del archivo de *logs* de errores, en el cual se escribirán cuando ocurra alguna incidencia en el sistema.
- Cross-Site Scripting (XSS).

Este es uno de los problemas más graves que presentan no sólo las aplicaciones en *PHP*, sino en sentido general, todas las que se realizan para la *Web*. Ella constituye una forma en que los *hackers* reúnen información del sitio *Web* utilizando código *JavaScript* para engañar al usuario o a su navegador, para que sigan un enlace no deseado o que presenten sus detalles personales mediante diferentes métodos, para así robar sus datos. El ataque se produce casi siempre en sitios que brindan la posibilidad de interacción con el usuario mediante foros, etc. Esto puede tener un impacto negativo para el usuario debido a que la falsificación que supone este tipo de ataque representa una grave violación de la confidencialidad que maneja un sitio *Web*.

Los ataques tipo *XSS* pueden clasificarse de dos formas:

- Persistentes: el código malicioso no siempre necesita ser inyectado mediante el navegador del usuario víctima, sino que este código ha sido previamente guardado por el atacante en la base de datos que utiliza el sitio víctima. De esta forma, cada vez que el sitio recupere y muestre

registros de su base de datos, el script maligno estará allí y será ejecutado para cualquier usuario visitante.

- No persistentes o reflejados: son aquellos en los que un script malicioso debe ser enviado explícitamente mediante el navegador del usuario víctima cada vez que el atacante desee ejecutar códigos malignos en ese navegador para enviar este tipo de script, se utilizan las variables `$_GET`, `$_POST` y `$_COOKIES`.

Ambas modalidades de XSS son altamente peligrosas, pero sin duda alguna los persistentes son mucho más dañinos por el hecho de encontrarse el código maligno en el propio servidor.

Un ejemplo del efecto de un ataque XSS se presenta a continuación:

```
<form>

<input type="text" name="message"><br/>

<input type="submit">

</form>

<?php

if (isset($_GET['message']))

{

    $fp = fopen('./messages.txt', 'a');

    fwrite($fp, "{$_GET['message']}<br />");

    fclose($fp);

}
```

```
readfile('./messages.txt');
```

```
?>
```

Si un usuario escribiera el siguiente mensaje, el próximo usuario que visitara el sitio sería redireccionado a la dirección *http://evil.example.org* y serían robadas las *cookies* asociadas al sitio, provocando un grave problema de seguridad:

```
<script>
```

```
document.location = 'http://evil.example.org/steal_cookies.php?cookies=' + document.cookie
```

```
</script>
```

Estos ataques pueden ser tan creativos como su creador lo decida y por tanto más dañinos para una víctima de los mismos.

La mejor forma de defensa ante *XSS* es deshabilitar *JavaScript* y las imágenes al surfear la *Web*, pero por causa de las grandes posibilidades a las que se restringe el usuario con esta opción, casi nunca se realiza.

Una manera más creativa de constatar este tipo de ataque *XSS*, se asocia al procesamiento de las variables que el usuario pueda manipular, lo cual se codifica de forma apropiada y es preciso siempre validar toda entrada y salida de una aplicación *Web* realizada en *PHP*. El lenguaje dispone de un conjunto de funciones para este fin lo que propicia la tranquilidad de todos los involucrados en una aplicación *Web*.

Si en el ejemplo anterior se hubiera sustituido la línea marcada en negro por la que se presenta a continuación, se podría evitar cualquier daño tipo *XSS* al sitio *Web*:

```
fwrite($fp, "{htmlentities($_GET['message'])}<br />");
```

El detalle está en la función *htmlspecialchars*, disponible en *PHP* para convertir todos los caracteres a su entidad *HTML* aplicables, por ejemplo en, “<”, se convertiría en “<”.

Otra forma adicional de protección contra un ataque *XSS* es el uso de *cookies* de sesión *HTTPOnly*. Este tipo de *cookies* tiene la particularidad de que es imposible obtener su valor desde *JavaScript*. De esa forma todo ataque *XSS* que intente obtener el *SID* mediante *document.cookie* (como en el ejemplo que se presentó) puede quedar inutilizado. Esta es una solución parcial que debe trabajar en conjunto con otras soluciones debido a que ello protege solamente las *cookies* del usuario, pero no otros tipos de ataques *XSS* como en el caso de una redirección o cualquier otro que pueda aparecer.

- SQL Injections (SQLi).

Es otro de los problemas más comunes y más antiguos que presentan las aplicaciones realizadas en *PHP*. Su potencial destructivo es enorme.

El motivo de que esté tan extendido este tipo de errores reside en la falta de conciencia en torno a la seguridad que debe ser tenida en cuenta por los programadores, pues no prestan atención a la importancia de ésta en una aplicación *Web*.

Su funcionamiento se basa en la “inyección” de código *SQL* en consultas que han sido escritas en una aplicación *Web*. Normalmente una consulta *SQL* está influenciada por la entrada de un usuario debido a que en su composición existen variables que serán sustituidas por las entradas de éstos.

Los ataques asociados a inyecciones *SQL* están vinculados fundamentalmente a:

- La obtención de la base de datos completa utilizando sentencias *SELECT*.
- La modificación o inserción de datos usando *INSERT* o *UPDATE*.
- El borrado de la base de datos usando *DELETE*.

- La propia ejecución de comandos del sistema operativo usando EXEC.
- El apagado remoto del servidor.
- El compartimento de ficheros.

Un ejemplo del grave efecto de las inyecciones SQL se presenta a continuación en la siguiente consulta:

```
$sql = "INSERT INTO usuario (name) VALUES ('${_POST['nombre_usuario']}');"
```

La consulta anterior inserta en la tabla *usuario* cualquier elemento que se haya introducido por cualquier usuario.

Si un atacante escribiera por el siguiente código se podría producir algo desagradable, pues se eliminarían todos los usuarios de la Base de Datos:

```
INSERT INTO usuario (name) VALUES ('Pedro'); DROP TABLE usuario;---
```

De esta forma, se está terminando una consulta con el punto y coma (;) y lo que venga después se interpreta como una nueva consulta, en este caso eliminar la tabla *usuario*. Los 3 guiones (---) presentes al final de la consulta permiten que todo lo que aparezca después de la misma se pueda interpretar como comentario, el intérprete SQL no devolvería error alguno.

La protección contra *SQL Injections* es bastante simple, la regla fundamental es considerar que ninguna entrada es segura y todo dato debe ser validado.

- Ejecución de Código Remoto.

Desde el año 2004, una de las cuestiones en cuanto a seguridad se refiere más latentes en *PHP* es la relacionada con la ejecución de código remoto. En este sentido, se incluye el ataque por inclusión de archivos remotos o *RFI (Remote File Intrusion)*. Su efecto es bastante dañino debido a que se produce la inclusión de archivos remotos, o *RFI*, mediante la exploración remota de una aplicación *PHP* vulnerable,

inyectándose código malicioso o incluso acceso a la carpeta ruta del servidor. Estos ataques pueden incluir la ejecución de virus en el servidor *Web*, se gana acceso al servidor, lo que puede causar problemas serios como el abuso de información personal almacenada en las Bases de Datos.

En el siguiente ejemplo se presenta la acción negativa de un ataque RFI:

```
<?php
    $pagina = isset($_GET['pagina']) ? $_GET['pagina'] : 'inicio';
    require $pagina . '.php';
?>
```

En el *script* anterior se incluye en el código mediante la instrucción *require* una página obtenida por la entrada de un usuario.

Si el sitio se encontrara en la dirección *http://ejemplo.com/index.php* y un atacante tiene una página con código malicioso en la dirección *http://atacante.com/maligno.php*, el mismo puede realizar un ataque RFI de la siguiente forma: *http://ejemplo.com/index.php?pagina=http://atacante.com/maligno*. Este fichero será ejecutado en el servidor, produciendo efectos insospechados.

Existen directivas en el archivo *php.ini* para evitar estos tipos de ataques. Ellas son:

- Allow_url_fopen: por defecto está activada, controla si los ficheros remotos deben ser incluidos cuando se requiera.
- Allow_url_include: por defecto está desactivada en el archivo de configuración. Controla si las sentencias *include()*, *require()*, *include_once()* and *require_once()* pueden ser capaces de incluir ficheros remotos.

En la ejecución remota existe un punto muy importante de atención, es decir, los comandos del *shell*. Cuando un *script PHP* contiene una línea que invoca comandos de la consola, éstos deben ser seguidos cuidadosamente, pues los parámetros contenidos en las funciones que *PHP* brinda para este fin, pueden ser realmente muy peligrosas. Un atacante astuto puede tomar el control incluso del servidor *Web* donde se ejecuta la aplicación. En este sentido existen algunas funciones de *PHP* que deben ser tenidas en cuenta en este sentido. Ellas son: *exec()*, *passthru()*, *proc_open()*, *shell_exec()*, *system()*, etc.

- Seguridad en Sesiones.

En *PHP* las Sesiones representan una herramienta muy importante en el desarrollo de las aplicaciones, debido a que permiten “recordar” datos del usuario entre una visita y otra. El uso clásico de las mismas se presenta cuando un visitante ha iniciado su sesión en un sitio, las sesiones en este sentido permiten que no tenga que ingresar su contraseña nuevamente cuando vuelva a entrar.

PHP le asigna a cada sesión un identificador único llamado *PHPSESSID*, el cual le es enviado al cliente mediante una *Cookie*, del lado del servidor *PHP* se genera un archivo con el nombre de *PHPSESSID*, en el cual se guarda la información de las variables de la sesión.

Una de las formas de lograr accesos no autorizados a un sistema puede ser por medio del secuestro de la sesión de un usuario conectado a la aplicación. El atacante de alguna forma podría acceder a la *Cookie*, de esta forma es posible conocer el *PHPSESSID* que *PHP* ha asignado al cliente. Con este procedimiento la sesión puede ser comprometida y el atacante podría acceder al sistema sin tener siquiera que buscar un nombre de usuario y una contraseña, escribiendo solamente la *URL* y en una *Cookie* el *PHPSESSID*.

La solución en este caso está asociada a los siguientes aspectos, aunque son variadas las alternativas en este sentido:

- Regenerar los identificadores de las sesiones cada vez que se usan mediante la función *session_regenerate_id()*.

- Utilizar tokens en el arreglo `$_SESSION` y verificar la integridad de éste cada vez que se inicializa la sesión;
- Encriptar las claves del arreglo `$_SESSION`.
- Registrar la IP del cliente al momento de crear la sesión de autorización para el sistema.

1.5. Tipos de análisis a aplicaciones *Web*.

Existen dos tendencias en el análisis de aplicaciones *Web*. En sentido general, la utilización de una u otra en el análisis de código es de libre decisión por el usuario, cada una tiene características que las hacen especiales para determinado entorno de desarrollo.

1. *Análisis Dinámico.*

En el caso que se examina, se observa que la aplicación en ejecución para ver el comportamiento de la misma en diferentes situaciones, posee sus peculiaridades.

Las ventajas resultantes de este tipo de análisis son:

- Identifica vulnerabilidades en un entorno en ejecución con un tiempo relativamente corto.
- Permite analizar aplicaciones a las que no se puede acceder a su código fuente actual.
- Identifica vulnerabilidades que podrían ser falsos positivos en el análisis estático.
- No requiere tener acceso al código fuente de la aplicación.
- Se revisa la aplicación en el estado final: configuración, plataforma, etc.

Sus limitaciones están dadas por:

- Es muy difícil encontrar una vulnerabilidad en su localización exacta en el código, tomando gran

cantidad de tiempo resolver el problema.

- No es posible examinar todas las posibilidades en un tiempo limitado.
- Existe el riesgo de afectar el servicio (generalmente en producción).

2. *Análisis Estático.*

Se detectan vulnerabilidades a través de la revisión automatizada del código fuente de las aplicaciones sin llegar a ejecutarlo. El análisis se realiza principalmente en base a patrones o reglas, mediante el análisis del flujo de los datos o métricas para detectar problemas de seguridad en el desarrollo.

Algunas de las ventajas de este análisis radican en que:

- Se revisan todas las posibles entradas/salidas de una aplicación.
- Se encuentran problemas del código en su localización exacta.
- Es rápido si se utilizan herramientas automatizadas.
- Puede ser escaneado completamente el código base.
- Permite encontrar errores en el desarrollo de las aplicaciones bien temprano en el ciclo de desarrollo del software, lo que reduce el costo de arreglarlo.

Algunas de sus limitaciones se encuentran en que:

- Requiere el acceso al código fuente de la aplicación.
- Se producen falsos positivos y negativos.
- Pueden producir una falsa sensación de seguridad.

- No se pueden analizar todos los elementos del despliegue final (configuración del servidor o del entorno).

1.6. Tipos de análisis de código estático.

En el análisis de código estático existen tres técnicas fundamentales, las que determinarán las capacidades de detección de problemas en las aplicaciones *Web*, ellas son: [7]

- Análisis de flujo de datos: forma parte de los llamados Métodos Formales¹. Es una técnica basada en la identificación de los parámetros de entrada de las aplicaciones (manipulables por un atacante) y detectar la ausencia de validaciones en cualquier punto de la aplicación en que se trate. Detecta la propagación de las variables a lo largo de un programa.
- Análisis semántico: valida el uso de código potencialmente peligroso por sí mismo, es decir, llamadas a funciones y procedimientos vulnerables;
- Análisis estructural: se encarga de detectar problemas debido a una incorrecta organización de sentencias en el código.

1.7. Características deseables de los analizadores de código estático.

Dos aspectos de importancia en la selección de cualquier herramienta informática lo constituyen la variedad de características deseables que deben brindar al usuario y las ventajas de sus especificidades técnicas.

Los analizadores de código estático deben ofrecer una serie de posibilidades, lo cual repercute en el criterio de evaluación y selección de los mismos. Algunas de estas características incluyen: [9]

¹ Es el término aplicado al análisis de *software* (y *hardware*), cuyos resultados se obtienen exclusivamente mediante el uso de rigurosos métodos matemáticos. Las técnicas matemáticas utilizadas incluyen semántica denotativa, la semántica axiomática, la semántica operacional y la interpretación abstracta. [8]

- Capacidad en la detección de problemas: se entiende en este sentido la capacidad de la herramienta en la detección de vulnerabilidades y la clasificación que realiza de las mismas.
- Gestión ágil de falsos positivos: estas herramientas deben ser capaces de permitir la gestión de vulnerabilidades que se detectan por errores de las mismas, evitando que vuelvan a mostrarse en análisis posteriores.
- Posibilidad de ajuste del nivel de riesgo: el nivel de riesgo o las comprobaciones a realizar podrán no ser las mismas en todas las aplicaciones Web, se pueden aplicar incluso plantillas diferentes a distintos análisis.
- Calidad de las recomendaciones para la mitigación de problemas e informes generados: los reportes y recomendaciones generados deben ser concretos y personalizados al lenguaje al que se aplica el análisis.
- Capacidades de análisis incremental: evita el procesamiento de todo el código en cada análisis, teniendo en cuenta los análisis previos.
- Revisión evolutiva de los análisis: en este sentido se debe ser capaz de estudiar los avances en la generación de código seguro por parte del equipo de desarrollo, además de comparar el resultado del análisis a varias aplicaciones o tendencias en la seguridad de las aplicaciones de la organización.
- Posibilidades de gestión de informes: se considerarán los mecanismos de distribución de informes a todas las partes que intervienen en la fase de análisis. Se distinguen varias opciones como interfaces Web autenticadas, envío por correo integrados en los entornos de desarrollo, etc.

1.8. Analizadores de código estático para *PHP*.

1.8.1. Analizadores propietarios.

1.8.1.1. Code Secure.

Analizador multilenguaje que detecta vulnerabilidades en *ASP.NET*, *VB.NET*, *C#*, *Java/J2EE*, *JSP*, *EJB*, *PHP*, *Classic ASP* y *VBScript*. La misma forma parte de un conjunto de aplicaciones de seguridad de *Armorize Appsec Suite*.

Ésta modela el comportamiento de una aplicación *Web* y tracea el flujo de datos de cada entrada vulnerable de un fichero. Además, brinda reportes detallados con estadísticas y gráficos vinculados del análisis realizado.

1.8.1.2. CodeScan.

Es una herramienta avanzada de seguridad diseñada para analizar vulnerabilidades de varios entornos de desarrollo como *PHP* y *ASP*. El mismo pertenece a los laboratorios *CodeScan Labs Ltd* y utiliza técnicas de análisis inteligentes para buscar los posibles valores de las variables e identificar si poseen algún índice de peligrosidad.

La versión Demo sólo permite analizar vulnerabilidades del tipo *SQLi* y el tiempo de uso que brinda la herramienta es de quince días. El costo de esta herramienta para *PHP* es \$397 USD.

1.8.2. Analizadores de Código Libre.

1.8.2.1. PHP-SAT.

Es una de las herramientas de análisis de código estático más interesantes entre las existentes para *PHP*. Sus creadores son Eric Bouwers y Martin Bravenboer. Esta se encuentra bajo licencia *LGPL*.

La herramienta define su funcionamiento en patrones, es decir, los que son correctos acorde a la gramática de *PHP*. La misma se basa en la documentación de *PHP* y en experiencias pasadas, además, reflejan información de posibles errores y el programador tiene que decidir cuándo un patrón constituye un error en una situación específica.

Los patrones se clasifican en cuatro categorías:

- Corrección.
- Exposición de Información.
- Optimización.
- Estilo.

La herramienta escribe en el mismo código analizado el resultado de su análisis. No existe para ella, actualmente, una versión estable. Su instalación es un tanto difícil, además, de la comprensión del código fuente.

1.8.2.2. SWAAT (*Securitycompass Web Application Auditing Tool*).

Herramienta de código abierto construido y mantenido por *Security Compass*. Tiene soporte para analizar código *Java*, *JSP*, *ASP*, *.NET* y *PHP*.

SWAAT busca a través de una base de datos de cadenas potencialmente peligrosas, dados en archivos *XML* que contienen expresiones regulares.

Para la ejecución del análisis es necesaria la consola de comandos. Los reportes se constatan a través de una página *HTML*, donde se imprime el nombre de la vulnerabilidad que se detecta, además de la severidad, descripción y localización.

El método de búsqueda que implementa *SWAAT* en opinión de sus creadores no es sofisticado. Se espera en futuras versiones una ampliación y mejores métodos de búsqueda del mismo.

1.8.2.3. RATS (*Rough Auditing Tool for Security*).

Es una herramienta de seguridad que se desarrolla y mantiene por ingenieros de seguridad de *Secure Software*. Ella realiza el escaneo de código *C++*, *Perl*, *PHP* y *Python*.

En la realización del análisis se hace uso de una lista de vulnerabilidades potenciales de cada lenguaje, esbozados en archivos *XML*, donde se describe la vulnerabilidad, o sea, descripción y recomendaciones para solucionar el problema.

Según los propios creadores, el hecho de que la herramienta base su funcionamiento en una base de datos de vulnerabilidades, realiza un análisis "rústico" del código fuente y no encuentra muchos defectos y genera gran cantidad de falsos positivos.

1.8.2.4. YASCA (*Yet Another Source Code Analyzer*).

Esta representa una herramienta de análisis multilenguaje, la cual está escrita en *PHP* y con licencia *BSD*, su creador, Michael Scovetta. El funcionamiento de la herramienta es mediante el uso de *plugins*, recopilando analizadores de códigos libres disponibles no sólo de seguridad, también escáneres de sintaxis de código. Algunos de estos analizadores son *FindBugs*, *JLint*, *PMD* y *Pixy*, en los mismos, no se realizan cambios ni innovaciones, todo lo cual es capaz de escanear código de *C++*, *Java*, *JavaScript*, *ASP*, *PHP*, *HTML/CSS*, *ColdFusion*, *Cobol*, entre otros.

La filosofía del mismo se localiza en brindar un conjunto de herramientas a los desarrolladores para que su trabajo sea mejor en cuanto a la seguridad y métricas del *software*.

1.8.2.5. Pixy.

El mismo realiza análisis a dos de los tipos más comunes de vulnerabilidades en *PHP*, es decir, *SQLi* y *XSS* sobre código *PHP 4*. Es una aplicación realizada en *Java* y bajo licencia *GPL*. Se basa en el método formal de análisis del flujo de datos del programa para verificar el estado de las variables a lo largo del mismo. Su funcionamiento tiene tres características fundamentales: [10]

- Sensibilidad al flujo: computa información de cada punto sensible del programa. Por

ejemplo, si se considera el siguiente ejemplo:

1. `$x = 'ok';`
2. `echo $x;`
3. `$x = $_GET['x'];`
4. `echo $x;`

La herramienta en este caso lanza una alarma del tipo XSS sólo para la segunda sentencia `echo`, pero no para la primera. Ello ocurre debido a que en la primera instrucción se imprime un valor totalmente “sano”, sin embargo, en la segunda, se muestra el valor de una variable recopilada del usuario sin la debida validación.

- **Interprocedural:** se refiere al análisis que se realiza cuando se hacen llamadas a funciones, o sea, a parámetros de funciones y valores de retorno. En el ejemplo:

1. `$x = $_GET['x'];`
2. `$y = foo($x);`
3. `echo $y;`
4. `function foo($p) {`
5. `return $p;`
6. `}`

La herramienta lanza una alarma correcta del tipo XSS por el uso de la variable `$y` cuando se imprime con la instrucción `echo`, exactamente, en la línea tres. Esto es debido a que la

variable `$y` se le asigna el retorno de una función (línea 2) la cual devuelve una variable que no se ha filtrado, ni validado.

- Sensibilidad al contexto: distingue la diferencia entre diferentes puntos de llamadas de una función y propaga valores de retorno correctos para cada llamada. En el siguiente ejemplo:

```
1. $x = foo($_GET['x']);  
2. echo $x;  
3. $y = foo('ok');  
4. echo $y;  
5. function foo($p) {  
6.     return $p;  
7. }
```

Como se constata, se reporta una vulnerabilidad en la primera impresión que se realiza (línea dos). Ello se produce, en tanto que en la línea uno, se le ha asignado a la variable `$x` el valor de retorno de una función que devuelve un elemento vulnerable. Sin embargo, en la línea cuatro, donde se produce un caso similar con la diferencia de que a la función de la línea tres, le es asignado como parámetro un valor “limpio” (“ok”), que no resulta ninguna vulnerabilidad. Una herramienta que no tenga en cuenta esta diferencia propaga el posible peor valor de retorno, es decir, se produciría una falsa alarma en la línea cuatro, cuando esto no es correcto.

Pixy utiliza como analizador léxico *JFlex* [11] y como analizador sintáctico *JCup* [12], los que generan gramáticas tipo *LALR* [13].

Para cada vulnerabilidad que se detecta, *Pixy* genera un grafo de dependencia, en el cual se muestra el origen del problema detectado y con ello se brinda la comprensión o posible solución de la vulnerabilidad detectada.

Este analizador, para ser ejecutado, necesita de la consola de comandos y su ejecución consta de la llamada de un archivo *.bat* (en el caso de *Windows*) o *.pl* (*Linux*) seguido del fichero que se quiera analizar. Los reportes de la herramienta se brindan en la misma consola, se ofrece también la posibilidad de ver los grafos generados en la carpeta que crea para los mismos.

El hecho de que sólo analice código *PHP 4* representa un gran problema, dado a que no se consideran elementos de la Programación Orientada a Objetos, funciones ni elementos novedosos en sintaxis de la versión 5 de *PHP*.

En declaraciones de Engin Kirda y Christopher Kruegel, asesores del proyecto *Pixy*, el desarrollo del mismo se detuvo en el año 2007, el conocimiento detallado de la herramienta fue perdido y hasta el momento no existen planes de continuar con su expansión.

Pixy es una de las alternativas de *Software Libre* para el análisis de vulnerabilidades en código estático más completas que existe, aunque la misma se encuentra restringida a la versión 4 de *PHP* y sus resultados se esbozan en la consola de comandos. Ella posee un algoritmo de búsqueda de vulnerabilidades muy efectivo y eficiente, además de que es flexible en las actualizaciones que se pueden realizar hacia versiones recientes de *PHP*.

1.9. Propuesta de Desarrollo del sistema.

1.9.1. Plataforma de Desarrollo Linux.

El uso de *Linux* es una posibilidad real a la hora de implementar un sistema. Dada su filosofía de libertad de código, se encuentran muchos programas cuyo código fuente se pueden observar para aprender y tomar ideas. También, es posible corregir o extender esos programas para adaptarlos a las necesidades

propias del desarrollador. *Linux* ha demostrado ser una plataforma seria para el desarrollo, provee gran variedad de soluciones y además la capacidad de crecer continuamente por medio de su comunidad. *Linux* se resume en seguridad, rapidez y economía.

1.9.2. XML (Extensible Markup Language).

Es un Lenguaje de Etiquetado Extensible muy simple, pero estricto que juega un papel fundamental en el intercambio de gran variedad de datos. Es similar a *HTML* pero su función principal es describir datos y no mostrarlos como es el caso de *HTML*. *XML* es un formato que permite la lectura de datos a través de diferentes aplicaciones.

Las tecnologías *XML* son un conjunto de módulos que ofrecen servicios útiles a las demandas más frecuentes por parte de los usuarios. *XML* sirve para estructurar, almacenar e intercambiar información.

Algunos usos prácticos de *XML* están enmarcados en la comunicación de datos, migración de datos y aplicaciones *Web*.

1.9.3. Software de Visualización Gráfica Graphviz.

Graphviz es una aplicación de código abierto para realizar grafos, árboles o diagramas de relaciones, es decir, es un *software* para la visualización de datos. El mismo consiste en la descripción del lenguaje gráfico *DOT* [14] y un conjunto de herramientas que generan y procesan ficheros *.DOT*. Este *software*, puede ser ejecutado mediante la línea de comandos en sistemas *Unix*, lo que permite su uso en sistemas que generan archivos *.DOT* y su conversión a formatos como *.GIF*, *.JPG* e incluso *.PDF*. De igual forma, *Graphviz* tiene soporte para diferentes Sistemas Operativos, es utilizado en sistemas de análisis estático de código como *Pixy* para mostrar el flujo que toman las variables a lo largo de su uso en una aplicación.

1.9.4. Lenguaje de Modelado UML.

El Lenguaje Unificado de Modelado (*UML*) es uno de los más conocidos en la actualidad. El mismo es utilizado para especificar, construir, visualizar y documentar los artefactos de un sistema de *software*

orientado a objetos. También, este lenguaje permite la abstracción de la realidad que se plasma en notación gráfica conocida como modelado visual. Este lenguaje permite igualmente el modelado de sistemas complejos, tanto de *software* como de arquitectura de *hardware* donde se ejecuten.

1.9.5. Metodología de desarrollo RUP.

La Metodología de Desarrollo RUP es una plataforma flexible de proceso de desarrollo de *software* que ayuda a brindar un proceso de guía personalizado y más consistente a los equipos de proyecto. A través de RUP es posible conseguir una mejor estructura y disciplina del proceso de desarrollo. La misma se divide en nueve flujos de trabajo y cuatro fases, cada fase se desarrolla mediante iteraciones que generan artefactos. De igual forma, se distinguen como características fundamentales del mismo: iterativo incremental, dirigido por casos de usos y centrado en la arquitectura, también, genera gran cantidad de documentación para lograr un mejor entendimiento entre los involucrados en un proyecto. Esta metodología es una de las más importantes y está enfocada en el desarrollo de proyectos de gran envergadura, aunque puede ser utilizada en los de mediana y pequeña complejidad igualmente. [15]

1.9.6. Herramienta CASE Visual Paradigm.

Es una herramienta CASE que proporciona excelentes facilidades de interoperabilidad con otras aplicaciones, también es muy completa y fácil de usar y con soporte multiplataforma.

Visual Paradigm para UML se ha diseñado para una amplia gama de usuarios, incluidos los ingenieros de *software*, analistas de sistemas, analistas de negocios, sistema de arquitectos, al igual que para aquellas personas interesadas en la construcción de sistemas de forma fiable a través de la utilización del enfoque orientado a objetos.

En esta oportunidad, se consta de otras características como generación del código, creación de ingeniería, tanto directa como inversa, lo que permite invertir al código fuente de programas, archivos ejecutables y binarios en modelos UML al instante, generando, de manera simple toda la documentación.

Ella se encuentra diseñada para usuarios interesados en sistemas de *software* de gran escala, incorpora el soporte para el trabajo en equipos, lo que permite que varios desarrolladores trabajen a la vez en el mismo diagrama y que puedan constatar, en un período real, los cambios realizados por otros usuarios. Por otro lado, ella se encuentra disponible en múltiples plataformas, tal y como es el caso de *Microsoft Windows (98, 2000, XP, o Vista)*, *Linux*, *Mac OS X*, *Solaris* o *Java* y, finalmente, ofrece un número considerable de estereotipos a utilizar, los que permiten un mayor entendimiento de los diagramas.

1.9.7. Entorno de Desarrollo Eclipse.

Eclipse es un Entorno Integrado de Desarrollo para todo tipo de aplicaciones libres. El mismo es una herramienta para el programador desarrollada principalmente para el desarrollo de aplicaciones *Java*, facilitando al máximo la gestión de proyectos colaborativos mediante el control de versiones.

Algunas de las características fundamentales de Eclipse son:

- Multiplataforma (*GNU/Linux*, *Solaris*, *Mac OSX*, *Windows*).
- Soportado para distintas arquitecturas (x86, 64, etc.).
- Presenta una estructura de *plugins* que hace sencillo añadir nuevas características y funcionalidades.
- Control de versiones con *subversion*.

1.9.8. Servidor Web Apache.

Apache es un servidor *Web* potente y flexible, el cual puede funcionar en la más amplia variedad de plataformas y entornos. De igual forma, es el servidor *Web* hecho por excelencia, su configurabilidad, robustez y estabilidad hacen que cada vez millones de servidores reiteren su confianza en este programa.

Algunas características distintivas de este servidor *Web* son:

- Funciona en una multitud de Sistemas Operativos, lo que lo hace prácticamente universal.
- Es un servidor altamente configurable de diseño modular, es muy sencillo ampliar las capacidades del mismo.
- Permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor, es posible configurarlo para que ejecute un determinado script cuando ocurra un error en concreto.
- Tiene una alta configurabilidad en la creación y gestión de logs, Apache permite la creación de ficheros de log a la medida del administrador, de este modo se puede tener un mayor control sobre lo que sucede en el servidor.

1.9.9. Servidor FTP VSFTPD (*Very Secure FTP Daemon*).

Es un servidor *FTP* bajo licencia *GPL* para sistemas *UNIX* incluyendo *Linux*. Es seguro y extremadamente rápido, además de ser estable. El mismo se distingue principalmente porque sus valores predeterminados son muy seguros y por su sencillez en la configuración.

Según estudios realizados, en 24 horas, VSFTPD sirvió 2.6 TB con una concurrencia de 1500 usuarios.

Algunos de los sitios reconocidos que usan este servidor FTP son: *ftp.redhat.com*, *ftp.suse.com*, *ftp.debian.org*, *ftp.freebsd.org*, *ftp.gnu.org*, *ftp.gnome.org*, *ftp.kde.org*, entre otros.

1.9.10. Gestor de Base de Datos PostgreSQL.

El Gestor de Base de Datos *PostgreSQL* es un sistema de base de datos multiplataforma muy potente dirigido por una comunidad de desarrollo. El mismo soporta gran parte del estándar *SQL* y, en algunos aspectos, está diseñado para que sea extensible por los usuarios. Éste posee interfaces gráficas de usuarios y enlazadores para algunos lenguajes de programación. Este gestor es *software* libre y está bajo licencia *BSD*. De igual forma, el mismo permite una alta concurrencia, debido fundamentalmente a que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. La

escalabilidad que presenta el mismo es muy interesante, es capaz de ajustarse al número de *CPU* y a la cantidad de memoria que posee el sistema de forma óptima, lo cual permite aceptar muchas peticiones simultáneas al mismo tiempo. También, tiene la capacidad de comprobar la integridad referencial e implementa el uso de *rollback*, subconsultas y transacciones, lo que posibilita un funcionamiento mucho más eficaz.

1.9.11. Lenguaje de Desarrollo PHP (Hypertext Preprocessor).

Es un lenguaje de código abierto que se interpreta con un alto nivel embebido en páginas *HTML* y se ejecuta en el servidor al nivel más básico, *PHP* puede hacer cualquier cosa que se pueda hacer con un script *CGI*, como procesar la información de formularios, generar páginas con contenidos dinámicos, o mandar y recibir cookies. Las cuatro grandes características de este lenguaje de programación son: velocidad, estabilidad, seguridad y simplicidad. Este lenguaje presenta una gran documentación y excelente comunidad.

Este lenguaje interpretado y multiplataforma de propósito general es ampliamente usado, se encuentra instalado en más de 20 millones de sitios *Web* y más de un millón de servidores. El mismo permite la conexión a diferentes tipos de servidores de bases de datos tales como *MYSQL*, *PostgreSQL*, *Oracle*, *ODBC*, *DB2*, *Microsoft SQL Server*, *Firebird* y *SQLite*. De igual forma, posee soporte para la Programación Orientada a Objetos, soporte para *Unicode* (*PHP 6*), manejo de excepciones, etc.

1.9.12. Biblioteca de Javascript *JQuery*.

La biblioteca de *Javascript JQuery* es una liviana librería de *software* libre y de código abierto. La misma permite cambiar el contenido de una página *Web* sin necesidad de recargarla, mediante la manipulación del árbol *DOM* y peticiones *AJAX*. La utilización de esta librería permite lograr buenos resultados en poco tiempo y espacio. Las ventajas fundamentales de la utilización de esta librería son:

- Ahorra muchas líneas de código.

- Hace transparente el soporte de las aplicaciones para los navegadores principales.
- Provee un mecanismo para la captura de eventos.
- Provee un conjunto de funciones para animar el contenido de la página en forma muy sencilla.
- Integra funcionalidades para trabajar con *AJAX*.

1.9.13. Framework de desarrollo para *PHP KumbiaPHP*.

Es un *Framework* libre escrito en *PHP 5* inspirado en *Ruby-Rails / Python-Django*. Ello permite la separación de las reglas de negocio, lógica de aplicación y vistas de presentación de una aplicación *Web*. Además, el mismo posee otras herramientas y clases que ayudan a acortar el tiempo de desarrollo, se basa en las mejores prácticas de desarrollo *Web*, el mismo es usado en *software* comercial y educativo, *KumbiaPHP*, en este caso, fomenta la velocidad y eficiencia en la creación y mantenimiento de aplicaciones *Web*, reemplazando tareas de codificación repetitivas por poder, control y placer. Por otro lado, se implementan los mejores patrones de programación orientados a la *Web*. En este sentido, se constata el soporte para *AJAX*, componentes gráficos, etc. Los niveles de seguridad y rendimiento que implementa la versión 1.0 “*Spirit*” del *Framework* son novedosos e interesantes.

La idea fundamental del *Framework* es proporcionar facilidades para construir aplicaciones robustas, apoyado en su flexibilidad y fácil configuración. [16]

1.10. Conclusiones.

En el capítulo abordado se ofrecen consideraciones en torno a un sistema, el cual, una vez aplicado, en la práctica que se determine, permite el análisis de código estático *PHP* para la detección de vulnerabilidades. Los conceptos y ejemplos presentados en torno a la seguridad de las aplicaciones *PHP* evidencian la importancia que reviste el tema que se presenta en este sentido. De igual forma, la propuesta de desarrollo del sistema demuestra un marcado uso en la utilización de herramientas de reconocida popularidad y bajo patentes libres, requisitos indispensables en el sistema que se propone.

CAPÍTULO II

CARACTERÍSTICAS DEL SISTEMA

Este capítulo se enfoca en el objeto de estudio de la aplicación, se enfatiza en los procesos involucrados en el campo de acción, y también se realiza un análisis de la ejecución del mismo. De igual forma, se describen las características del sistema, concebidas previamente luego de una modelación del entorno en que se desarrolla el mismo. Se presentan los requisitos funcionales y no funcionales, indispensables en el proceso de desarrollo del *software*.

2.1. Objeto de Estudio.

2.1.1. Problema y situación problemática.

La Universidad de las Ciencias Informáticas es un centro productivo de *software* de gran importancia por los resultados exhibidos en los últimos tiempos en la creación de programas informáticos. En la actualidad, a los proyectos productivos desarrollados en el lenguaje *PHP* no se les realizan pruebas de calidad en cuanto a la seguridad del código escrito en los mismos. Esta dificultad repercute de forma negativa en la calidad del producto que brinda la universidad y, por tanto, en el prestigio de este centro de altos estudios, pues la seguridad de las aplicaciones, sobre todo las del tipo *Web*, es un requisito indispensable y exigido por los clientes, los que lamentan los efectos dañinos que ocasionan las malas prácticas de escritura de código.

Los usuarios de la universidad se ven necesitados de un sistema que permita la revisión del código *PHP* que programan. Estos usuarios recurren a alternativas como la descarga de programas que sirvan para este fin, los que en algunos casos son propietarios y su uso es restringido o la generación de los informes es bastante engorrosa, además de ser difícil la ejecución del análisis con los mismos.

El enfoque estadístico sobre el conocimiento de los problemas de generación de código en *PHP* no es posible en estos momentos, pues no existe la forma de computar resultados en este sentido.

2.1.2. Objeto de automatización.

Las ideas expresadas anteriormente permiten constatar la necesidad de desarrollar un sistema capaz de detectar las vulnerabilidades del código fuente en aplicaciones o *scripts* desarrollados en el lenguaje de programación *PHP* de forma flexible y amigable mediante una interfaz *Web*, el cual oriente al usuario en los problemas que presenta su código, así como, tener persistentes los datos relacionados con las vulnerabilidades del código escrito en la universidad con el objetivo de trazar pautas en la formación de los estudiantes y profesores en este sentido.

2.1.3. Información que se maneja.

Se manipula la información referente a los proyectos y *scripts* de los usuarios interesados en la revisión de código fuente en *PHP*. De igual forma, se manejan reportes de los análisis realizados mediante la herramienta de revisión de código estático. Estos reportes presentan los elementos y estadísticas que se asocian al análisis que realizan los usuarios. Se constata en los mismos de igual forma, un resumen, en el que su resumen final se basa en la metodología de valoración de riesgos *OWASP*. En este sentido se parte de la fórmula:

$$\text{Riesgo} = \text{Probabilidad de ocurrencia} * \text{Impacto}$$

En el [anexo #7](#) se presenta la modificación de esta fórmula adaptada al ambiente de análisis de *Pixy*.

2.2. Propuesta del Sistema.

Se propone la realización de un sistema *Web* que permita a todo usuario de la Universidad de las Ciencias Informáticas la revisión de código realizado en el lenguaje *PHP*. Este sistema debe brindar la posibilidad de analizar tanto un proyecto como un sencillo *script*, además de brindar reportes de forma detallada y en

un formato entendible y elegante al usuario. El sistema, de igual forma, debe posibilitar la generación de reportes estadísticos en sentido general de los resultados de los análisis realizados.

2.3. Modelo de Dominio.

El negocio que se aborda tiene un bajo nivel de estructuración, no se definen concretamente los procesos del mismo, la misma se focaliza en la mayoría de los casos, en el uso de herramientas y tecnologías informáticas, es por ello que se decide dar un nuevo enfoque a todo el proceso, con lo que se utiliza un Modelo del Dominio, que permite, de manera visual, mostrar los principales conceptos u objetos del mundo real que se manejan en el dominio de la aplicación en desarrollo. Este modelo contribuye posteriormente a identificar algunas clases que se utilizarán en el sistema, además de que permite entender el contexto en que se enmarca el sistema.

A continuación se presenta el modelo de dominio del sistema:

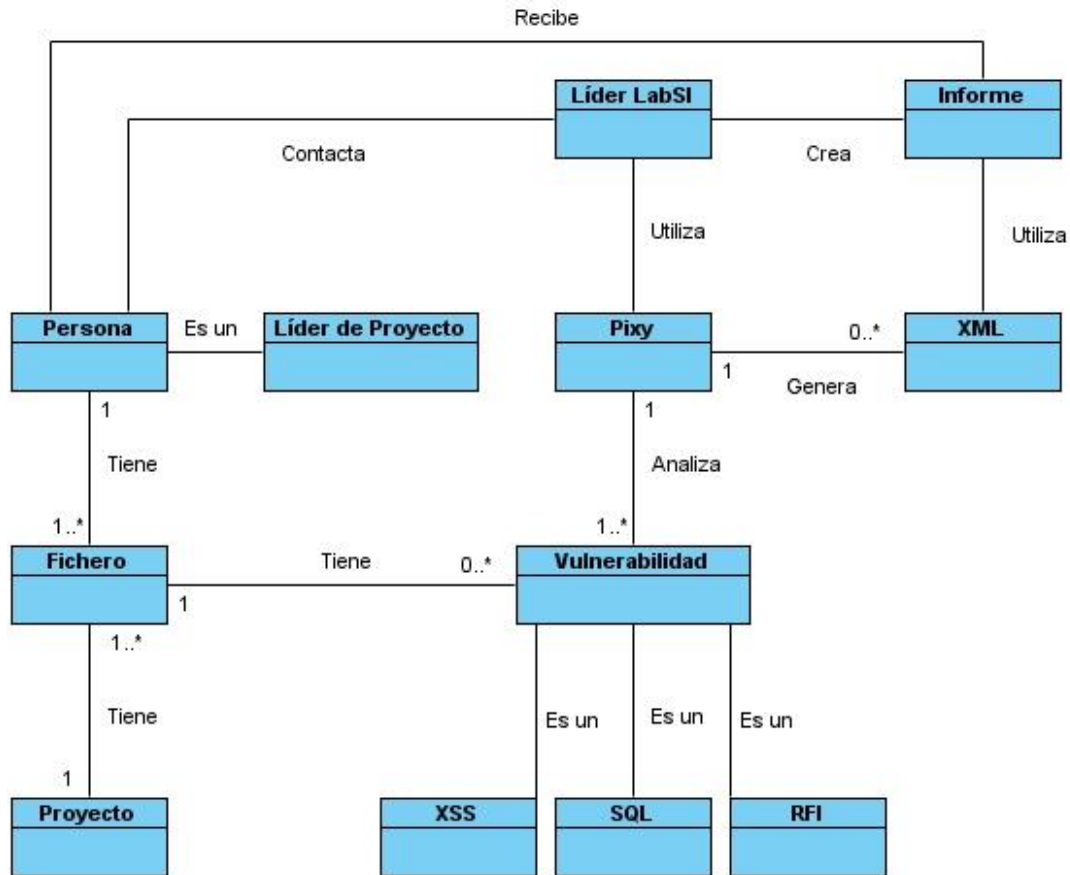


Fig. 2 Modelo de Dominio

2.3.1 Definición de las clases del modelo del dominio.

La definición de cada uno de los conceptos del modelo de dominio se presenta a continuación:

- Persona: estudiante o líder de proyecto que necesita un análisis de calidad en cuanto a seguridad de código PHP.
- Líder de proyecto: es el responsable mayor de un proyecto productivo determinado.

- Líder LabSI: especialista del Laboratorio de Seguridad Informática que realiza análisis de vulnerabilidades.
- Fichero: representa un archivo que posee código PHP, el cual será analizado y puede contener determinadas vulnerabilidades.
- Proyecto: representa un conjunto de ficheros PHP que serán analizados, se refiere en esencia a proyectos productivos.
- Pixy: analizador de código estático para la detección de vulnerabilidades.
- Vulnerabilidad: representa un problema de seguridad en el código que se analiza.
- XSS: representa el análisis de vulnerabilidades del tipo XSS.
- SQL: representa el análisis de vulnerabilidades del tipo SQL injections.
- RFI: representa el análisis de vulnerabilidades del tipo inyección de código remoto.
- XML: archivo generado por el analizador Pixy con la descripción de todas las vulnerabilidades analizadas.
- Reporte: informe que se realiza por el especialista del Laboratorio de Seguridad Informática luego del análisis de ficheros *XML* generados por *Pixy*.

2.4. Especificación de los requisitos de Software.

2.4.1. Requisitos funcionales.

Los requisitos funcionales especifican comportamientos particulares de un sistema, son características requeridas del sistema que expresan una capacidad de acción del mismo. A continuación se presenta un listado de los mismos en el sistema que se modela:

R1. Subir fuente.

1.1. Subir fichero.

1.2. Subir código.

R2. Generar reporte.

2.1. Analizar fichero.

2.1.1. Mostrar opciones de análisis.

2.1.2. Establecer funciones *sinks*.

R3. Cancelar análisis en curso.

R4. Mostrar reportes.

R5. Eliminar reporte.

R6. Descargar reporte.

R7. Mostrar estadísticas.

7.1. Permitir seleccionar criterio estadístico.

7.1.1. Listado de vulnerabilidades asociadas a proyecto.

7.1.2. Listado de vulnerabilidades asociadas a usuario.

7.1.3. Listado general de vulnerabilidades en *PHP*.

7.1.4. Listado de archivos analizados por el usuario.

2.4.2. Requisitos no funcionales.

Los requisitos no funcionales expresan exigencias de cualidades que deben tener los sistemas. Éstos representan características de la aplicación que señalan restricciones en la misma y hacen al producto atractivo, usable, rápido o confiable. En la mayoría de los casos, los requisitos no funcionales influyen en el éxito de todo producto informático. A continuación se presentan los requisitos no funcionales del sistema:

- *Apariencia o interfaz externa.*
 - La interfaz del sistema debe ser agradable y fácil de utilizar, de forma que no se necesite mucha práctica para la manipulación de la misma.
 - Debido al gran uso que presupone la aplicación, es necesario que se combinen correctamente los colores, tipo de letra y tamaño.
 - La interfaz debe brindar una ayuda del uso de la misma al usuario.
- *Usabilidad.*
 - La aplicación debe brindar la posibilidad al usuario de realizar el análisis de acuerdo a sus necesidades como tipo de vulnerabilidades a analizar, características del análisis, etc.
 - Todas las opciones brindadas deben ser fácilmente entendibles.
 - La manipulación del sistema podrá ser usado por toda persona con conocimientos básicos de ambientes *Web*.
- *Rendimiento.*
 - La aplicación requiere de un procesamiento elevado de información debido al análisis que se realiza por parte de la herramienta de análisis estático, la misma sin embargo, tiene niveles de

Capítulo II. Características del Sistema

desempeño aceptables debido al algoritmo de búsqueda de vulnerabilidades que implementa. El tiempo de respuesta del sistema está en dependencia de la cantidad de archivos a analizar, el promedio para la respuesta al usuario de un archivo es de aproximadamente dos segundos, aunque este valor puede variar en dependencia de la cantidad de ficheros a analizar. Los clientes no deben necesitar muchos recursos para ejecutar la aplicación.

- *Soporte.*
 - La aplicación debe ser extensible y bien documentada para facilitar el soporte de la misma en caso que sea necesario.
 - Los elementos necesarios para el despliegue del sistema (configuraciones y elementos a instalar) pueden ser consultados en el documento de despliegue del Sistema de Análisis Estático de Vulnerabilidades SAEV.

Configuración para el cliente:

- Se debe tener instalado un navegador Web con JavaScript habilitado y *Acrobat Reader* en cualquier versión.
- *Seguridad.*
 - El acceso al sistema será controlado por usuario y contraseña, mediante el dominio *UCI*.
 - Los administradores son los únicos que tienen acceso a la parte administrativa del sistema.
 - Los reportes son personales, ningún usuario está autorizado a ver otros reportes.
 - La navegación por el sistema y la comunicación con el servidor *FTP* es mediante el protocolo de capa de conexión segura *SSL*.
- *Confiabilidad.*

Capítulo II. Características del Sistema

- Los reportes deben ser resguardados por la importancia que representan.
- Debe garantizarse las salvadas de seguridad de la Base de Datos para la recuperación en caso de fallas.
- *Aspectos legales.*
 - La aplicación y la documentación de la misma, pertenecen a la Universidad de las Ciencias Informáticas y sólo a ésta se le permite su uso.
 - Las herramientas de desarrollo son libres, por tanto, las licencias están avaladas.
- *Diseño e Implementación.*
 - Los lenguajes de programación a utilizar son *Java* por parte del analizador, *PHP* como lenguaje del lado del servidor y *JavaScript* del lado del cliente.
 - El *framework* de desarrollo para *PHP* que se requiere es *KumbiaPHP 1.0 stable*.
 - La biblioteca de *JavaScript* es *JQuery 1.4*.
 - El estilo arquitectónico es el *MVC* para la separación de los datos, interfaz de usuario y lógica de negocio.
- *Hardware.*

Servidores:

- Se requiere un servidor *Web* que tenga como mínimo 1 GB de memoria RAM, procesador *Pentium IV* ó superior y 5 GB de disco duro disponibles.
- Se requiere un servidor de Base de Datos y otro de *FTP*, los que deben disponer de un disco duro

de 160 GB de capacidad como mínimo.

- Se requiere tarjeta de red.

Cientes:

- Se requiere tarjeta de red.
- Computadora con al menos 256 MB de memoria RAM, al menos 100 MB de disco duro, procesador *Pentium* III ó superior.
- *Software.*
 - Se requiere la instalación de *Linux*, específicamente Ubuntu Server 9.10.
 - El servidor *Web* es *Apache* 2.0 con soporte para *SSL* tanto para la navegación como la comunicación con el servidor *FTP*.
 - El servidor de Base de Datos es *PostgreSQL* 8.3 ó superior.
 - El servidor *FTP* *VSFTPD* 2.2.2.
 - Se requiere igualmente la instalación de la máquina virtual de Java, versión 1.6.0 ó superior.
 - El lenguaje de programación es *PHP*, versión 5.2.
 - Se debe desplegar el directorio raíz del *Framework KumbiaPHP*, versión 1.0 estable.
 - La versión del analizador de código estático *Pixy* a utilizar es la 3.03 estable con las modificaciones que se realizaron.
 - Las computadoras clientes deben tener instalado navegador *Internet Explorer*, versión 7 ó superior, *Mozilla Firefox* versión 2.0.0.1 ó superior, *Safari* 2.0 ó superior. De igual forma, se necesita *Acrobat*

Reader para la lectura de reportes.

2.5. Modelo de casos de uso del sistema.

El Modelo de Casos de Uso del Sistema permite que los desarrolladores de software y los clientes lleguen a un acuerdo sobre los requisitos, es decir sobre las condiciones y posibilidades que debe cumplir el sistema. Además proporciona la entrada fundamental para el análisis, diseño y las pruebas. [17]

2.5.1. Definición de los actores del sistema a automatizar.

Actor del Sistema	Justificación
Usuario	Persona encargada de solicitar el análisis de ficheros o proyectos realizados en <i>PHP</i> y recibir los resultados en los informes correspondientes.

Tabla 1 Actores del Sistema

2.5.2. Diagrama de casos de uso del sistema a automatizar.

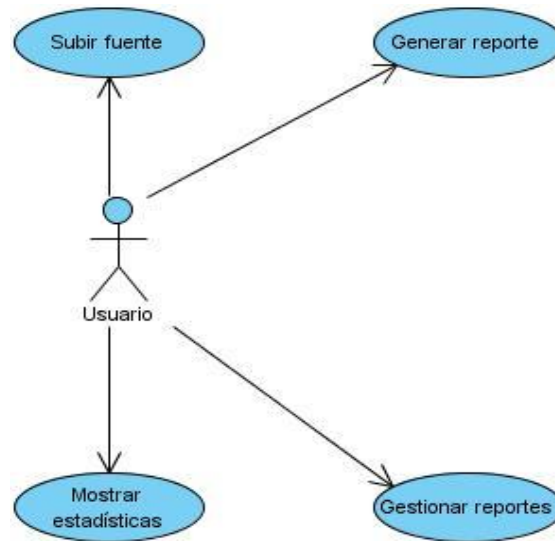


Fig. 3 Diagrama de Casos de Uso del Sistema

2.5.3. Descripción de los casos de uso.

En el [Anexo #1](#) se muestran las descripciones textuales de los casos de uso presentados.

2.6. Conclusiones.

En el capítulo abordado se ha efectuado un análisis ingenieril de las características del sistema que se presenta, lo cual contribuye a la documentación y correcta implementación de la aplicación. Se realizó la definición de los requisitos, tanto funcionales, como no funcionales y la representación visual de las funcionalidades a través del Diagrama de Casos de Uso del Sistema. En este orden, además del cumplimiento de los requisitos expresados, es posible el comienzo de la construcción del sistema que se presenta.

CAPÍTULO III

ANÁLISIS Y DISEÑO DEL SISTEMA

El presente capítulo está enfocado en el análisis y diseño de la aplicación mediante la metodología de desarrollo *RUP*. Se establece una mejor comprensión de los requisitos funcionales del sistema y se proporciona una visión general del mismo, lo que permite la entrada segura en la fase de implementación. Se presentan para este fin, modelos que permiten determinar con precisión lo que se debe programar.

3.1. Análisis.

El análisis consiste en obtener una visión del sistema, se preocupa de ver qué hace, de modo que sólo se interesa por comprender los requisitos funcionales con que cuenta el *software* para así estructurar el proyecto de forma clara y precisa. En este sentido, se profundiza en el dominio de la aplicación, lo que permite una mayor comprensión del problema para modelar la solución.

3.1.1. Modelo de Análisis.

El Modelo de Análisis ofrece la primera representación técnica del sistema. Se identifican las clases que describen la realización de los Casos de Uso, atributos y relaciones entre ellas, lo cual permite la realización de los diagramas de clases del Análisis. De esta forma, se logra una comprensión más exacta de los requisitos del sistema, se describe lo que requiere el cliente y se establece una aproximación a lo que sería el Modelo de Diseño.

3.1.2. Diagramas de clases del Análisis.

Capítulo III. Análisis y Diseño del Sistema

Los diagramas de clases del Análisis se centran en los requisitos funcionales y son evidentes en el dominio del problema, debido a que representan conceptos y relaciones del dominio. A continuación se presentan los tres estereotipos de clases estandarizados en UML que se utilizan para ayudar a los desarrolladores a distinguir el ámbito de las diferentes clases:

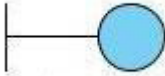

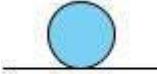
Nombre	Características	Figura
Interfaz	Modelan la interacción entre el sistema y sus actores.	 Interfaz
Control	Coordinan la realización de uno ó unos pocos Casos de Uso, coordinan las actividades de los objetos que implementan la funcionalidad del Caso de Uso.	 Control
Entidad	Modelan información que posee larga vida y que es a menudo persistente.	 Entidad

Tabla 2 Estereotipos de clases UML

A continuación se presentan los diagramas de clases del Análisis (*DCA*) para cada Caso de Uso del sistema:

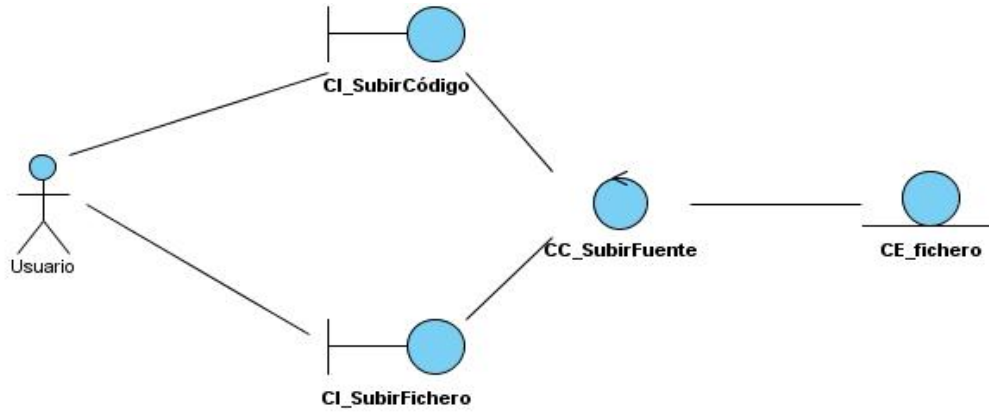


Fig. 4 DCA Subir Fuente

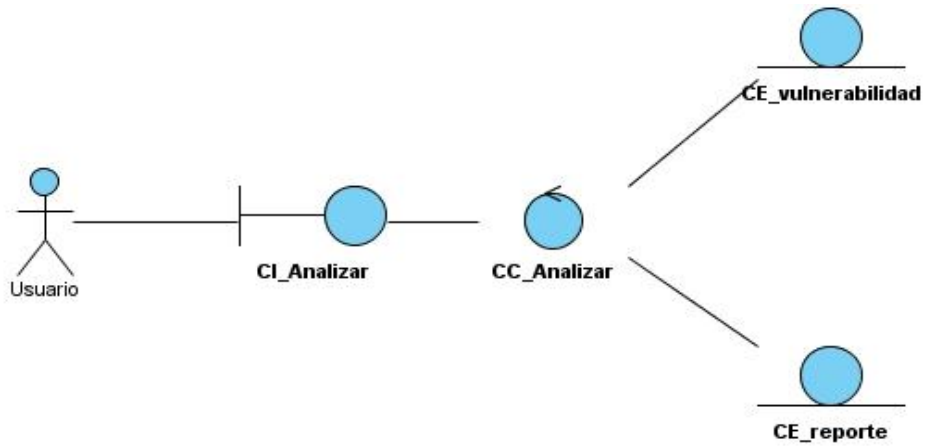


Fig. 5 DCA Generar Reporte

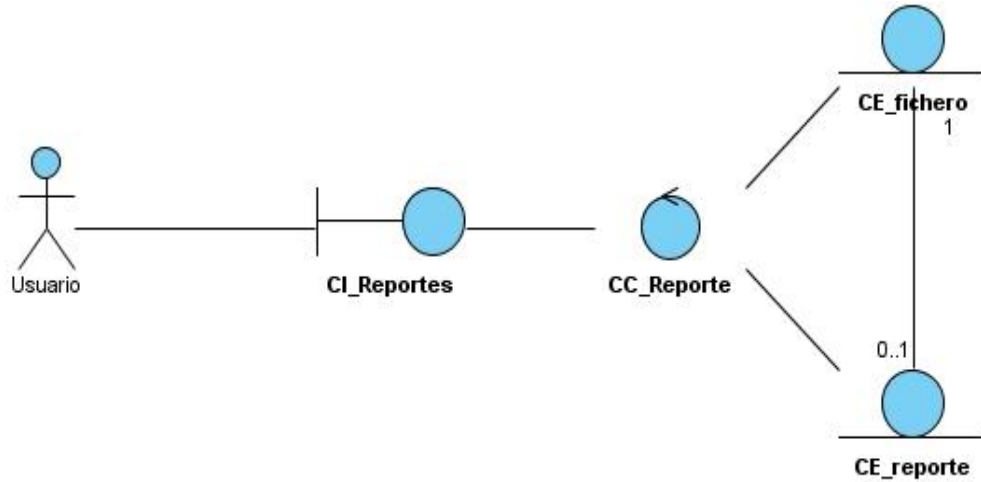


Fig. 6 DCA Gestionar Reportes

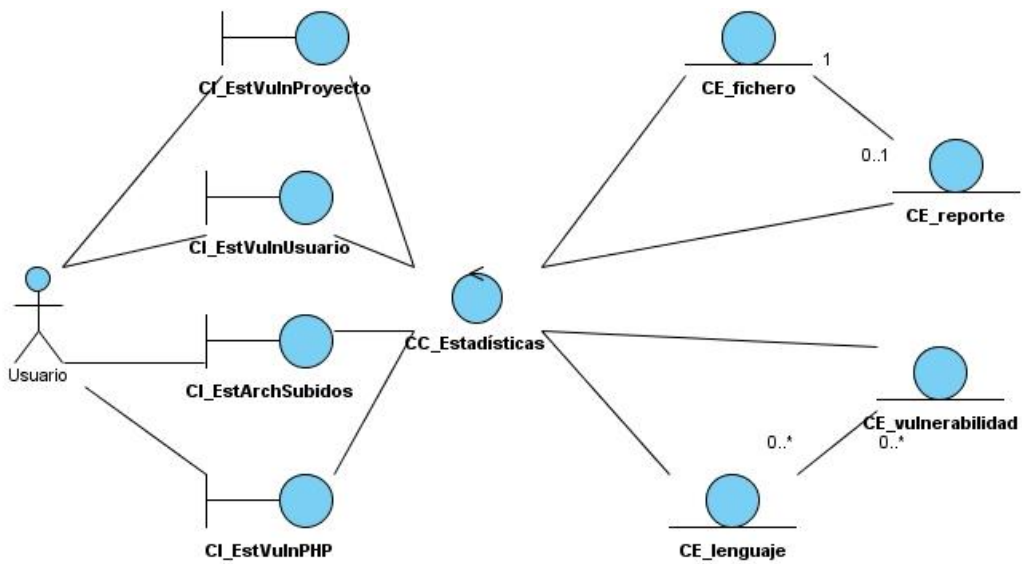


Fig. 7 DCA Mostrar Estadísticas

3.2. Diseño del Sistema.

El Diseño representa el refinamiento del análisis, se tienen en cuenta los requisitos no funcionales, es decir, cómo debe cumplir el sistema sus objetivos. El mismo tiene el propósito de formular los modelos para preparar la entrada a las actividades de implementación y pruebas del sistema. Un buen diseño del sistema permite que el mismo pueda ser implementado sin ambigüedades. El resultado más importante de esta etapa es el Modelo de Diseño.

“En el diseño modelamos el sistema y encontramos su forma (incluida la arquitectura) para que soporte todos los requisitos- incluyendo los requisitos no funcionales y otras restricciones que se le suponen.” [18].

3.2.1. Patrones que se utilizan.

Un patrón de diseño es una solución a un problema de diseño. Se consideran como procedimientos para solucionar diversos problemas del mismo tipo y son la base para la búsqueda de soluciones a dificultades comunes en el desarrollo de *software*. *“El patrón es una pareja de problema/solución con un nombre y que es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en situaciones nuevas.” [19]*

En el diseño de la propuesta de solución del módulo, se tiene en cuenta el patrón arquitectónico: Modelo – Vista - Controlador, que implementa *KumbiaPHP*, con lo cual se separan los datos de la aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos:

- **Modelo:** encapsula los datos y las funcionalidades. Es independiente de cualquier representación de salida y/o comportamiento de entrada. Se encarga de la abstracción de la lógica relacionada con los datos, lo que permite que la vista y las acciones sean independientes.
- **Vista:** muestra la información al usuario en forma de página *Web* y le permite interactuar con ella.
- **Controlador:** se encarga de procesar las interacciones del usuario y realiza los cambios necesarios en el modelo o en la vista. Mantiene aislado al modelo y a la vista de los detalles del protocolo utilizado para las peticiones.

En el desarrollo del módulo, se aplicaron además, patrones de asignación de responsabilidades (*GRASP*) y patrones *Gang of Four (GOF)*, con lo cual se aprovechan las bondades que brinda el *framework* de desarrollo. Estos patrones se presentan a continuación:

Patrones GRASP:

Los patrones *GRASP* describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Los mismos constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable.

El uso de estos patrones en el sistema se describe a continuación:

Experto: el uso de este patrón permite a los objetos valerse de su propia información para hacer lo que se les pide, favorece la existencia de mínimas relaciones entre las clases, lo que permite contar con un sistema sólido y fácil de mantener. En el caso de *KumbiaPHP*, las clases controladoras manejan las peticiones del cliente, y las clases del modelo son las encargadas del acceso a datos, pues contienen y representan los datos que manejará el sistema, lo que permite que se conserve el encapsulamiento y se dé soporte a un bajo acoplamiento y una alta cohesión.

Controlador: se considera para efectuar las asignaciones en cuanto al manejo de los eventos del sistema y definir sus operaciones. *KumbiaPHP* contribuye a la utilización de este patrón debido a que define un Controlador Frontal (*Dispatcher*) que implica que todas las solicitudes son dirigidas a un único script *PHP* que se encarga de instanciar al controlador frontal y redirigir las llamadas.

Creador: se considera para la asignación de responsabilidades a las clases relacionadas con la creación de objetos, de forma tal que una instancia de un objeto sólo pueda ser creada por el objeto que contiene la información necesaria para ello. El uso de este patrón admite crear las dependencias mínimas necesarias entre las clases, lo que beneficia el mantenimiento del sistema y se brindan oportunidades de reutilización. En el *framework KumbiaPHP* existe un único script *PHP* que se encarga de instanciar al controlador frontal (*Dispatcher*), este último es el encargado de instanciar las clases controladoras y éstas, a su vez,

instancian objetos del modelo que extienden de la clase *ActiveRecord*. El uso de este patrón permite crear las dependencias mínimas necesarias entre las clases, lo que favorece al mantenimiento del sistema.

Alta cohesión: es el responsable de asignar una responsabilidad de forma tal que la cohesión siga siendo alta, este patrón caracteriza a las clases que están estrechamente relacionadas y consiste en colaborar con otros objetos para compartir el esfuerzo si la tarea a realizar es grande.

Bajo acoplamiento: garantiza que exista una alta reutilización entre las funcionalidades de las clases y una escasa dependencia, lo que contribuye al mantenimiento de las mismas. El uso de los patrones Experto y Creador favorecen al bajo acoplamiento entre las clases del sistema. Este patrón se tiene en cuenta por la importancia que tiene realizar un diseño de clases independientes que puedan soportar los cambios de una manera fácil y permitan la reutilización.

Patrones GOF.

Abstract Factory: este patrón proporciona una interfaz para crear familias de objetos relacionados o dependientes sin especificar su clase concreta. El mismo permite configurar en tiempo de ejecución un sistema con una familia u otra de objetos. Además garantiza que un conjunto de clases se usen a la vez. Cuando el *framework* necesita, por ejemplo, crear un nuevo objeto, busca en la definición de la factoría el nombre de la clase que se debe utilizar para esta tarea.

Singleton: este patrón garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. El acceso a la Base de Datos en el sistema se realiza mediante una única instancia para evitar que se creen múltiples conexiones innecesarias.

Virtual Proxy: este patrón se utiliza en objetos de la clase *Auth*, cuando no es necesario instanciarlos a no ser que sea solicitado por el usuario o se cumplan determinadas condiciones.

Acceso a datos:

ActiveRecord: este patrón representa de forma Orientada a Objetos los datos de una Base de Datos Relacional, se definen interfaces sencillas para acceder y manipular esos datos. Es un enfoque al problema de acceder a los datos de una base de datos. Los modelos en *KumbiaPHP* heredan de una clase denominada *ActiveRecord*.

Otros patrones:

Template View: este patrón permite proporcionar flexibilidad en la escritura de código, el mismo brinda la opción de utilizar *tags* personalizados o marcas embebidas en el contenido dinámico de la aplicación.

FrontController: este patrón permite que todas las peticiones a la aplicación sean atendidas inicialmente por él y luego sean enrutadas a controladores de usuario y acciones que atienden cada una. Este patrón se utiliza en unión al *Dispatcher* de *KumbiaPHP*.

3.2.2. Funcionamiento general de una aplicación en *KumbiaPHP 1.0 Spirit*.

En *KumbiaPHP* los proyectos se dividen en sub-aplicaciones y éstas a su vez en diferentes módulos. Una aplicación en sentido general tiene la siguiente estructura:

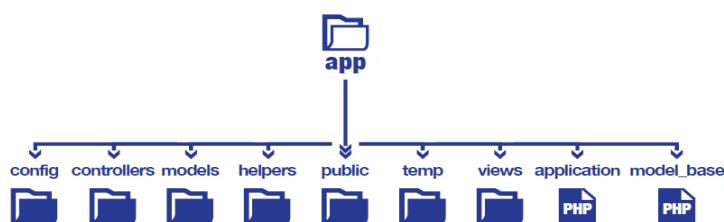


Fig. 8 Estructura común de una aplicación *KumbiaPHP*

En la carpeta *controllers* se encuentran todos los controladores de la aplicación, en *views* las vistas de la aplicación y en *models*, los modelos que contienen la lógica de la aplicación. El resto de las carpetas permiten organizar la aplicación, donde es posible definir carpetas para archivos *JavaScript*, imágenes,

archivos de configuración, etc. Para la definición de módulos sólo es necesaria la creación de la carpeta correspondiente en *controllers* y *views*.

La conexión que se establece entre los controladores que se crean y el *framework* se representa a continuación:

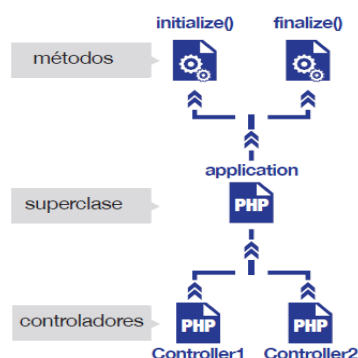


Fig. 9 Funcionamiento de clases controladoras en *KumbiaPHP*

El método `initialize()` se ejecuta antes de llamar cualquier controlador. Igualmente, `finalize()` se ejecuta después que se llama al controlador. Todos los controladores heredan de una superclase llamada `applicationController` (`application.php`), que contiene métodos disponibles en todos los controladores de una aplicación.

En el caso de los modelos se establece la siguiente relación:

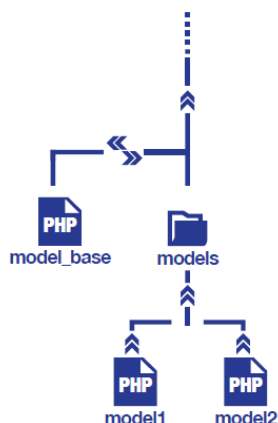


Fig. 10 Funcionamiento de los modelos en *KumbiaPHP*

En este sentido, las clases modelos se relacionan con la superclase *ActiveRecord* (*model_base.php*). En esta clase existe una gran cantidad de métodos en los que se puede apoyar para construir la parte lógica de la aplicación.

El *framework KumbiaPHP* posee un sistema de presentación basado en Vistas (*Views*) que es el tercer componente del sistema Modelo Vista Controlador (*MVC*), el *framework* permite, mediante el uso de plantillas, la reutilización de código para que no sea repetido. Las vistas deberían contener una cantidad mínima de código en *PHP* para que fuese suficientemente entendible por un diseñador *Web* y además, para dejar a las vistas sólo las tareas de visualizar los resultados generados por los controladores y presentar las capturas de datos para usuarios.

3.2.3. Arquitectura del sistema.

La arquitectura se desarrolló sobre el patrón arquitectónico Modelo Vista Controlador. Ésta da comienzo cuando un usuario realiza una petición segura al servidor, la cual es capturada por el fichero *index.php* que es el que se encarga de inicializar el núcleo donde están presentes un conjunto de funcionalidades y librerías que se utilizan en toda la aplicación, este archivo, de igual forma, instancia el enrutador para que dirija la petición al controlador que la gestionará. El componente Filtros de Seguridad es el encargado de

filtrar los datos, autenticar y tener un control de acceso de los usuarios. La capa Controlador es la que atiende, responde y enruta las acciones solicitadas por el usuario, constituye además todo lo referente a cálculos de entrada de datos y cómo se presentará la información en la capa de vista. En la capa modelo se encapsulan los datos y funcionalidades, ésta interactúa con la Base de Datos a través de la capa de abstracción *ORM Active Record*. Por último, la capa vista permite la interacción entre el sistema y el usuario, para finalmente retornar una respuesta. Los conceptos expresados anteriormente se presentan gráficamente a continuación:

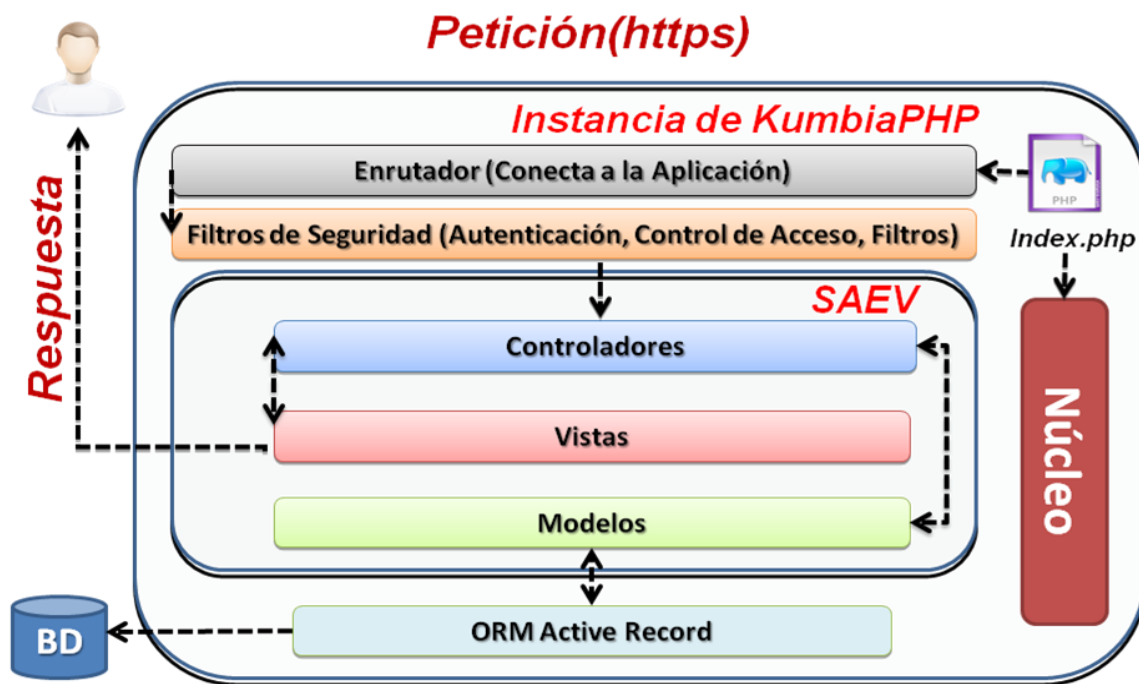


Fig. 11 Arquitectura del sistema



3.2.4. Diagramas de Interacción.

Los diagramas de interacción representan una vista dinámica del sistema y constituyen la secuencia de acciones que ocurren cuando el actor comienza el caso de uso, así como los mensajes que se envían entre cada una de las clases. Los mismos se pueden clasificar en: diagramas de colaboración y

diagramas de secuencia. En el caso especial de los diagramas de secuencia, se destaca la ordenación temporal de los mensajes. En el [anexo #2](#) se presentan los diagramas de secuencia de cada uno de los Casos de Uso que se proponen.

3.2.5. Diagramas de clases del Diseño.

Los diagramas de clases del Diseño describen gráficamente las especificaciones de las clases del *software* y contienen las clases, atributos, métodos, navegabilidad y dependencias existentes entre ellas. La forma habitual de modelar estos diagramas no es factible cuando lo que se desea diseñar es una aplicación *Web*, como es el caso, por tanto se utiliza una extensión de *UML* para *Web*, que se ajusta a la arquitectura de este tipo de sistemas. Los estereotipos de clases fundamentales utilizados en este sentido son:

Estereotipo	Características	Figura
<i>Server page</i>	Modelan páginas <i>Web</i> con scripts ejecutados por el servidor. Los scripts interactúan con recursos del servidor como bases de datos, lógica del negocio, sistemas externos, entre otros. En <i>KumbiaPHP</i> , las <i>Server Page</i> representan páginas con extensión <i>.php</i> , las que tienen el nombre <i>nombreclaseController.php</i> y controlan las peticiones que se realizan al servidor y con clases que se destinan a lógica de negocio.	
<i>Client page</i>	Una instancia de <i>Client Page</i> es una página <i>Web</i> con formato <i>HTML</i> que es mostrada por un	


	<p>navegador o <i>browser</i>. Representa la mezcla de datos, presentación y lógica. En el <i>framework KumbiaPHP</i>, las <i>Client Page</i> se representan por páginas que tienen la extensión <i>phtml</i> y están asociadas a páginas servidoras. Si se tiene una página controladora <i>ejemploController.php</i> que tiene el método <i>Adicionar()</i>, debe existir para el mismo, en la carpeta del <i>framework Views</i>, otra carpeta con el nombre <i>Ejemplo</i> y dentro de ésta un archivo llamado <i>adicionar.phtml</i>.</p>	
<i>Form</i>	<p>Representa una colección de campos de entrada de datos, es parte de una <i>Client Page</i>. En este caso, las páginas que se muestran al cliente tienen el formato <i>HTML</i>, cuando se realiza un <i>submit</i>, los datos son procesados por páginas servidoras.</p>	

Tabla 3 Estereotipos Web de UML

De igual forma, los estereotipos para las relaciones entre clases son:

Estereotipo	Descripción
<code><<link>></code>	<p>Representa un apuntador desde una página cliente hacia otra de este mismo tipo o página servidora.</p>

<code><<submit>></code>	Esta relación siempre ocurre entre formularios y una página servidora, esta última procesa los datos procedentes de los formularios que envían información.
<code><<build>></code>	Sirve para identificar cuál(es) página(s) servidora(s) son responsables de la creación de una página cliente. Una página servidora puede crear varias páginas clientes, pero una página cliente sólo puede ser creada por una sola página servidora. Esta relación siempre es unidireccional.
<code><<include>></code>	Asociación direccional desde una página servidora a otra página servidora o página cliente. Esta asociación indica que las páginas incluidas se procesan mientras que la página se ensambla.
<code><<redirect>></code>	Representa una relación unidireccional que indica que una página <i>Web</i> redirige a otra.

Tabla 4 Estereotipos de relaciones entre clases

El módulo que se desarrolla depende de un sistema que se compone de diferentes módulos. Cada módulo posee una página inicio que redirecciona a las diferentes funcionalidades que se presentan. En el caso del módulo PHP, existe una página servidora (*indexPHP.php*), la cual construye la página cliente *indexPHP.phtml*. A continuación se presentan los diagramas de clases del Diseño (DCD) del Sistema de Análisis Estático de Vulnerabilidades, módulo *PHP*:

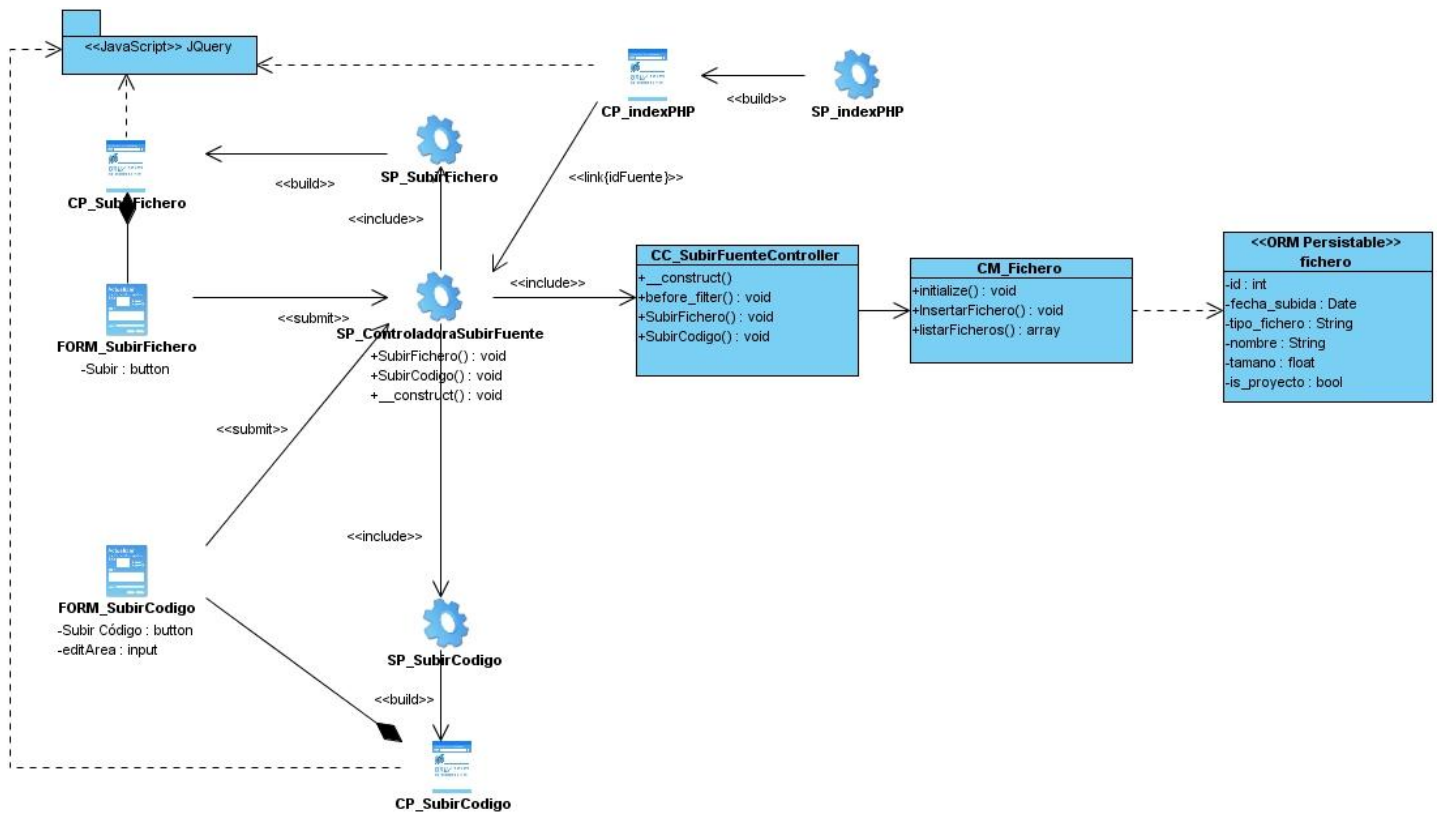


Fig. 12 DCD Subir Fuente

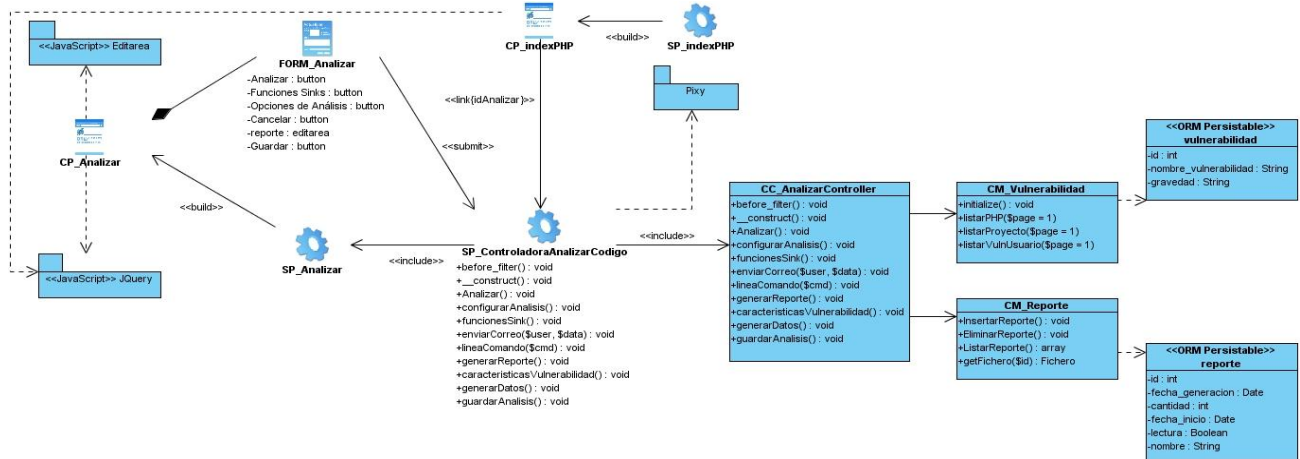


Fig. 13 DCD Generar Reporte

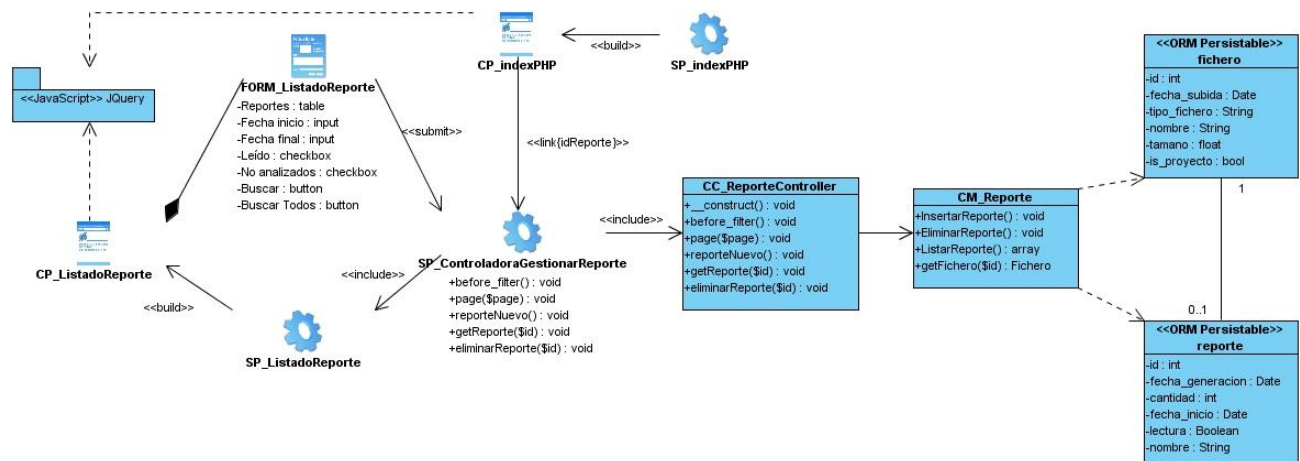


Fig. 14 DCD Gestionar Reportes

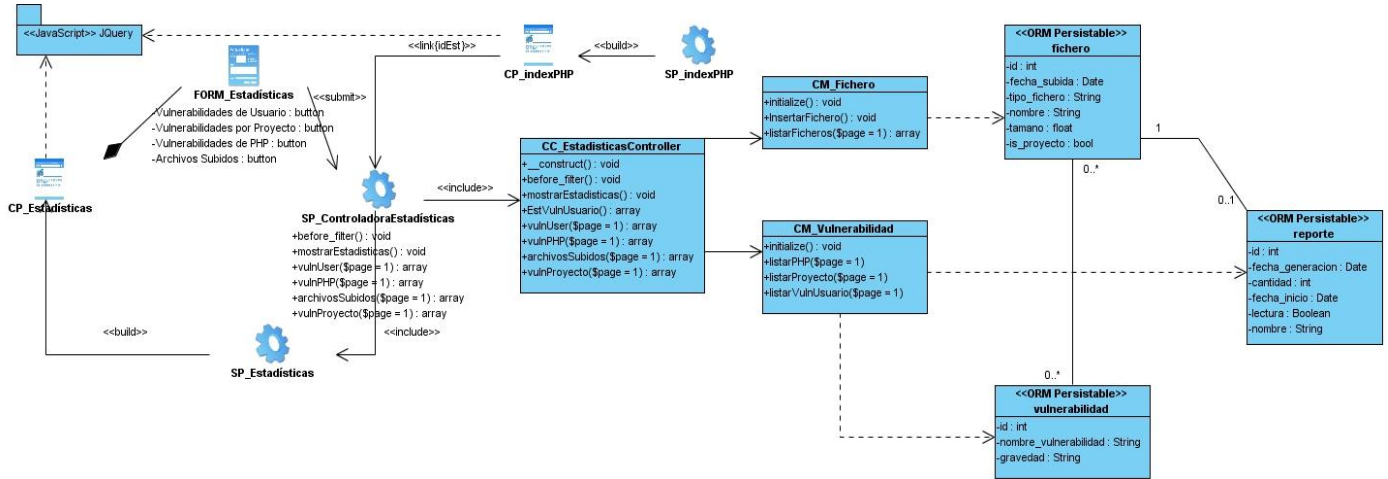


Fig. 15 DCD Mostrar Estadísticas

3.3. Modelo de Datos.

El Modelo de Datos describe las representaciones lógicas y físicas de datos persistentes utilizados por una aplicación. [20]

3.3.1. Modelo lógico de datos (diagrama de clases persistentes).

El diagrama de clases persistentes representa los objetos que se deben almacenar en la Base de Datos y que son necesarios para que la información pueda persistir aún cuando el usuario abandona la sesión. Las clases persistentes, por lo general, tienen como origen las clases entidades del análisis, debido a que éstas son las encargadas de modelar la información y el comportamiento asociado a algún fenómeno o concepto, como una persona, un objeto del mundo real o un suceso. En la **figura #16** se muestra el diagrama de clases persistentes.

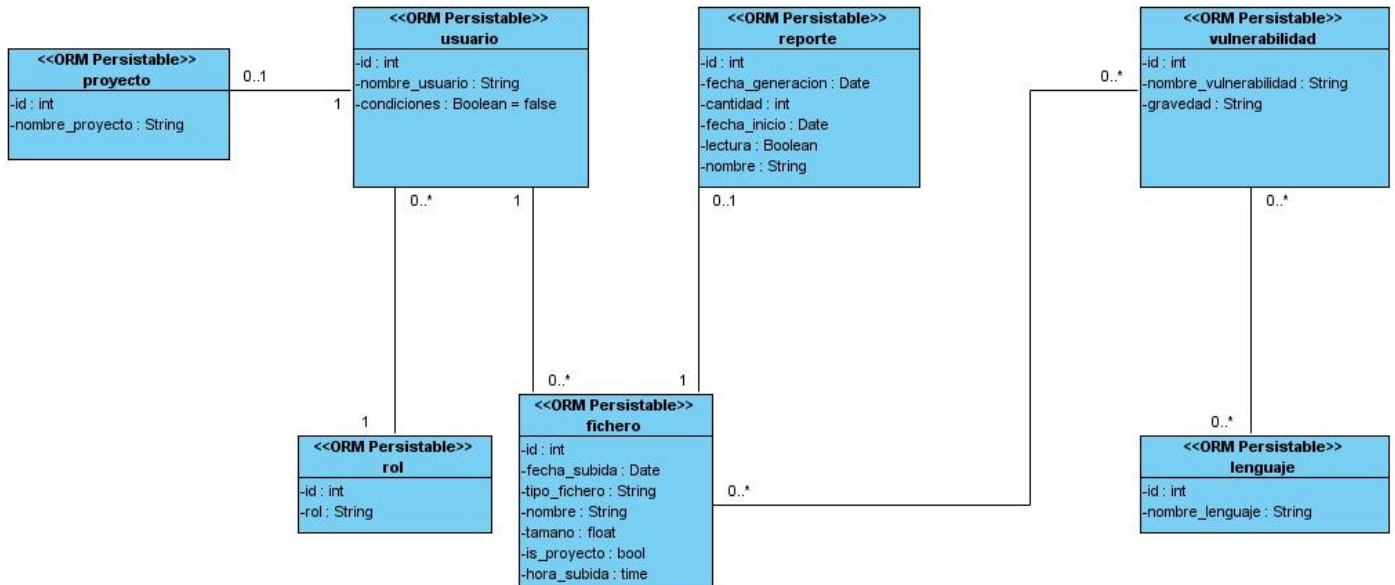


Fig. 16 Diagrama de clases persistentes

3.3.2. Modelo físico de datos (Modelo de Datos).

El Modelo de Datos representa la base conceptual para diseñar aplicaciones que hacen un uso intensivo de datos, así como la base formal para las herramientas y técnicas empleadas en el desarrollo y uso de sistemas de información, es decir, un Modelo de Datos es la estructura o representación física de las tablas de la Base de Datos. El modelo físico de datos se desarrolló a partir de la base del conjunto de clases persistentes y sus asociaciones en el Modelo de Diseño. En la **figura #17** se presenta el Modelo de Datos.

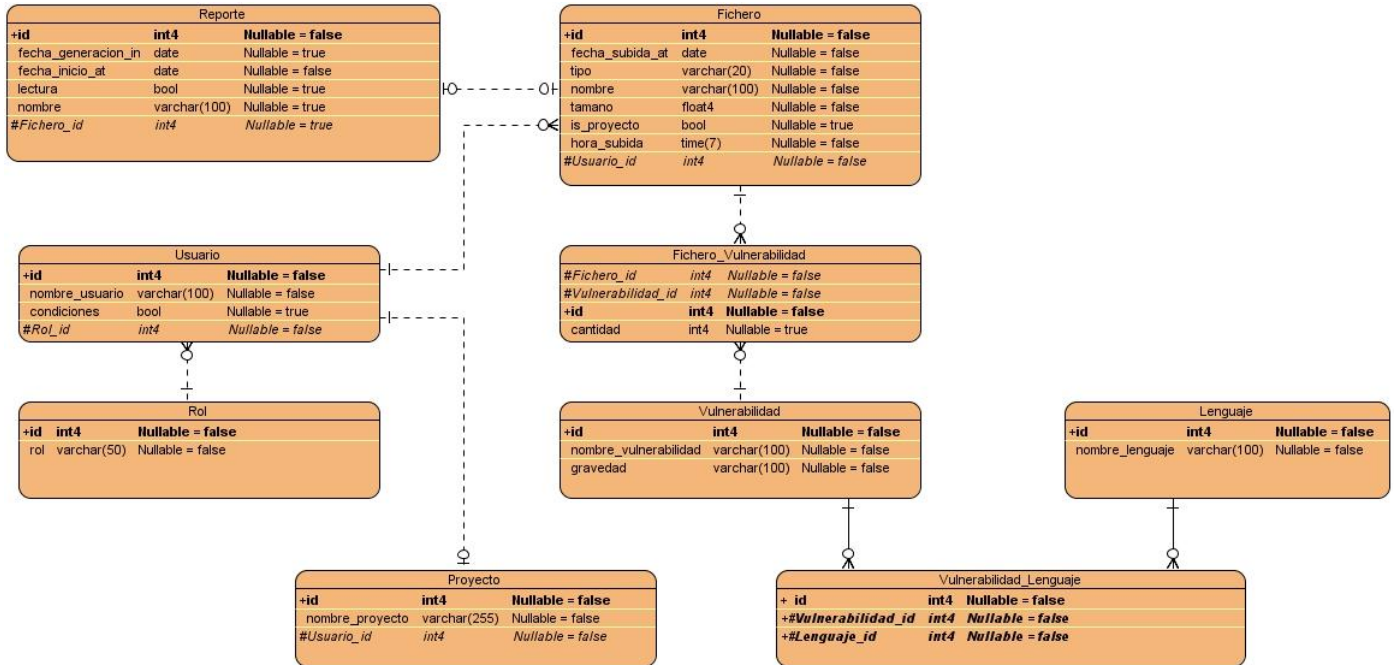


Fig. 17 Modelo de Datos

3.4. Conclusiones.

En este capítulo se ha presentado un estudio relacionado con el Análisis y Diseño de la aplicación que sirven de base fundamental en la implementación del sistema. Se ha logrado modelar todos los procesos que han sido objeto de estudio durante el transcurso de la investigación, lo que proporciona una idea completa de lo que realmente es el *software* que se presenta. Los diagramas y especificaciones de diseño que se proponen constituyen una guía que puede ser fácilmente comprendida por los desarrolladores con el objetivo de implementar la aplicación que se ha diseñado.

CAPÍTULO IV

IMPLEMENTACIÓN DEL SISTEMA

El presente capítulo tiene como objetivo desarrollar los artefactos correspondientes a la implementación del sistema, el cual tiene como entrada principal el Modelo de Diseño que se analiza en el capítulo anterior. Se presentan los diagramas de componentes y despliegue respectivamente, con lo que se conforma el Modelo de Implementación. De igual forma, se presentan estándares de codificación necesarios en la implementación del sistema.

4.1 Diagrama de Despliegue.

El diagrama de Despliegue se utiliza para describir la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo. Se muestra la configuración de los nodos de procesamiento en tiempo de ejecución y la comunicación entre ellos. A continuación se presenta el diagrama de Despliegue del Sistema de Análisis Estático de Vulnerabilidades.

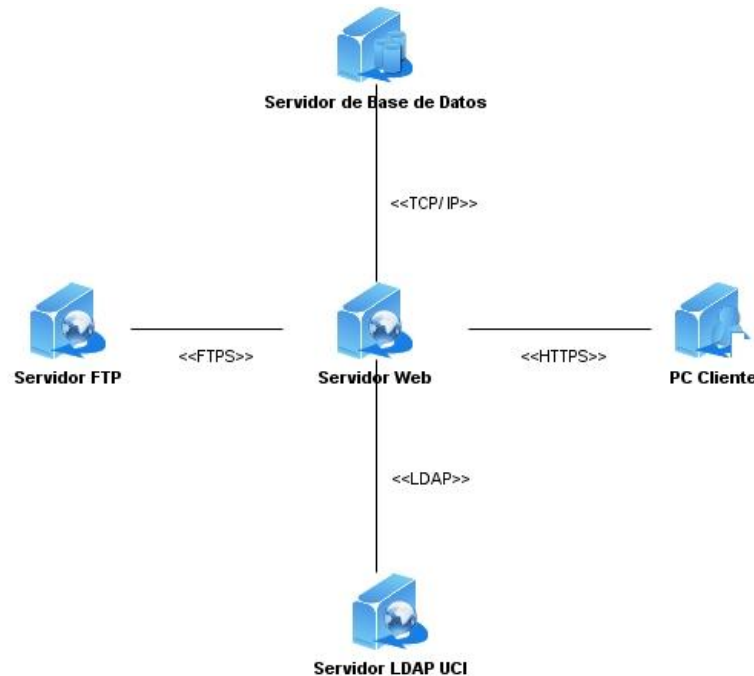


Fig. 18 Diagrama de Despliegue

La explicación de los nodos de este diagrama se presenta a continuación:

- Nodo Servidor Web: es el encargado del procesamiento de las peticiones de los usuarios. Se utiliza el servidor *Web Apache*. De igual forma, este nodo incluye al analizador de código estático *Pixy*.
- Nodo Servidor LDAP UCI: este nodo representa la Base de Datos de los usuarios *UCI*, en donde se facilitan tareas como autenticación, obtención de nombres, etc.
- Nodo Servidor de Base de Datos: en este nodo se encuentra la Base de Datos del sistema, se cuenta para este objetivo con el gestor *PostgreSQL*.

- Nodo Servidor FTP: en este sentido se presenta el servidor de *FTP VSFTPD*, para el almacenamiento de reportes.
- Nodo PC Cliente: este nodo modela a los clientes que se conectan al servidor *Web*, este proceso se realiza mediante conexión segura *https*.

4.2 Estándares de codificación.

El desarrollo de una aplicación informática exige que se adopten estándares de codificación y estilo. El uso de estándares asegura la legibilidad del código entre distintos desarrolladores, permite la guía para el mantenimiento o actualización del sistema, además de que facilita la portabilidad entre plataformas y aplicaciones. En este sentido, el *framework* de desarrollo *KumbiaPHP* requiere determinadas medidas de codificación para una estandarización de las aplicaciones. A continuación se presentan los estándares que se aplican en el desarrollo del sistema:

- **Estándar 1:** las estructuras de control deben tener un espacio entre el *keyword* de la estructura y el signo de apertura de paréntesis para distinguir entre las llamadas de las funciones, el signo de llaves debe estar sobre la línea de la estructura.
- **Estándar 2:** siempre se debe utilizar las etiquetas `<?php ?>` para abrir un bloque de código. No se debe utilizar el método de etiquetas cortas (`<? y ?>` o `<% y %>`) porque esto depende de las directivas de configuración en el archivo *PHP.INI* y hace que el *script* no sea tan portable.
- **Estándar 3:** los nombres de las clases controladores deben seguir la convención *Nombrefichero_controller*.
- **Estándar 4:** los archivos con código *PHP*, deben de ser guardados en formato *ASCII* utilizando la codificación *ISO-8859-1*. Este formato permite que no existan incompatibilidades entre diferentes editores de texto, en especial *Dreamweaver*, el cual agrega códigos de caracteres extraños de

salto de línea, lo que puede ocasionar que el intérprete de *PHP* encuentre problemas a la hora de leer el *script*.

4.3 Diagrama de Componentes.

Los diagramas de componentes se utilizan para modelar la vista estática de un sistema, muestran las organizaciones y dependencias lógicas entre componentes de *software* que formarán un sistema, sean éstos librerías, componentes de código fuente, binarios o ejecutables.

Los usos más comunes de los diagramas de componentes se encuentran en:

- Modelación de código fuente.
- Modelación de versiones ejecutables y bibliotecas.
- Modelación de la Base de Datos física.

A continuación se presentan los diagramas de componentes de acceso a datos, código fuente y componentes *Web*:

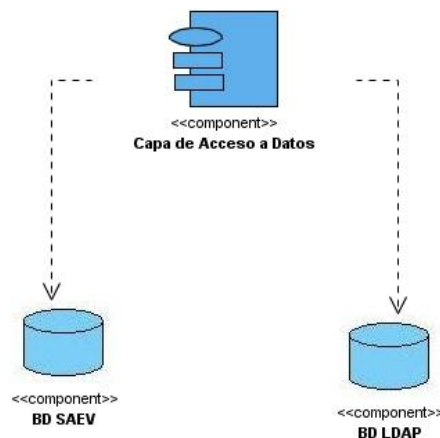


Fig. 19 Diagrama de Componentes, Base de Datos

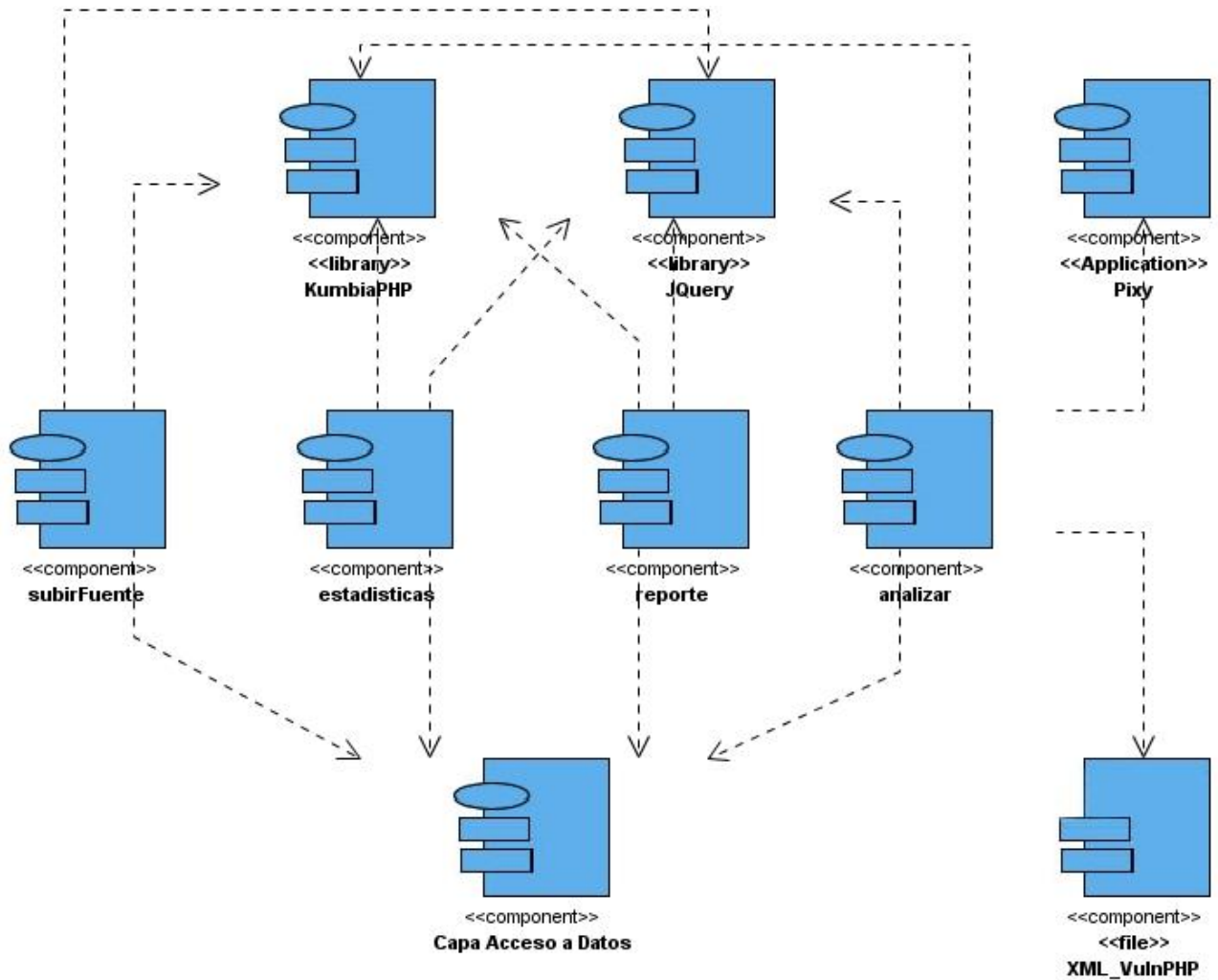


Fig. 20 Diagrama de Componentes, código fuente

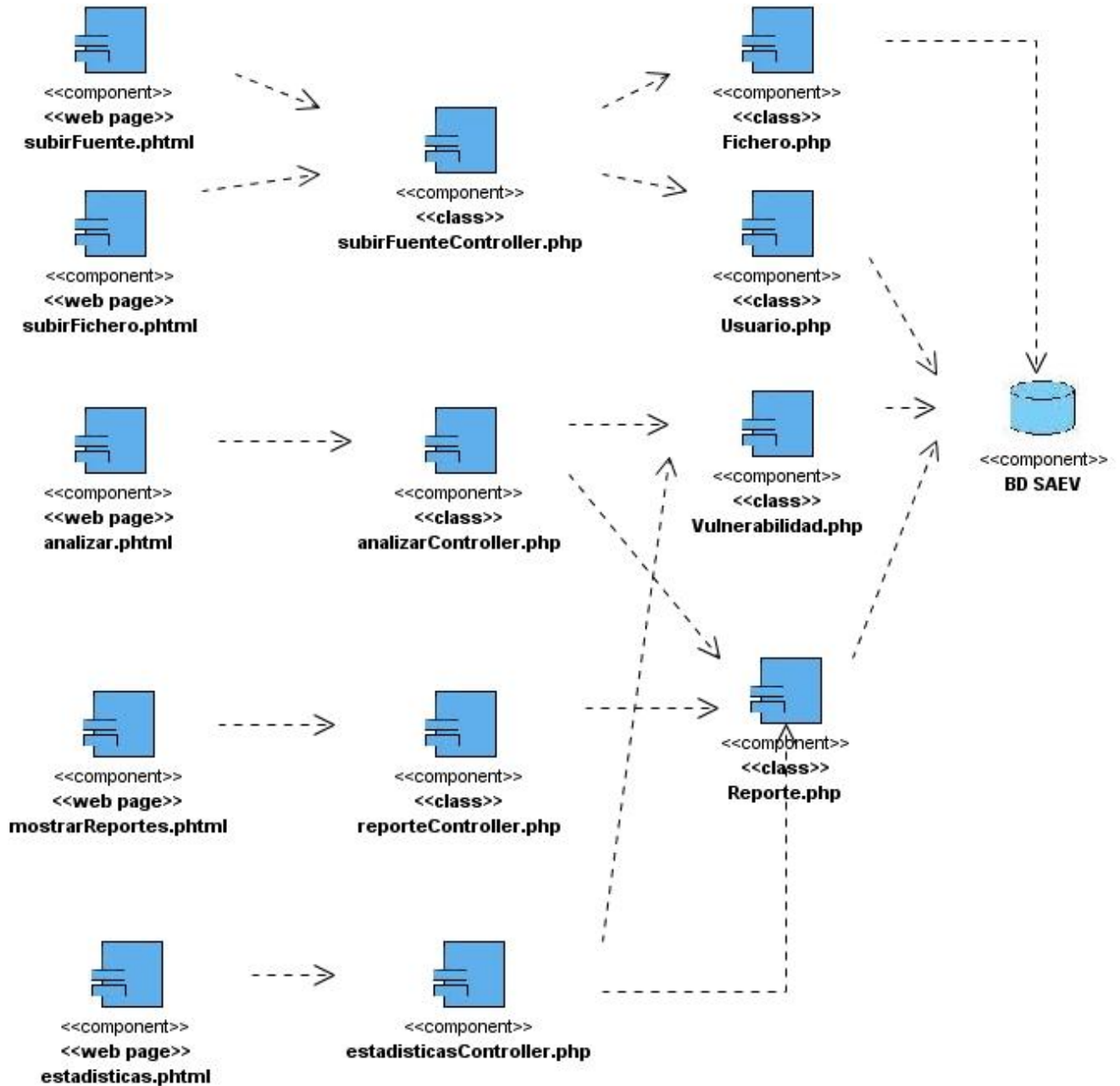


Fig. 21 Diagrama de componentes, componentes Web o código ejecutable

4.4 Conclusiones.

En el capítulo presentado se mostró la relación entre los principales componentes del sistema, cómo las clases y los objetos se organizan en términos de componentes, de igual forma, se constató la distribución del sistema mediante el Diagrama de Despliegue, así como los estándares de codificación que se utilizan. Se culmina de esta forma la implementación de la aplicación, con lo que se da cumplimiento al objetivo general trazado inicialmente.

CAPÍTULO V

MODIFICACIONES AL ANALIZADOR DE CÓDIGO ESTÁTICO *PIXY*

Este capítulo se enfoca en torno a las modificaciones efectuadas en el analizador de código estático *Pixy*. Se presentan elementos relacionados a los cambios en la gramática del analizador, así como, nuevas características en cuanto a reportes en formato *XML*, búsqueda de nuevas vulnerabilidades (ataque de inyección de código) y funciones actualizadas para *PHP 5* en el analizador. De igual forma, se presentan una serie de recomendaciones para el funcionamiento correcto de *Pixy*.

5.1. Modificaciones en el parser de *Pixy* *PHPParser*.

El analizador de código estático *Pixy* realiza el análisis del código fuente que se le proporciona mediante la técnica de análisis de flujo de datos. Toda entrada de código *PHP* es escaneado mediante *JFLEX* y luego parseado con *Java Cup*. Este parser utiliza el método de análisis ascendente *LALR*. Este proceso finaliza con la construcción de un árbol de parseo, el cual se transforma al código intermedio *TAC* (*Three Address Code*) para un mejor funcionamiento en el análisis de flujo de datos.

El analizador que se estudia posee algunas limitaciones en la gramática que analiza, las cuales están enfocadas principalmente en problemas del no reconocimiento de estructuras sintácticas de *PHP 5*. Cuando el analizador encuentra un problema de sintaxis en el código que analiza, automáticamente detiene el análisis, lo que provoca que existan vulnerabilidades en el código que no puedan ser analizadas. Para que sea posible el reconocimiento sintáctico de nuevas estructuras, se hace preciso modificar el escáner y generador de gramática *JFLEX* y *JCUP* respectivamente. De igual forma, se debe trabajar con el analizador en torno a la conversión del árbol de parseo a *TAC* y se puedan buscar las vulnerabilidades finalmente, mediante este código intermedio.

Capítulo V. Modificaciones al analizador de código estático Pixy

En el [anexo #4](#) se presentan las reglas agregadas en *JFLEX* y *CUP* para la modelación de los bloques de instrucciones. A continuación se presentan los cambios más importantes que se realizaron al *parser* de *Pixy*:

- *Bloque try...catch.*

El cambio que se presenta está enfocado en la detección y captura de excepciones en *PHP 5*. La estructura general de este bloque es la siguiente:

```
try{  
  
    //bloque instrucciones  
  
}  
  
catch (T_STRING $var){  
  
    //bloque instrucciones  
  
}  
  
// catch opcionales
```

- *Sentencia throw.*

En este sentido, esta sentencia permite el lanzamiento de excepciones, que son capturadas por el bloque *try...catch*.

La estructura de esta sentencia es la siguiente:

- **throw** new *Exception*("texto mensaje");
- **throw** new *T_STRING*();

Capítulo V. Modificaciones al analizador de código estático Pixy

- **throw** *\$var*;
 - Clases en PHP.

A partir de la versión 5 de *PHP* existe soporte por parte del lenguaje para la Programación Orientada a Objetos (*POO*). El analizador que se estudia tiene soporte hasta la versión 4.3.10 de *PHP*, en la cual existen algunos elementos de la *POO* como son las definiciones de clases. Los cambios para el reconocimiento sintáctico son:

- Visibilidad de atributos en clases.

En este sentido se refiere a la visibilidad (***public***, ***private***, ***protected***) que pueden tomar los atributos en clases.

- Visibilidad de funciones.

Las funciones también pueden tener visibilidad en *PHP 5*.

- Palabras reservadas delante de *class*.

En *PHP 5* las clases pueden ser declaradas como abstractas (***abstract class*** *NombreClase*) o se puede establecer que no se pueda heredar de ellas (***final class*** *NombreClase*).

- Métodos abstractos.

En este sentido, se define como posible valor sintáctico:

abstract visibilidad ***function*** *NombreFuncion*();

- Métodos *final*.

De la misma forma que sucede en las clases *final*, los métodos pueden ser definidos de la misma forma.

Capítulo V. Modificaciones al analizador de código estático Pixy

```
Visibilidad final function miMetodoNoHeredable() {  
  
    //Código del método  
  
}
```

- Constantes de clases.

En este sentido se permite la definición de constantes en clases.

```
class MyClass {  
  
    const MY_CONSTANT = "Constante de Clase";  
  
}
```

- Interfaces.

En *PHP 5* las interfaces permiten las declaraciones de funcionalidades que tienen que cubrir las clases que las implementan.

```
Interface interfaz {  
  
    visibilidad function encender();  
  
}
```

- Elementos estáticos.

Los elementos estáticos representan otra de las características de *PHP 5*.

- o Métodos estáticos.

```
visibilidad static function method(){
```

Capítulo V. Modificaciones al analizador de código estático Pixy

```
//bloque method
```

```
}
```

- Atributos estáticos.

```
visibilidad static $atributo;
```

- Acceso a métodos y atributos estáticos.

```
NombreClase::method($parameters);
```

```
NombreClase::attribute;
```

- Clonado de objetos.

En *PHP 5* se puede definir la clonación de objetos mediante la palabra reservada *clone*.

```
$copia_objeto = clone $objeto;
```

- Instancias de objetos:

```
if ($objeto instanceof Clase){
```

```
//cuerpo
```

```
}
```

- Análisis de vulnerabilidades en código *PHP* embebido en *HTML*.

El analizador de código estático *Pixy*, no analiza vulnerabilidades en código *PHP* que esté mezclado en *HTML*. El logro de este objetivo es posible mediante la modificación de escáner *JFLEX*.

5.2. Generación de reportes en formato *XML*.

Capítulo V. Modificaciones al analizador de código estático Pixy

Pixy muestra el resultado del análisis que realiza en la consola de comandos. Esta característica del analizador representa un problema debido a que la comunicación con otros sistemas que lo utilicen se torna complejo.

Teniendo en cuenta las posibilidades de expansión que brinda el analizador, se hace posible la generación de los reportes vía archivos *XML*. En el [anexo #3](#) se presenta el diagrama de clases con el soporte para *XML* incorporado.

5.3. Análisis de vulnerabilidades del tipo inyección de código.

El analizador de código estático *Pixy* sólo analiza vulnerabilidades del tipo XSS e inyecciones SQL. Las aplicaciones *Web* y en especial las que se realizan en *PHP* son objeto de las amenazas provocadas por la inyección de código en sentido general. Un analizador de código estático para *PHP* requiere el análisis de este tipo de vulnerabilidad, de gran impacto en la actualidad.

La flexibilidad que presenta *Pixy* en el procesamiento de “datos manchados” permite la modelación de la inyección de código.

En el [anexo #3](#) se presentan las clases necesarias para la modelación de la inyección de código maligno.

5.4. Nuevas funciones de repercusión en la seguridad de aplicaciones *PHP* en *Pixy*.

Pixy clasifica las funciones de *PHP* en 5 categorías, ellas son:

- Fuertes sanitizadoras: representan las funciones que cuando son utilizadas son capaces de “limpiar” cualquier “dato manchado” que esté en una variable. Un ejemplo de este tipo de función lo representa *htmlentities()*.
- Débiles sanitizadoras: representan las funciones que pueden o no, “limpiar” los datos que poseen las variables donde se utilizan. Un ejemplo de este tipo de funciones lo representa *addslashes()*.

Capítulo V. Modificaciones al analizador de código estático Pixy

- Multidependencia: representan las funciones en las que determinados parámetros “manchados” son expandidos a otros parámetros, los que se imprimen posteriormente. Un ejemplo de este tipo de funciones lo representa la función `str_replace($search, $replace, $subject)`, la cual busca mediante el parámetro `$search` en `$subject` y reemplaza con `$replace`. En esta función, si cualquier parámetro es “manchado”, el valor de retorno también lo será.
- Multidependencia inversa: representan las funciones que pueden tomar un número arbitrario de argumentos. Si alguno de estos parámetros se torna “manchado”, la salida también podría serlo. Un ejemplo de este tipo de funciones lo representa `max()`, la cual acepta cualquier cantidad de argumentos.
- Maligna: modelan las funciones que siempre retornan valores malignos a través de sus parámetros. Un ejemplo de este tipo de funciones es `urldecode()`, la cual puede convertir un valor previamente “limpio” en peligroso.

El analizador *Pixy* modela también las llamadas *sinks* o sumideros. Estas funciones representan los puntos de un programa *PHP* que imprimen o evalúan valores, ya sean procedentes de variables o directamente. Un ejemplo es la función `echo()`, encargada de imprimir una cadena o variable pasada por sus argumentos.

La modelación de todos los tipos de funciones expresados con anterioridad en *Pixy* presenta como dificultad que está limitado a pocas funciones, no se tienen en cuenta elementos de *PHP 5*. Las nuevas funciones que se agregaron son el resultado de un estudio del efecto de las mismas en aplicaciones *PHP*. En el [anexo #5](#) se presentan las nuevas funciones de repercusión para la seguridad de *PHP* con sus respectivas clasificaciones.

5.5. Otras modificaciones.

Capítulo V. Modificaciones al analizador de código estático Pixy

Los cambios expresados anteriormente tienen un impacto significativo en el funcionamiento del analizador *Pixy*. Existen otras modificaciones cuyo objetivo es la optimización de *Pixy* en su vinculación a una aplicación *Web*, las mismas están enfocadas en:

- *Generación de grafos al formato .jpg*: *Pixy* por defecto genera los grafos en el formato *.dot*, lo cual requiere una conversión manual de este archivo a otro formato como *.jpg*, *.gif*, *.pdf*, etc., lo que permite una visualización más humana de los mismos. La incorporación de la generación automática de los grafos a *.jpg* permite un mejor rendimiento en la conexión de *Pixy* a un sistema externo.
- *Cambios en la generación de grafos*: el analizador por defecto genera para todo tipo de vulnerabilidad un grafo asociado a la misma para un mejor entendimiento del origen de la mala práctica de programación. Cuando existen vulnerabilidades simples que dependen de una sola sentencia en todo el fichero (ejemplo, *echo \$a*), *Pixy* genera un solo nodo en el grafo que presenta. En este tipo de vulnerabilidades no es necesario visualizar el grafo asociado a la vulnerabilidad, debido a que se torna trivial el análisis de la misma. El rendimiento de la aplicación desde el punto de vista de velocidad de ejecución mejora en este sentido si no se incluye la generación de este tipo de grafos. De igual forma, *Pixy* elimina todos los grafos que están contenidos en la carpeta por defecto del analizador para este fin, cada vez que se analiza un fichero. Esto representa un problema debido a que si se requiere el análisis de diferentes archivos, se produce la pérdida de la información contenida en la carpeta *graphs*, excepto para el último reporte.
- *Eliminación de análisis de ficheros incluidos*: en este sentido, cuando *Pixy* por defecto en el análisis de vulnerabilidades encuentra algún archivo incluido y éste posee alguna vulnerabilidad, se muestra el resultado de la búsqueda de este fichero aún sin ser invocado el mismo. En un entorno donde se analicen proyectos completos, esto puede representar un problema debido a la duplicación en el análisis de ficheros.

Capítulo V. Modificaciones al analizador de código estático Pixy

- *Cambios en nombres de archivos y grafos:* cuando Pixy genera reportes y grafos de los análisis realizados y detecta que éstos tienen nombres similares a ficheros generados anteriormente, se produce la sobrescritura de estos últimos, lo cual representa un problema por la pérdida de información. Debido a esta dificultad, los nombres de ficheros y grafos se generan con las modificaciones que se realizaron con el siguiente formato: **milisegundos_tipo_nombrefichero**, donde *milisegundos* representa los milisegundos en que se genera el fichero, *tipo* es el tipo de análisis realizado (*XSS*, *RFI*, *SQL*) y *nombrefichero* es el nombre del fichero que se analiza.
- *Reconocimiento de funciones pertenecientes a la gramática de Pixy:* el analizador no reconoce funciones como *eval()*, debido a que es propia de la gramática de *PHP* y por ello no es posible agregarla en los ficheros de configuración de Pixy. Este tipo de funciones, si se usa de forma incorrecta puede provocar grandes problemas de seguridad como la ejecución de funciones que invocan comandos del sistema.
- *Captura de errores sintácticos para posterior procesamiento:* la gramática de Pixy está encapsulada en archivos *.class* de Java. Cuando sucede un error sintáctico, por defecto no se puede capturar la línea donde ocurre la misma, lo que provoca problemas en la generación de reportes. Se hace necesaria la obtención de esta información para su presentación en ficheros *XML*.
- *Cambio en dirección de salida de reportes y grafos:* la generación de reportes y grafos en Pixy por defecto se produce en la misma carpeta del analizador, lo cual resulta engorroso y problemático en el momento de buscar los reportes, una salida personalizada ayuda en este sentido. La opción que permite esta nueva característica es *-x dirección*.
- *Cambio en dirección de salida de XML de archivos con errores sintácticos:* en este sentido, existen determinadas sentencias de código que Pixy no reconoce correctamente, por lo tanto, no se puede presentar al usuario estos problemas. Desde el punto de vista administrativo, la creación de una localización específica de los ficheros y reportes *XML* con errores sintácticos resulta importante

Capítulo V. Modificaciones al analizador de código estático Pixy

para la corrección de los mismos y que no se vuelvan a repetir esos errores en futuros análisis. La nueva opción que permite este cambio es *-e dirección*.

- *Cambio en nombre de direcciones en nodos de grafos*: los nodos de los grafos que se generan en *Pixy* presentan la dirección completa del archivo de la vulnerabilidad que se analiza. Esto puede representar un problema, por ejemplo, si un fichero vulnerable se encuentra en la dirección */var/www/proyecto/fichero.php*, el grafo asociado pondrá en cada nodo esta dirección, lo cual en términos de seguridad no es correcto, pues se presenta la configuración interna del servidor de la aplicación. Es por este motivo que a *Pixy* se le incorpora la opción de especificar desde qué carpeta mostrar las direcciones de los nodos de los ficheros vulnerables, esto es posible mediante la opción *-u dirección*.
- *Problema en generación doble de funciones*: el analizador presenta como defecto que procesa doble las funciones de cualquier tipo que encuentra, en el reporte que se genera en formato *XML* se imprimen dos *tags* por cada función que se encuentra. Esto representa un problema en cuanto a la optimización de la lectura de los archivos *XML*, además de que no sería factible el procesamiento estadístico en este sentido. El cambio en este sentido fue necesario para la optimización del sistema.

5.6. Recomendaciones para el correcto funcionamiento de *Pixy*.

El analizador de código estático *Pixy* exige el cumplimiento de determinadas reglas que mejoran su poder en la detección de vulnerabilidades y disminuyen la salida de falsos positivos e incluso falsos negativos. Las recomendaciones esbozadas a continuación permiten un mejor desempeño en la búsqueda de vulnerabilidades de *Pixy*, las mismas están adaptadas a la fusión entre el analizador y la interfaz *Web* con que se conecta, ellas son:

- En un mismo fichero no deben existir dos clases ó más con el mismo nombre.
- En un mismo fichero no deben existir dos métodos ó más con el mismo nombre.

Capítulo V. Modificaciones al analizador de código estático Pixy

- En un mismo fichero no es recomendable tener dos funciones ó más con el mismo nombre.
- El código a analizar no debe contener errores sintácticos, ni semánticos.
- Aunque no representa un error sintáctico y *Pixy* es capaz de detectarlo, es buena práctica utilizar las funciones con la cantidad de parámetros requeridos.
- El analizador tiene un mejor desempeño si no existen ficheros incluidos. En caso de que sea necesario su uso, es recomendable que no existan ambigüedades en la llamada a estos ficheros.
- Evitar llamadas recursivas infinitas como la siguiente, pues el analizador puede reportar alarmas:

```
function foo() {  
    foo();  
}
```

- Evitar el uso de la función *die()* como en el siguiente ejemplo:

```
function foo() {  
    echo 'Ejemplo';  
    die();  
}
```

- El analizador hasta el momento sólo analiza código de PHP puro hasta la versión 5, se recomienda no analizar códigos de *frameworks* del lenguaje, pues se pueden producir errores sintácticos.

5.7. Conclusiones.

Capítulo V. Modificaciones al analizador de código estático Pixy

En el capítulo presentado se ha realizado un análisis de las nuevas funcionalidades incorporadas en el analizador de código estático *Pixy*, las que permiten la ampliación del mismo en la detección de un nuevo grupo de vulnerabilidades y reconocimiento sintáctico de elementos de *PHP 5*, en especial lo referente a la Programación Orientada a Objetos. En este sentido, se analizan vulnerabilidades en sentencias propias de esta versión de *PHP* como es el caso del bloque *try...catch*. Las recomendaciones presentadas resultan de vital importancia en el desempeño del analizador en entornos productivos y deben ser tenidas en cuenta por todo usuario del sistema.

CAPÍTULO VI

FACTIBILIDAD DEL SISTEMA

El presente capítulo se enfoca en el cálculo de la factibilidad del sistema, la que permite conocer si resulta conveniente llevarlo a cabo o no. Para la realización de este análisis, se utiliza el método de Análisis de Puntos de Casos de Uso, que se hace efectivo para estimar el esfuerzo requerido en el desarrollo de los primeros casos de uso de un sistema si se sigue una aproximación iterativa como el Proceso Unificado de *Rational*, el cual coincide con la metodología de desarrollo aplicada al presente proyecto.

6.1. Método de estimación Puntos por Casos de Uso.

La estimación por Puntos de Casos de Uso es un método que permite estimar tiempo de desarrollo mediante la asignación de pesos a factores que influyen en su resultado final.

A continuación se detallan los pasos a seguir para la aplicación del método de estimación Puntos por Casos de Uso:

6.1.1. Cálculo de Puntos de Casos de Uso sin ajustar.

$$UUCP = UAW + UUCW$$

Donde:

UUCP: puntos de casos de uso sin ajustar.

UAW: factor de peso de los actores sin ajustar.

Capítulo VI. Factibilidad del Sistema

UUCW: factor de peso de los casos de uso sin ajustar.

Para calcular factor de peso de los actores sin ajustar **UAW:**

Tipo	Descripción	Peso	Cant * peso:
Simple	Otro sistema que interactúa con el sistema a desarrollar mediante una interfaz de programación. (API, Application Programming Interface).	1	0*1
Medio	Otro sistema que interactúa con el sistema a desarrollar mediante un protocolo o una interfaz basada en texto.	2	0*2
Complejo	Una persona que interactúa con el sistema mediante una interfaz gráfica.	3	1*3
Total:			3

Tabla 5 Cálculo del factor de peso de los actores sin ajustar

Para calcular el factor de peso de los casos de uso sin ajustar **UUCW:**

No.	Nombre de Caso de Uso	Cantidad transacciones	Tipo
1	Subir fuente.	3	Simple

Capítulo VI. Factibilidad del Sistema

2	Generar reporte.	2	Simple
3	Gestionar reportes.	4	Medio
4	Mostrar Estadísticas.	4	Medio

Tabla 6 Cantidad de transacciones por casos de uso.

Tipo	Descripción	Peso	Cant * peso
Simple	El Caso de Uso contiene de 1 a 3 transacciones.	5	2*5
Medio	El Caso de Uso contiene de 4 a 7 transacciones.	10	2*10
Complejo	El Caso de Uso contiene más de 8 transacciones.	15	0*15
Total:			30

Tabla 7 Cálculo del factor de peso de los casos de uso sin ajustar

La cantidad de transacciones se determina a partir de la descripción textual de los Casos de Uso.

Luego:

$$\text{UUCP} = 3 + 30$$

UUCP = 33.

6.1.2. Cálculo de Puntos de Casos de Uso ajustados.

$$UCP = UUCP * TCF * EF$$

Donde:

UCP: puntos de casos de uso ajustados.

UUCP: puntos de casos de uso sin ajustar.

TCF: factor de complejidad técnica.

EF: factor de ambiente.

Para calcular factor de complejidad técnica (**TCF**):

$$TCF = 0.6 + 0.01 * \sum (\text{Peso}_i * \text{Valor}_i) \text{ (Donde } \textit{Valor} \text{ es un número del 0 al 5).$$

Significado de los valores:

0: No presente o sin influencia.

1: Influencia incidental o presencia incidental.

2: Influencia moderada o presencia moderada.

3: Influencia media o presencia media.

4: Influencia significativa o presencia significativa.

5: Fuerte influencia o fuerte presencia.

Capítulo VI. Factibilidad del Sistema

Factor	Descripción	Peso	Valor	Comentario	$\Sigma (\text{Peso}_i * \text{Valor}_i)$
T1	Sistema distribuido.	2	4	El sistema es distribuido.	8
T2	Objetivos de performance o tiempo de respuesta.	1	5	Se requiere que el sistema tenga un buen rendimiento.	5
T3	Eficiencia del usuario final.	1	2	No hay restricciones de eficiencia.	2
T4	Procesamiento interno complejo.	1	4	No hay cálculos complejos.	4
T5	El código debe ser reutilizable.	1	4	Es reutilizable.	4
T6	Facilidad de instalación.	0.5	3	El sistema debe ser fácil de instalar.	1,5
T7	Facilidad de uso.	0.5	5	De ser un sistema	2,5

Capítulo VI. Factibilidad del Sistema

				amigable.	
T8	Portabilidad.	2	0	Se requiere que el sistema sea portable.	0
T9	Facilidad de cambio.	1	5	Se requiere que sea un sistema flexible ante cambios.	5
T10	Concurrencia.	1	5	Si hay concurrencia.	5
T11	Incluye objetivos especiales de seguridad.	1	5	El sistema gestiona información cuya confidencialidad es de carácter limitado.	5
T12	Provee acceso directo a terceras partes.	1	3	Provee acceso directo a terceras partes.	3
T13	Se requieren facilidades especiales de entrenamiento a los usuarios.	1	2	No se requieren facilidades especiales de entrenamiento a los usuarios.	2
Total:					49

Capítulo VI. Factibilidad del Sistema

Tabla 8 Cálculo del Factor de Complejidad Técnica

$$TCF = 0.6 + 0.01 * 49$$

$$TCF = 1.09$$

Para Calcular EF:

$$EF = 1.4 - 0.03 * \Sigma (\text{Peso}_i * \text{Valor}_i) \text{ (Donde Valor es un número del 0 al 5)}$$

Factor	Descripción	Peso	Valor	Comentario	$\Sigma (\text{Peso}_i * \text{Valor}_i)$
E1	Familiaridad con el modelo de proyecto utilizado.	1.5	2	El grupo no está familiarizado con el modelo de proyecto.	3
E2	Experiencia en la aplicación.	0.5	2	No hay mucha experiencia en la aplicación.	1
E3	Experiencia en orientación a objetos.	1	4	La mayoría del grupo ha programado Orientado a Objetos.	4

Capítulo VI. Factibilidad del Sistema

E4	Capacidad del analista líder.	0.5	4	La Analista Principal comenzó recientemente en el proyecto, así que no estuvo presente en la fase de inicio del proyecto.	2
E5	Motivación.	1	1	El grupo no está altamente motivado.	1
E6	Estabilidad de los requerimientos.	2	3	Se esperan cambios.	6
E7	Personal part-time.	-1	0	Todo el equipo es full-time.	0
E8	Dificultad del lenguaje de programación.	-1	3	Se usará el lenguaje PHP orientado a objetos, no estudiado. Aunque el lenguaje no es difícil de aprender.	-3
Total:					13

Tabla 9 Cálculo del Factor Ambiente

$$EF = 1.4 - 0.03 * 13$$

$$EF = 1.01$$

Luego:

$$UCP = 33 * 1.09 * 1.01$$

$$UCP = 36.3297.$$

6.1.3. Cálculo del Esfuerzo.

$$E = UCP * CF$$

Donde:

E: esfuerzo estimado en horas-hombre.

UCP: Puntos de Casos de Uso ajustados.

CF: factor de conversión.

Para calcular Factor de Conversión (CF):

CF = 20 horas-hombre (si Total EF \leq 2).

CF = 28 horas-hombre (si Total EF = 3 ó Total EF = 4).

CF = abandonar o cambiar proyecto (si Total EF \geq 5).

Total EF = Cant. EF < 3 (entre E1 – E6) + Cant. EF > 3 (entre E7 – E8)

Como:

Total EF = 3 + 0

Total EF = 3.

CF = 28 horas-hombre (porque Total EF = 3).

Luego:

$E = 36.3297 * 28 \text{ horas-hombre}$

$E = 1017,59 \text{ horas-hombre.}$

6.1.4. Distribución del Esfuerzo entre las diferentes actividades del módulo.

Actividad	% esfuerzo	Valor esfuerzo
Análisis	10	169,59
Diseño	30	508,79
Implementación	60	1017,59
Total	100	1695,98

Tabla 10 Distribución del esfuerzo estimado entre los flujos de trabajo de RUP.

6.2. Beneficios tangibles e intangibles.

El desarrollo del Módulo de *PHP* del Sistema de Análisis Estático de Vulnerabilidades del Laboratorio de Seguridad Informática de la Facultad 2 de la Universidad de las Ciencias Informáticas, aporta para esta última un considerable beneficio económico. El mayor aporte del módulo se enfoca en el servicio que se brinda para las pruebas de calidad en cuanto a seguridad de los proyectos en *PHP*, lo cual repercute en requerimientos de seguridad de todo proyecto informático que se realiza en este lenguaje en la Universidad. De igual forma, se brinda un servicio a la comunidad universitaria en general, no es necesaria la descarga de herramientas similares de un alto costo monetario y limitadas en tiempo de uso.

6.3. Análisis de costos y beneficios.

Salario medio de la fuerza de trabajo: \$100.

Cantidad de trabajadores: 2.

Como se tienen 1695,97 horas-hombre según lo estimado, se divide este valor entre 6 horas (cantidad real de horas que le dedica cada trabajador al módulo) para calcular la cantidad de días de trabajo-hombre que se necesitan para realizar el módulo. Al realizar esta operación matemática se obtiene que el módulo tiene 282,66 días de trabajo-hombre. Este valor se divide entre la cantidad de trabajadores para obtener la cantidad de días de trabajo que tiene cada trabajador para el desarrollo del módulo. De este resultado se concluye que con 2 trabajadores que trabajan 6 horas diarias, el módulo se terminaría en 142 días. Debido a que sólo se trabajan 6 días a la semana, se debe sumar a esta estimación 14 días más, que se corresponden con los 14 domingos que están dentro de esa planificación y que no se trabajan. Por tanto finalmente para realizar el módulo se necesitan aproximadamente 156 días (5 meses y 6 días). El costo final del módulo es:

Costo total del módulo = salario medio * cantidad de trabajadores * duración proyecto (meses).

Costo total del módulo = \$ 100 * 2 * 5.2 = \$ 1040.

La creación de *software* en el mundo tiene un costo elevado por los altos precios que implantan las grandes compañías en este sentido. Sistemas similares al que se presenta son codiciados por los usuarios y su costo tiende a ser elevado, como se muestra en la Fundamentación Teórica de este trabajo de diploma.

Las herramientas que se utilizan en la elaboración del sistema son totalmente libres, no es necesario el pago de licencias, por tanto, no se incurre en gasto alguno en la utilización de las mismas.

Debido a lo planteado anteriormente, se concluye que la implantación del módulo *PHP* en el Sistema de Análisis Estático de Vulnerabilidades, es **factible**.

6.4. Conclusiones.

En este capítulo se realizó un estudio de la factibilidad del sistema que se presenta. Esto permite llegar a la conclusión de que resulta factible implementar el módulo de *PHP* en el Sistema de Análisis Estático de Vulnerabilidades. Se analizaron los beneficios tangibles e intangibles que proporciona la implantación del módulo en la Universidad de las Ciencias Informáticas, así como los costos que presuponen herramientas similares con respecto a la que se presenta y la factibilidad de ésta para su implantación en la Universidad.

CONCLUSIONES GENERALES

Luego de dar cumplimiento a los objetivos trazados al inicio de la investigación del Sistema de Análisis Estático de Vulnerabilidades, módulo *PHP*, se arriban a conclusiones generales, las que se presentan a continuación:

- El estudio realizado sobre las herramientas existentes hoy en día para el análisis de vulnerabilidades de código en *PHP* condujo a plantear que las mismas no reúnen los requisitos necesarios para su aplicación en la Universidad de las Ciencias Informáticas, debido en algunos casos a la escasa búsqueda de vulnerabilidades o en otros por el monto monetario que supone su adquisición.
- El estudio crítico y seguimiento de la propuesta de solución que se realizó, permitió desarrollar una aplicación que cumple con los requisitos definidos para la misma.
- El sistema satisface los requerimientos del usuario final como son usabilidad, disponibilidad, agradable interfaz y fácil mantenimiento.
- La selección del analizador de código estático *Pixy*, hizo posible la detección de vulnerabilidades de forma precisa y flexible, con un aceptable número de falsos positivos.
- La vinculación de *Pixy* a un sistema *Web* en *PHP* requirió la optimización de la aplicación en sentido general, mediante el uso de funciones y procedimientos que permiten el multiproceso y facilitan la velocidad y estabilidad del sistema.
- El resultado de esta investigación puede ser aplicada en entornos productivos.

Conclusiones generales

- El sistema que culmina permite una validación de la calidad de los productos de la *UCI* en cuanto a seguridad de código se refiere.

RECOMENDACIONES

La experiencia acumulada en la realización de esta investigación, permite la propuesta de las siguientes recomendaciones:

- Utilizar esta investigación como referencia para investigaciones futuras en el área de la programación segura, en especial para *PHP*.
- Configurar el analizador de código estático *Pixy* a versiones de *PHP* de futuro desarrollo, así como a *frameworks* de este lenguaje de frecuente uso en la Universidad de las Ciencias Informáticas como *Symfony* y *Code Igniter*.
- Implantar el sistema en el Laboratorio de Seguridad Informática de la Facultad 2, para que se brinde un nuevo servicio a la comunidad universitaria de la *UCI*.
- Ampliar las funcionalidades del sistema como resultado de nuevos requisitos a cumplir como la gestión de las vulnerabilidades que se detectan.

REFERENCIAS BIBLIOGRÁFICAS

1. **Mondelo Hernández, Yonny.** Cuarto Estudio Webmétrico en la Universidad de las Ciencias Informáticas. [En línea]. 2009. [Citado el: 15 de diciembre de 2009]. [Disponible en: <http://gforge.f10.uci.cu/frs/download.php/435/gpsciba-geweb-recorrido-4.zip>].
2. **Ramiró Aguirre, Jorge.** Libro Electrónico de Seguridad Informática y Criptografía. [En línea] 2006. [Citado el: 20 de enero de 2010]. [Disponible en: http://eva.uci.cu/file.php/237/Criptografia/Libro_Electronico_de_Seguridad_Informatica/03IntroSegInfo.ppt].
3. **Fernández- Sanguino Peña, Javier.** Programación Segura. [En línea] 2006. [Citado el: 20 de enero de 2010]. [Disponible en: http://crl.iic.uam.es/descargas_web/cursos_verano/20060701/Javier_Fernandez/Programacion_segura.pdf].
4. **OWASP.** The Ten Most Critical WEB Application Security Risks -2010. [En línea] 2009. [Citado el: 10 de enero de 2010]. [Disponible en: http://www.owasp.org/index.php/File:OWASP_T10_-_2010_rc1.pdf].
5. **Huang, Yao-Wen y otros.** Securing Web Application Code by Static Analysis and Runtime Protection. [En línea] 2004. [Citado el: 25 de enero de 2010]. [Disponible en: <http://citeseerx.ist.psu.edu/viewdoc/download?Doi=10.1.1.62.2884&rep=rep1&type=pdf>].
6. **Consortium PHP Security.** PHP Security Guide. [En línea] 2007. [Citado el: 25 de enero de 2010]. [Disponible en: <http://phpsec.org/projects/guide>].
7. **Crespo Pérez, Joaquín.** El análisis de código, fuente de seguridad. [En línea] 2007. [Citado el: 26 de enero de 2010]. [Disponible en: [http://www.s21sec.com/descargas/Analisis de Aplicativos y del Codigo Fuente_ES.pdf](http://www.s21sec.com/descargas/Analisis_de_Aplicativos_y_del_Codigo_Fuente_ES.pdf)].

8. **Wikipedia**. Static code analysis. [En línea] 2009. [Citado el: 2 de febrero de 2010]. [Disponible en: http://en.wikipedia.org/wiki/Static_code_analysis].
9. Ídem Referencia 7.
10. **Jovanovic Nenad, Kruegel, Christopher y Kirda, Engin**. Pixy: XSS and SQLI Scanner for PHP Programs. [En línea] 2007. [Citado el: 5 de febrero de 2010]. [Disponible en: <http://pixybox.seclab.tuwien.ac.at/pixy/basics.php>].
11. **Klein, Gerwin**. JFlex - The Fast Scanner Generator for Java. [En línea] 1999. [Citado el: 3 de febrero de 2010]. [Disponible en: <http://jflex.de/manual.html>].
12. **E. Hudson, Scott**. CUP User's Manual. [En línea] 1999. [Citado el: 3 de febrero de 2010]. [Disponible en: <http://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html>].
13. **Grosch, Josef**. Lalr - A Generator for Efficient Parsers. [En línea] 1988. [Citado el: 3 de febrero de 2010]. [Disponible en: <ftp://www.cocolab.com/products/cocktail/doc.pdf/lalr.pdf>].
14. **Simionato, Michele**. An Introduction to GraphViz and dot. [En línea] 2004. [Citado el: 3 de marzo de 2010]. [Disponible en: http://linuxdevcenter.com/pub/a/linux/2004/05/06/graphviz_dot.html].
15. **Molpeceres, Alberto**. Procesos de desarrollo: RUP, XP y FDD. [En línea] 2002. [Citado el: 15 de enero de 2010]. [Disponible en: http://www.javahispano.org/contenidos/archivo/71/metodos_desarrollo.pdf].
16. **Tejeda Deivinson y otros**. Kumbia PHP Framework. [En línea] 2009. [Citado el: 26 de enero de 2010]. [Disponible en: http://ufpr.dl.sourceforge.net/sourceforge/kumbia/Manual_Kumbia_PHP_Framework_v0-5.pdf].
17. **Jacobson Ivar, Booch Grady y Rumbaugh James**. El proceso Unificado de Desarrollo de Software. 1era edición. Madrid : Pearson Educación, S.A. - Addison Wesley, 2000. 84-7829-036-2.

18. Ídem Referencia 17.
19. Ídem Referencia 17.
20. **Ayuda en español del Rational Unified Process.** s.l.: IBM Corporation, 2006.

BIBLIOGRAFÍA

1. **Alshanetsky, Iliia.** A step-by-step Guide to Writing Secure and Reliable PHP Applications. [En línea]. 2005. [Consultado el: 2 de febrero de 2010]. [Disponible en: <http://bibliodoc.uci.cu/pdf/0-9738621-0-6.pdf>].
2. **B. Carlsson., D. Baca.** “Software Security Analysis - Managing Source Code Audit”, 31st EUROMICRO Conference, 2005.
3. **B. Dejan, C. Bengt, L. Lars.** Evaluating the Cost Reduction of Static Code Analysis for Software Security. [En línea]. 2005. [Consultado el: 15 de enero de 2010]. [Disponible en: [https://www.bth.se/fou/forskinfo.nsf/alfs/fd6df0504ce32471c12574a900308b93/\\$file/plas08ACM.pdf](https://www.bth.se/fou/forskinfo.nsf/alfs/fd6df0504ce32471c12574a900308b93/$file/plas08ACM.pdf)]
4. **Consortium PHP Security.** PHP Security Guide. [En línea] 2007. [Consultado el: 2 de febrero de 2010]. [Disponible en: <http://phpsec.org/projects/guide>].
5. **Crespo Pérez, Joaquín.** El análisis de código, fuente de seguridad. [En línea] 2007. [Consultado el: 26 de enero de 2010]. [Disponible en: http://www.s21sec.com/descargas/Analisis de Aplicativos y del Codigo Fuente_ES.pdf].
6. **E. Hudson, Scott.** CUP User's Manual. [En línea] 1999. [Consultado el: 3 de febrero de 2010]. [Disponible en: <http://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html>].
7. **Fernández-Sanguino Peña, Javier.** Programación Segura. [En línea] 2006. [Consultado el: 20 de enero de 2010]. [Disponible en: http://crl.iic.uam.es/descargas_web/cursos_verano/20060701/Javier_Fernandez/Programacion_segura.pdf].
8. **Grosch, Josef.** Lalr - A Generator for Efficient Parsers. [En línea] 1988. [Consultado el: 3 de febrero de 2010]. [Disponible en: <ftp://www.cocolab.com/products/cocktail/doc.pdf/lalr.pdf>].

9. **Gutmans Andy, Bakken Stig, Rethans Derick.** PHP 5 Power Programming. [En línea]. 2005. [Consultado el: 2 de febrero de 2010]. [Disponible en: <http://bibliodoc.uci.cu/pdf/reg04073.pdf>].
10. **Hernández León Rolando, Alfredo y Coello González, Sayda.** El paradigma cuantitativo de la investigación científica. [PDF] Ciudad de la Habana : Edu, Eduniv - Editorial Universitaria, 2002. 959-16-0343-6.
11. **Huang Yao-Wen y otros.** Securing Web Application Code by Static Analysis and Runtime Protection. [En línea] 2004. [Consultado el: 4 de marzo de 2010]. [Disponible en: <http://citeseerx.ist.psu.edu/viewdoc/download?Doi=10.1.1.62.2884&rep=rep1&type=pdf>].
12. **J. Kabir, Mohammed.** Secure PHP Development: Building 50 practical applications. [En línea]. 2003. [Consultado el: 5 de febrero de 2010]. [Disponible en: http://rapidshare.com/files/34407161/Secure_PHP_Development_Building_50_Practical_Applications.rar].
13. **Jovanovic, Nenad.** Web Application Security. [En línea]. 2007. [Consultado el: 5 de marzo de 2010]. [Disponible en: <http://www.seclab.tuwien.ac.at/papers/phdthesis-nenad.pdf>]
14. **Jovanovic Nenad, Kirda Engin y Kruegel Christopher.** Preventing Cross Site Request Forgery Attacks. [En línea]. 2006. [Consultado el: 5 de marzo de 2010]. [Disponible en: <http://www.seclab.tuwien.ac.at/papers/noforge.pdf>]
15. **Jovanovic Nenad, Kirda Engin y Kruegel Christopher.** Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper). [En línea]. 2006. [Consultado el: 5 de marzo de 2010]. [Disponible en: <http://www.seclab.tuwien.ac.at/papers/pixy.pdf>]
16. **Jovanovic Nenad, Kirda Engin y Kruegel Christopher.** Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications. [En línea]. 2007. [Consultado el: 5 de marzo de 2010]. [Disponible en: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.148.5596&rep=rep1&type=pdf>]

17. **Jovanovic Nenad, Kirda Engin y Kruegel Christopher.** Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Technical Report). [En línea]. 2006. [Consultado el: 6 de marzo de 2010]. [Disponible en: www.iseclab.org/papers/pixy_techreport.pdf].
18. **Jovanovic Nenad, Kruegel Christopher y Kirda Engin.** Pixy: XSS and SQLI Scanner for PHP Programs. [En línea] 2007. [Consultado el: 6 de marzo de 2010]. [Disponible en: <http://pixybox.seclab.tuwien.ac.at/pixy/basics.php>].
19. **Kernel Panic Labs.** Desarrollo seguro de aplicaciones. [En línea]. [Consultado el: 1 de junio de 2010]. [Disponible en: <http://www.kriptopolis.org/docs/procseg.zip>].
20. **Klein, Gerwin.** JFlex - The Fast Scanner Generator for Java. [En línea]. 1999. [Consultado el: 3 de febrero de 2010]. [Disponible en: <http://jflex.de/manual.html>].
21. **Livshits, Benjamin.** Improving software security with precise static and runtime analysis. [En línea]. 2006. [Consultado el: 4 de febrero de 2010]. [Disponible en: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.77.7196&rep=rep1&type=pdf>].
22. **Laboratorio de Seguridad Informática.** Manual de Despliegue del Sistema de Análisis Estático de Vulnerabilidades SAEV. 2010.
23. **McGraw, Gary.** Static Analysis for Security. [En línea] 2004. [Consultado el: 3 de febrero de 2010]. [Disponible en: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.7235&rep=rep1&type=pdf>].
24. **Montero Ayala, Ramón.** XML. Inicialización y referencia. Madrid : Osborne McGraw-Hill, 2001. 84-481-2894-X.
25. **OWASP Foundation.** The Ten Most Critical WEB Application Security Risks -2010. [En línea] 2009. [Consultado el: 2 de febrero de 2010]. [Disponible en: http://www.owasp.org/index.php/File:OWASP_T10_-_2010_rc1.pdf].

26. **OWASP Foundation.** Guía de Pruebas OWASP V3.0. [En línea]. 2008. [Consultado el: 25 de mayo de 2010]. [Disponible en: <http://www.owasp.org/index.php>].
27. **Rumbaugh James, Jacobson Ivar y Booch Grady.** El lenguaje unificado de modelado. Manual de referencia. s.l. : Addison Wesley, 2000.
28. **Sitio Oficial de Pixy.** [En línea]. 2007. [Consultado el: 20 de mayo de 2010]. <http://pixybox.seclab.tuwien.ac.at/pixy/>
29. **Sitio Oficial de PHP.** [En línea]. 2001 - 2009. [Consultado el: 15 de abril de 2010]. <http://www.php.net/>.
30. **Sitio Oficial de JQuery.** [En línea]. 2010. [Consultado el: 10 de abril de 2010]. <http://jquery.com/>.
31. **Sitio Oficial de KumbiaPHP.** [En línea]. 2007-2010. [Consultado el: 10 de abril de 2010]. <http://www.kumbiaphp.com/>.
32. **Snyder Chris y Southwell.** Michael. Pro PHP Security. [En línea]. 2005. [Consultado el: 10 de mayo de 2010]. [Disponible en: <http://bibliodoc.uci.cu/pdf/1-59059-508-4.pdf>].
33. **Tejeda Deivinson y otros.** Kumbia PHP Framework. [En línea] 2009. [Consultado el: 15 de mayo de 2010]. [Disponible en: http://ufpr.dl.sourceforge.net/sourceforge/kumbia/Manual_Kumbia_PHP_Framework_v0-5.pdf].
34. **V. Scovetta, Michael.** Yasca v1.0 User Guide. [En línea]. 2009. [Consultado el: 20 de febrero de 2010]. [Disponible en: www.docstoc.com/docs/24243187/Yasca-v10-User-Guide]
35. **Vaswani, Vikram.** PHP Programming Solutions. [En línea]. 2007. [Consultado el: 15 de marzo de 2010]. [Disponible en: http://www.ebookpdf.net/find-real-world-solutions-to-php-programming-probl_1_399.html].


ANEXOS

ANEXO 1

DESCRIPCIÓN DE LOS CASOS DE USO DEL SISTEMA

Nombre del Caso de Uso:	Subir fuente.
Actores:	Usuario (inicia).
Propósito:	Permite subir un fichero al servidor para ser analizado.
Resumen:	El Caso de Uso inicia cuando el usuario selecciona el <i>tab</i> Subir fichero o el <i>tab</i> Subir código. En cualquiera de los dos casos, la fuente se sube al servidor para ser procesado posteriormente.
Referencias:	R1.
Precondiciones:	El usuario tiene que estar autenticado en el sistema.
Postcondiciones:	La fuente seleccionada está lista para ser analizada.
Curso normal de eventos	

Acción del actor	Respuesta del sistema
1. El usuario selecciona la opción “PHP” del menú principal.	2. El sistema muestra interfaz de análisis para “PHP”.
Sección “Subir fichero”	
3. El usuario selecciona la opción del panel Subir Fichero del menú principal.	4. El sistema muestra interfaz para subir archivo con los botones: Subir Fichero y Ayuda.
5. El usuario selecciona el botón Subir Fichero.	6. El sistema presenta ventana de carga de archivos.
7. El usuario selecciona archivo para ser subido al servidor.	
8. El usuario presiona el botón “Aceptar”.	9. El sistema comprueba que el fichero seleccionado tenga la extensión y tamaño requerido.
	10. El sistema sube archivo al servidor.
	11. El sistema presenta mensaje de advertencia: “El fichero fue salvado correctamente”, además de información del fichero que va a ser analizado como es nombre de fichero, extensión, tamaño, cantidad de ficheros PHP, memoria y tiempo utilizado en la subida al servidor.
Flujo alternativo 9a “Archivo con formato no permitido”	
	9a.1. El sistema muestra mensaje de error “No se

	pudo subir el fichero al servidor. El fichero que está intentando subir no tiene un formato permitido”.
Flujo alterno 9b “Archivo con tamaño excedido”	
	9b.1. El sistema muestra mensaje de error “Tamaño no permitido. Verifique que el tamaño del fichero sea mayor que 0 y menor o igual que 20 MB”.
Prototipo de Interfaz	
 <p>The screenshot shows a web interface for uploading files. At the top, there are navigation tabs: 'Subir Código', 'Subir Fichero', 'Analizar', 'Reportes', 'Estadísticas', and 'Ayuda para PHP'. The main heading is 'Subir Fichero'. Below it, there is a 'Subir Fichero' button. A 'Mensaje de SAEV' dialog box is open, displaying a success message: 'El fichero fue salvado correctamente.' with an 'Ok' button. To the left of the dialog, there is a section titled 'Datos del código fuente' with the following details: Nombre del Archivo: p, Extensión: .php, Tamaño: 0.0002 MB, Ficheros PHP: 1, Tiempo Total de Subida: 1.0975 SEG, and Memoria Utilizada: 0.9327 MB.</p>	
Sección “Subir código”	
3. El usuario selecciona la opción del panel Subir Código del menú principal.	4. El sistema muestra interfaz para subir archivo con los elementos: <i>editarea</i> con botones nuevo, salvar y tamaño de fuente.
5. El usuario selecciona el botón Salvar del	6. El sistema presenta mensaje de advertencia “El código se está salvando en el servidor. Espere unos

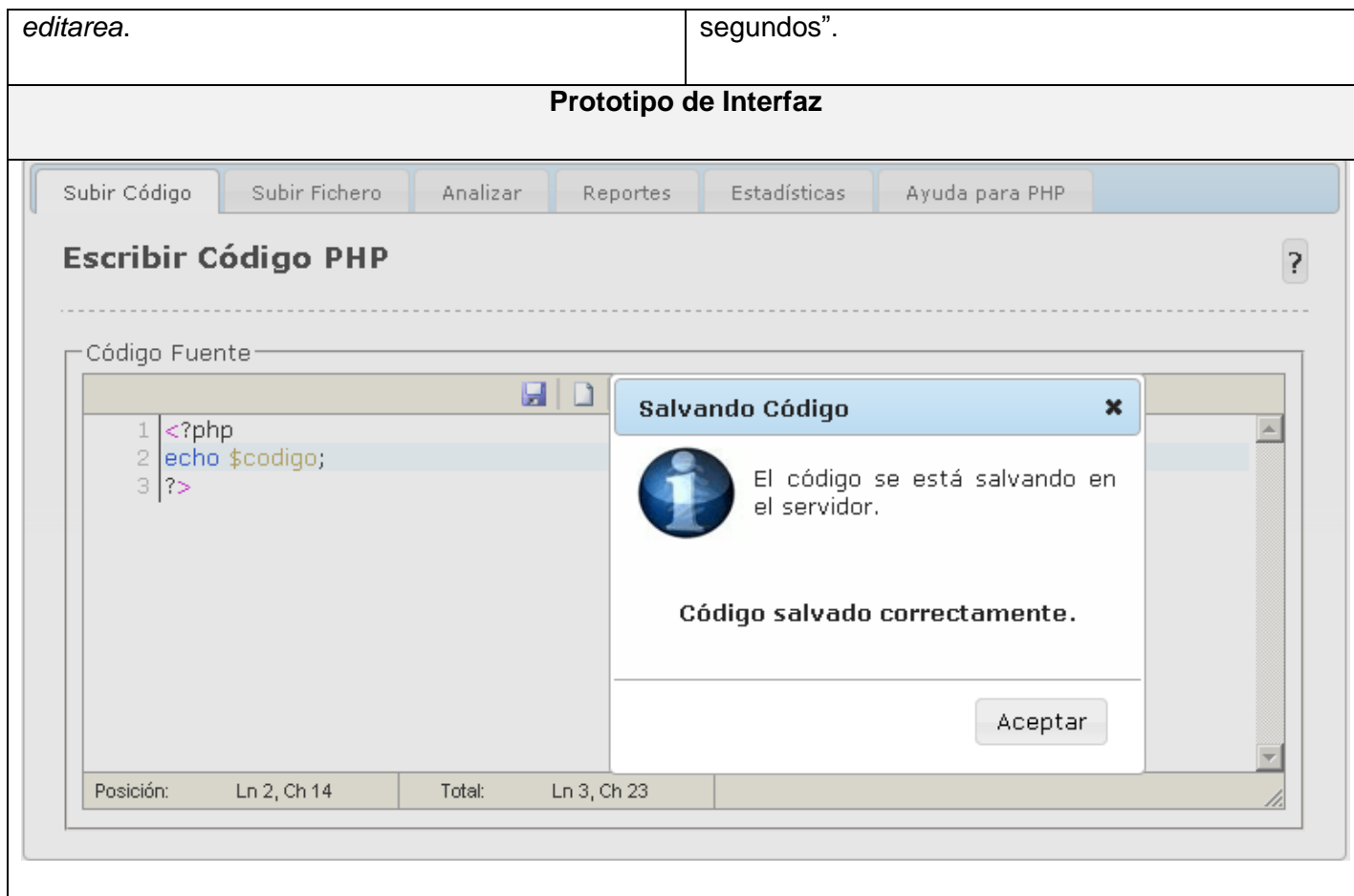


Tabla 11 Descripción textual CU Subir Fuente

Nombre del Caso de Uso:	Generar reporte.
Actores:	Usuario (inicia).
Propósito:	Se genera un reporte que es el resultado del análisis realizado a un fichero compactado o <i>PHP</i> y también al código escrito por el usuario en la ventana

	correspondiente.
Resumen:	El Caso de Uso se inicia cuando el usuario solicita el análisis de un fichero que ha sido subido al servidor. El sistema procesa la información y devuelve el resultado en un informe <i>.pdf</i> . El usuario puede seleccionar múltiples opciones tanto de análisis como de salida del reporte.
Referencias:	R2, R3.
Precondiciones:	El usuario debe estar autenticado en el sistema y el fichero debe estar subido en el servidor.
Postcondiciones:	El sistema genera reporte de acuerdo a las opciones establecidas.
Curso normal de eventos	
Acción del actor	Respuesta del sistema
1. El usuario selecciona la opción del panel principal Analizar.	2. El sistema presenta panel con las opciones: <ul style="list-style-type: none"> • Analizar. • Configuración de análisis. • Funciones <i>Sinks</i>.
3. El usuario selecciona el botón "Analizar".	4. El sistema presenta ventana de diálogo ¿Desea confirmar el análisis? con los botones Aceptar y Cancelar, además de información del fichero que va a ser analizado como es nombre de fichero, tamaño,

	tipo, memoria y tiempo utilizado en la subida al servidor.
5. El usuario presiona el botón Aceptar.	6. El sistema presenta botón “Cancelar análisis”.
	7. El sistema presenta mensaje “El análisis culminó correctamente. Tiempo empleado <i>Tiempo</i> ” y presenta numeración consecutiva en el tab Reporte del nuevo fichero analizado no leído.
Flujo alternativo 2a “No hay ninguna fuente subida al sistema”	
	2a.1. El sistema presente mensaje informativo “Debe subir alguna fuente para analizar”.
Flujo alternativo 3a “El usuario selecciona configuración de análisis”	
	<p>3a.1. El sistema presenta ventana con las siguientes opciones de análisis:</p> <p>Tipos de análisis:</p> <ul style="list-style-type: none"> • SQL • RFI • XSS <p>Opciones de análisis:</p> <ul style="list-style-type: none"> • Alias. • Register Globals.

	<p>Opciones adicionales:</p> <ul style="list-style-type: none"> • Envío de correo. <p>Opciones de Reporte:</p> <ul style="list-style-type: none"> • Mostrar errores sintácticos. • Mostrar warnings. • Mostrar funciones. • Mostrar incluidos.
<p>3a.2. El usuario selecciona opciones deseadas.</p>	
<p>3a.3. El usuario presiona botón “Aceptar”.</p>	<p>3a.4. El sistema establece opciones de análisis y regresa a vista principal.</p>
<p>Flujo alternativo 3a.1 “Usuario activo es líder de proyecto”</p>	
	<p>3a.1.1 El sistema presenta ventana con las siguientes opciones de análisis:</p> <p>Tipos de análisis:</p> <ul style="list-style-type: none"> • SQL • RFI • XSS <p>Opciones de análisis:</p>

	<ul style="list-style-type: none"> • Alias. • Register Globals. <p>Opciones adicionales:</p> <ul style="list-style-type: none"> • Envío de correo. • Archivo de proyecto. <p>Opciones de Reporte:</p> <ul style="list-style-type: none"> • Mostrar errores sintácticos. • Mostrar warnings. • Mostrar funciones.
Flujo alterno 3b “El usuario establece funciones <i>Sinks</i>”	
	<p>3b.1. El sistema presenta ventana con los siguientes campos editables:</p> <ul style="list-style-type: none"> • Función. • Parámetros. • Clasificación. <p>También se presentan los botones adicionar sink, deshacer todos y aceptar.</p>
<p>3b.2. El usuario selecciona el botón Adicionar Sink.</p>	<p>3b.3. El sistema presenta ventana con los campos editables: Función Sink, Parámetros y campo</p>

	seleccionable Clasificación (XSS, SQL, RFI), con los botones Aceptar y Cancelar. También se presentan los <i>radiobuttons</i> Función y Método.
2b.4. El usuario llena los campos solicitados.	
3b.5. El usuario selecciona el botón Aceptar.	3b.6. El sistema regresa a vista de Funciones Sinks.
Flujo alterno 3b.2 “Usuario selecciona botón Deshacer Todos”	
	3b.2.a. El sistema limpia todas las sinks con parámetros escritos.
Flujo alterno 3b.6 “Usuario introduce valores incorrectos”	
	3b.6.a. El sistema resalta en color rojo campo con valor introducido no permitido y limpia los textbox Nombre de Sink y Parámetros.
Flujo alterno 5a “Usuario selecciona botón No”	
	5a.1. El sistema regresa a pantalla de Análisis.
Flujo alterno 6a “Usuario cancela el análisis”	
	6a.1. El sistema detiene el análisis en proceso mostrando el mensaje “El análisis ha sido cancelado”.
Flujo alterno 7a “No existe código para analizar”	
	7a.1. El sistema presenta mensaje de advertencia “No existe código fuente para analizar.”

Flujo alternativo 7b “Lo subido al servidor es un código o un fichero PHP”	
	7b.1. El sistema presenta editarea del reporte de análisis de vulnerabilidades para código PHP, incluyendo el botón Guardar Análisis.
7b.2. El usuario selecciona el botón Guardar Análisis.	7b.3. El sistema abre en una nueva ventana documento .pdf con opciones de guardado.
Flujo alternativo 7c “Se especificó la opción Enviar Correo”	
	7c.1. El sistema presenta mensaje “El análisis culminó correctamente, el reporte le fue enviado por correo electrónico”, presenta numeración consecutiva en el tab Reporte del nuevo fichero analizado no leído.
Flujo alternativo 7d “No se detectaron vulnerabilidades en el análisis”	
	7d.1. El sistema presenta mensaje informativo “No se detectaron vulnerabilidades en el código”.
Prototipo de Interfaz	

Subir Código Subir Fichero Analizar Reportes Estadísticas Ayuda para PHP

Analizar

Configuración de Análisis Funciones Sink Analizar

Reporte de Análisis de Vulnerabilidades para Código PHP

Estadísticas generales

Cantidad total de vulnerabilidades: 1
 Cantidad de warnings: 0
 Funciones sanitizadoras: 0
 Funciones malignas: 0
 Funciones multidependencias: 0
 Funciones débiles sanitizadoras: 0
 Funciones inversas: 0

Vulnerabilidades

Guardar Análisis

Tabla 12 Descripción textual CU Generar Reporte

Nombre del Caso de Uso:	Gestionar reportes.
Actores:	Usuario (inicia).
Propósito:	Se muestran todos los reportes realizados que han sido generados a un usuario con la posibilidad de eliminarlos y descargarlos.
Resumen:	El Caso de Uso inicia cuando el usuario solicita ver los reportes personales generados por el sistema, se permite filtrar los mismos mediante fechas de

	generación y lecturas realizadas, se muestran vínculos disponibles para la descarga de los ficheros y para eliminarlos.
Referencias:	R4, R5, R6.
Precondiciones:	El usuario debe estar autenticado en el sistema.
Postcondiciones:	El sistema presenta todos los reportes consultados.
Curso normal de eventos	
Acción del actor	Respuesta del sistema
1. El usuario selecciona el <i>tab</i> "Reportes".	2. El sistema muestra interfaz de visualización de reportes con los siguientes elementos: <ul style="list-style-type: none"> • Campo fecha: Fecha Inicio. • Campo fecha: Fecha Final. • Checkbox: Leído. • Botón: Buscar. • Botón: Buscar todos.
3. El usuario selecciona opciones deseadas.	
4. El usuario selecciona el botón Buscar.	5. El sistema muestra interfaz de visualización de reportes filtrados según fecha y lectura con los siguientes campos: <ul style="list-style-type: none"> • Checkbox.

	<ul style="list-style-type: none"> • No. • Fecha de generación. • Código fuente. • Leído. • Acciones (eliminar y descargar).
	6. El sistema presenta botones Seleccionar Todos, Deseleccionar Todos, Eliminar todos.
7. El usuario selecciona botón Seleccionar Todos.	8. El sistema selecciona todos los <i>checkbox</i> de la pantalla.
9. El usuario selecciona botón Eliminar todos.	10. El sistema presenta mensaje de advertencia “Los reportes activos serán eliminados ¿Desea continuar?” con los botones Aceptar y Cancelar.
11. El usuario selecciona el botón Aceptar.	12. El sistema elimina reportes seleccionados.
Flujo alterno 3a “Usuario selecciona botón Buscar todos”	
	<p>3a.1. El sistema muestra interfaz de visualización de todos los reportes con los siguientes campos:</p> <ul style="list-style-type: none"> • No. • Fecha de generación. • Código fuente.

	<ul style="list-style-type: none"> • Leído. • Acciones (eliminar y descargar).
Flujo alternativo 5a “Intervalo de fechas inválido”	
	5a.1. El sistema presenta mensaje de advertencia “Error: La fecha inicial debe ser menor que la fecha final.”.
Flujo alternativo 5c “No existencia de reportes”	
	5c.1. El sistema presenta mensaje de advertencia: “No se encontraron reportes para mostrar”.
Flujo alternativo 11a “Usuario cancela eliminar todos los reportes”	
	11a.1. El sistema regresa a vista principal.
Sección “Descargar reporte”	
11. El usuario selecciona el ícono “Descargar reporte”.	12. El sistema presenta en una ventana link del reporte el cual puede ser guardado en visor PDF.
	13. El sistema modifica campo Leído del reporte seleccionado a Leído.
Prototipo de Interfaz	

The screenshot shows a web application interface with a navigation bar containing buttons for 'Subir Código', 'Subir Fichero', 'Analizar', 'Reportes', 'Estadísticas', and 'Ayuda para PHP'. The main section is titled 'Reportes' and includes a search area with 'Fecha Inicio' and 'Fecha Final' filters, and buttons for 'Buscar' and 'Buscar Todos'. A table lists reports with columns for 'No', 'Fecha', 'Origen Fuente', 'Leído', and 'Acciones'. A modal dialog titled 'Descargar Reporte' is open, displaying a loading icon and the text 'Se está descargando el reporte...' and 'Click para Abrir el Reporte'. Below the dialog are buttons for 'Seleccionar Todos', 'Deseleccionar Todos', and 'Eliminar seleccionados'. The page footer indicates 'pág. 1 de 1 | items 1-5'.

Sección “Eliminar reporte”

11. El usuario selecciona el <i>link</i> “Eliminar reporte”.	12. El sistema presenta ventana de confirmación: ¿Está seguro que desea borrar todos los reportes seleccionados? con los botones Aceptar y Cancelar.
13. El usuario selecciona el botón Aceptar.	14. El sistema elimina reporte seleccionado y presenta vista principal de reportes.
Flujo alterno 13a “Usuario selecciona botón Cancelar”	
	13a.1. El sistema regresa a la vista principal.

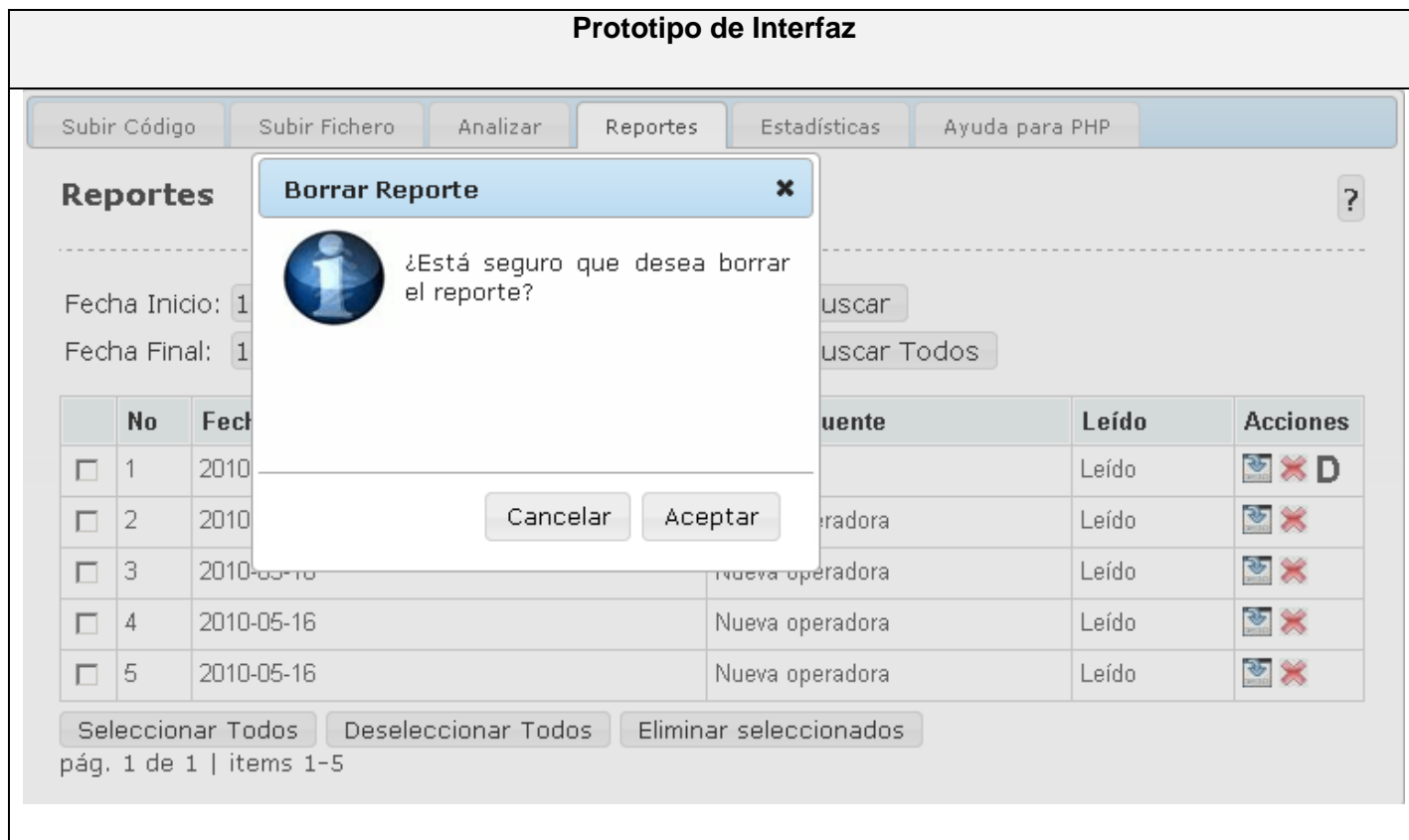


Tabla 13 Descripción textual CU Gestionar Reportes

Nombre del Caso de Uso:	Mostrar estadísticas.
Actores:	Usuario (inicia).
Propósito:	Se presentan las estadísticas asociadas al usuario en <i>PHP</i> bajo diferentes parámetros.
Resumen:	El Caso de Uso inicia cuando el usuario solicita ver las estadísticas de vulnerabilidades asociadas a proyecto, usuarios o de forma general para <i>PHP</i> . El

	sistema presenta las estadísticas asociadas a la selección realizada, terminando el caso de uso.
Referencias:	R7.
Precondiciones:	El usuario debe estar autenticado en el sistema
Postcondiciones:	El sistema presenta las estadísticas deseadas.
Curso normal de eventos	
Acción del actor	Respuesta del sistema
1. El usuario selecciona el <i>tab</i> “Estadísticas” del panel principal.	2. El sistema muestra interfaz de visualización de estadísticas con los siguientes botones: <ul style="list-style-type: none"> • Vulnerabilidades de <i>Usuario</i>. • Vulnerabilidades de PHP. • Estadísticas de <i>Proyecto</i>. • Archivos analizados.
Flujo alternativo 2a “Usuario no Líder de proyecto”	
	2a.1. El sistema muestra interfaz de visualización de estadísticas con las siguientes opciones: <ul style="list-style-type: none"> • Vulnerabilidades de <i>Usuario</i>. • Vulnerabilidades de PHP. • Archivos analizados.

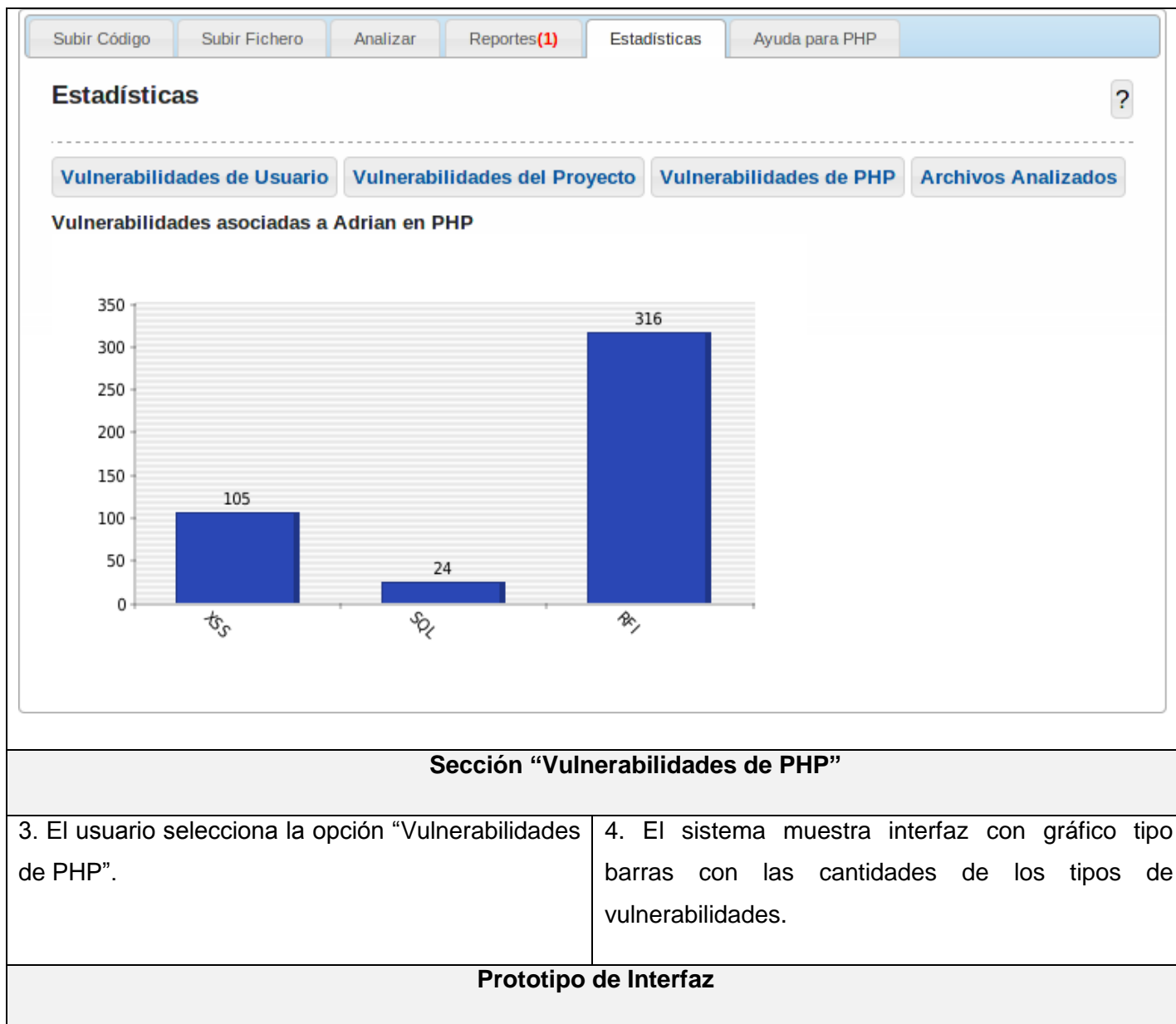
Sección “Vulnerabilidades de Usuario”

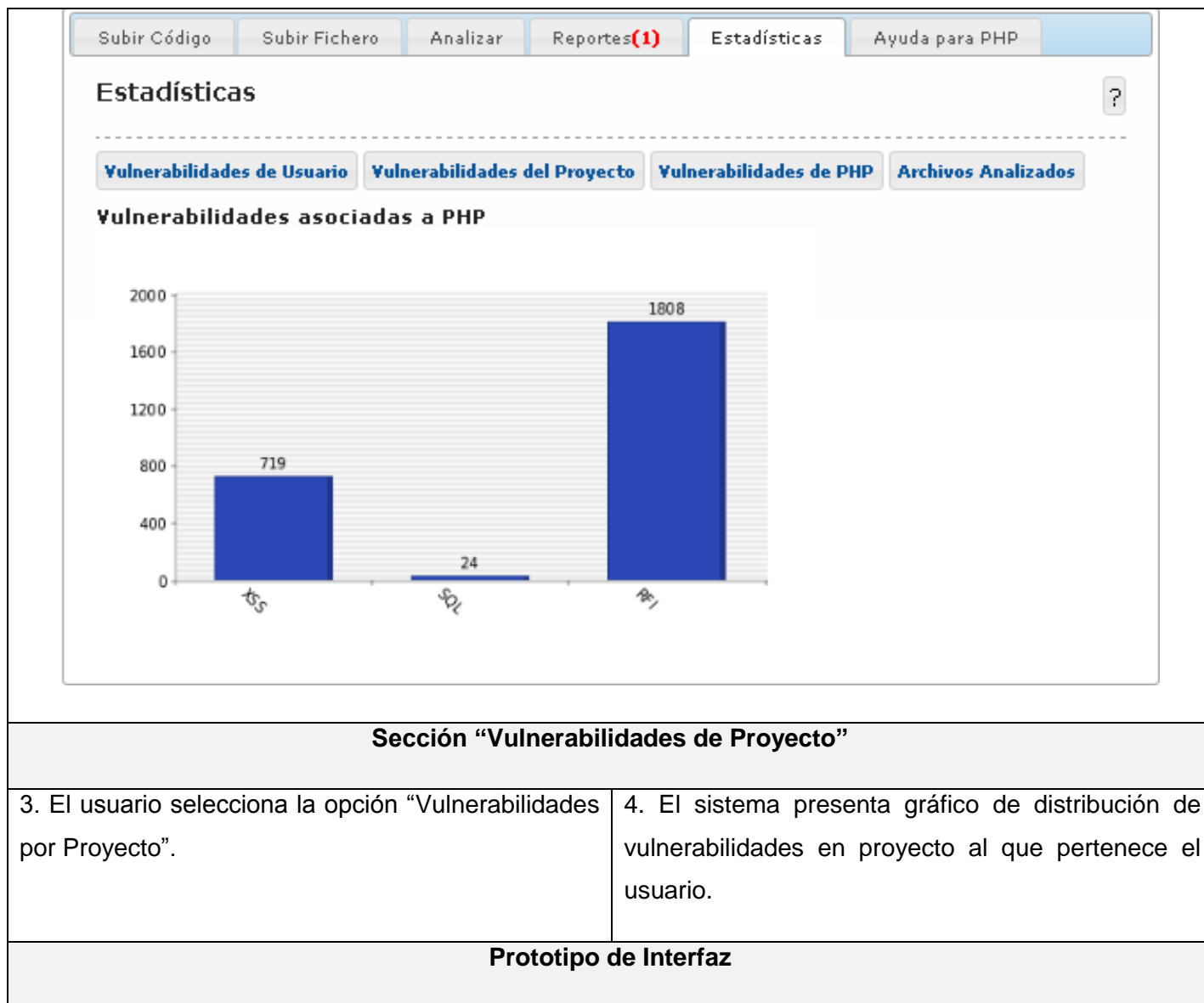
3. El usuario selecciona el botón “Vulnerabilidades de Usuario”.

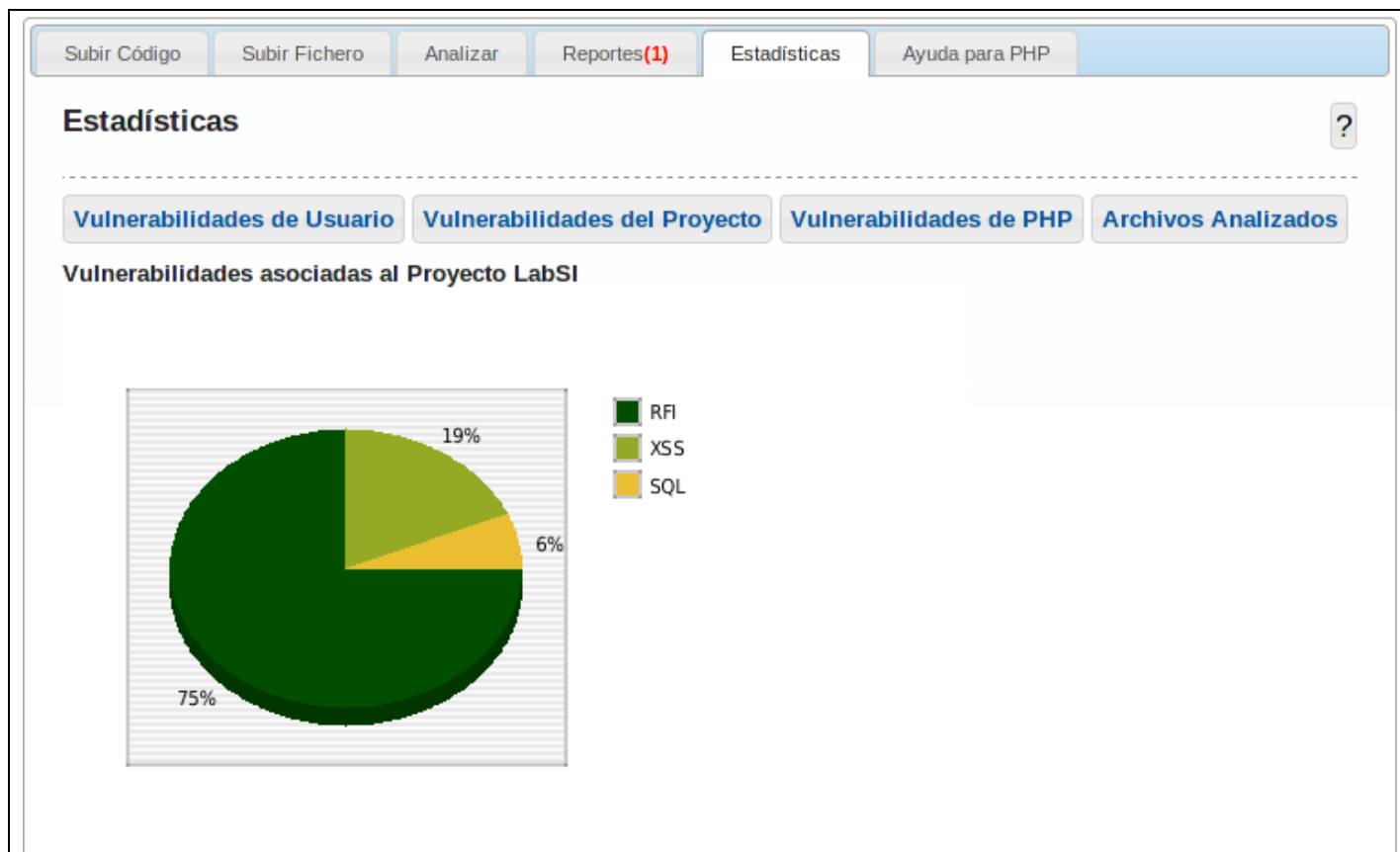
4. El sistema muestra tabla con los siguientes valores:

- No.
- Vulnerabilidad.
- Gravedad.
- Cantidad.

Prototipo de Interfaz







Sección "Archivos analizados"

3. El usuario selecciona la opción "Archivos subidos".

4. El sistema muestra interfaz con las siguientes opciones:

- No.
- Fecha de subida.
- Hora de subida.

	<ul style="list-style-type: none">• Tipo de fichero.• Nombre del fichero.• Tamaño.
Flujo Alternativo 4a "Usuario autenticado es líder de proyecto"	
	<p>4a.1 El sistema muestra interfaz con las siguientes opciones:</p> <ul style="list-style-type: none">• No.• Fecha de subida.• Hora de subida.• Tipo de fichero.• Nombre del fichero.• Tamaño.• Archivo de Proyecto.
Prototipo de Interfaz	

Subir Código Subir Fichero Analizar Reportes(2) Estadísticas Ayuda para PHP

Estadísticas

Vulnerabilidades de Usuario Vulnerabilidades del Proyecto Vulnerabilidades de PHP Archivos Analizados

Archivos Analizados en el Servidor

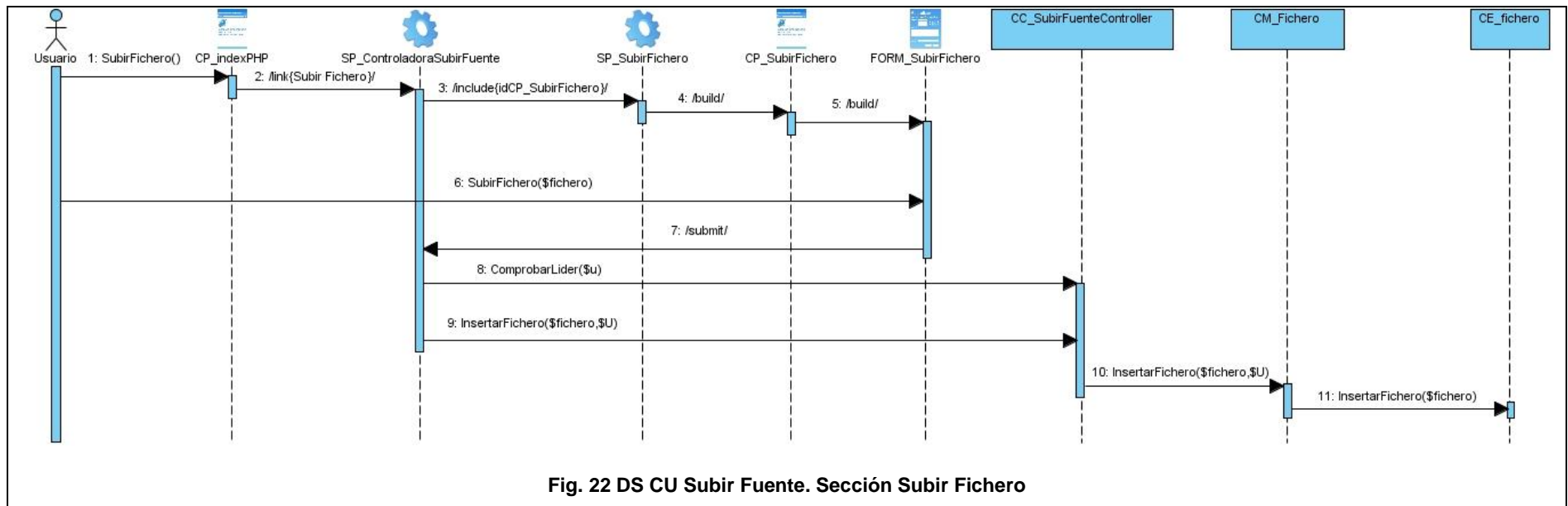
No	Fecha de Subida	Hora de Subida	Tipo Fichero	Nombre del Fichero	Tamaño	Fichero de Proyecto
1	2010-05-19		zip	apps	952.58	NO
2	2010-05-19		zip	apps	952.58	NO
3	2010-05-19		rar	kumbia	655.28	NO
4	2010-05-19		zip	apps	952.58	NO
5	2010-05-19		zip	apps	952.58	SI
6	2010-05-19		zip	apps	952.58	SI
7	2010-05-19		rar	kumbia	655.28	SI
8	2010-05-19		rar	kumbia	655.28	SI
9	2010-05-19		rar	kumbia	655.28	SI
10	2010-05-19		zip	apps	952.58	SI

pág. 1 de 3 | items 1-10 **Siguiente**

Tabla 14 Descripción textual CU Mostrar Estadísticas

ANEXO 2

DIAGRAMAS DE SECUENCIA (DS) DE CADA CASO DE USO (CU)



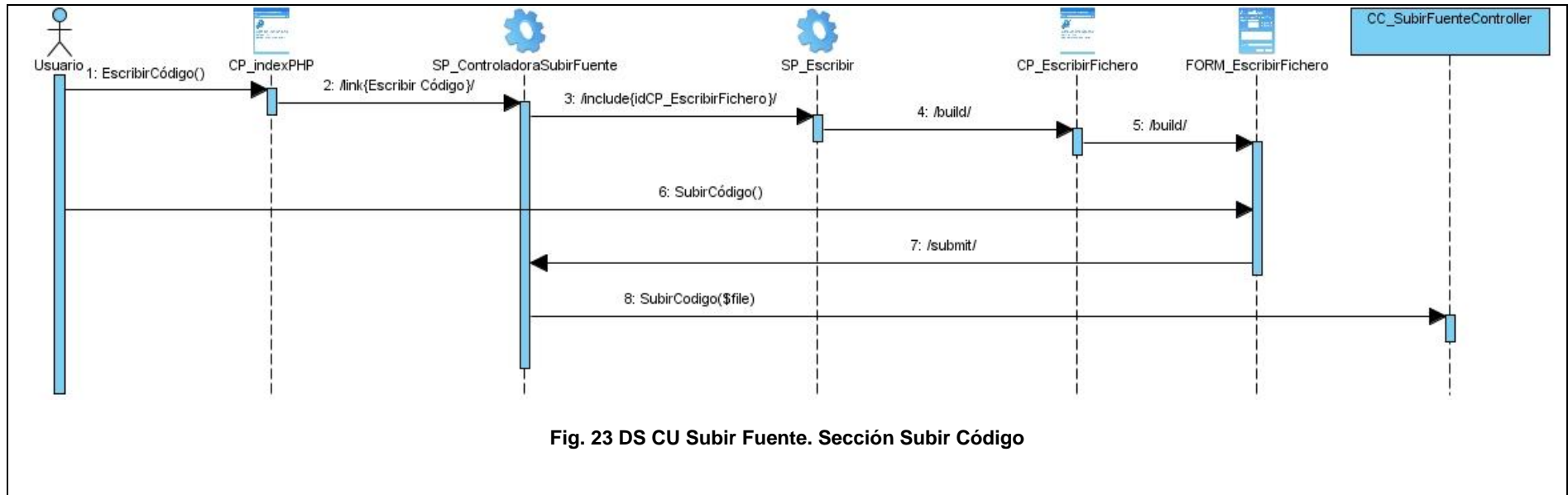
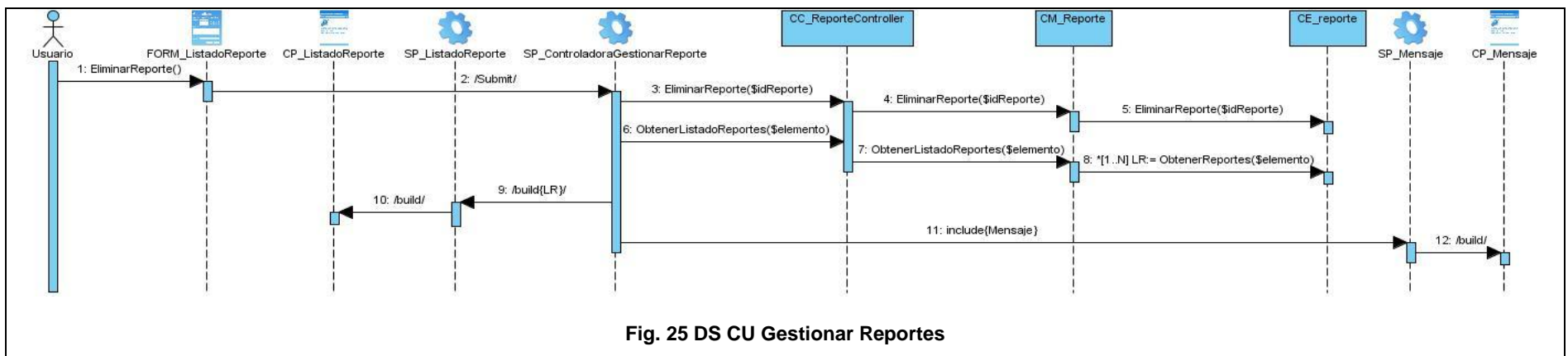
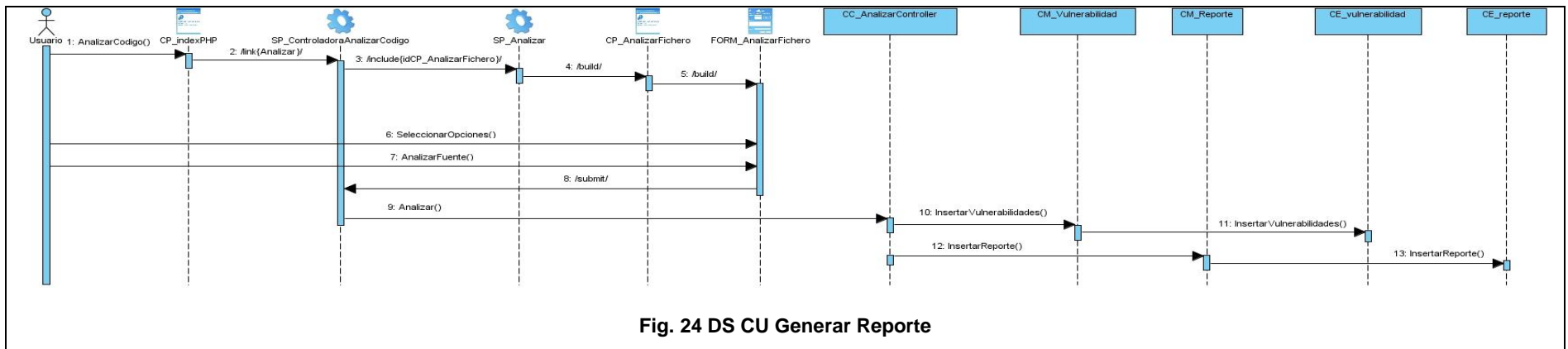
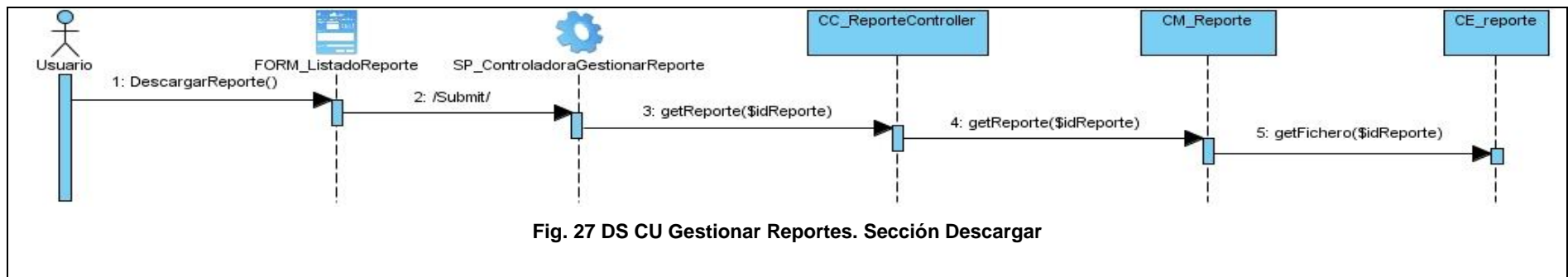
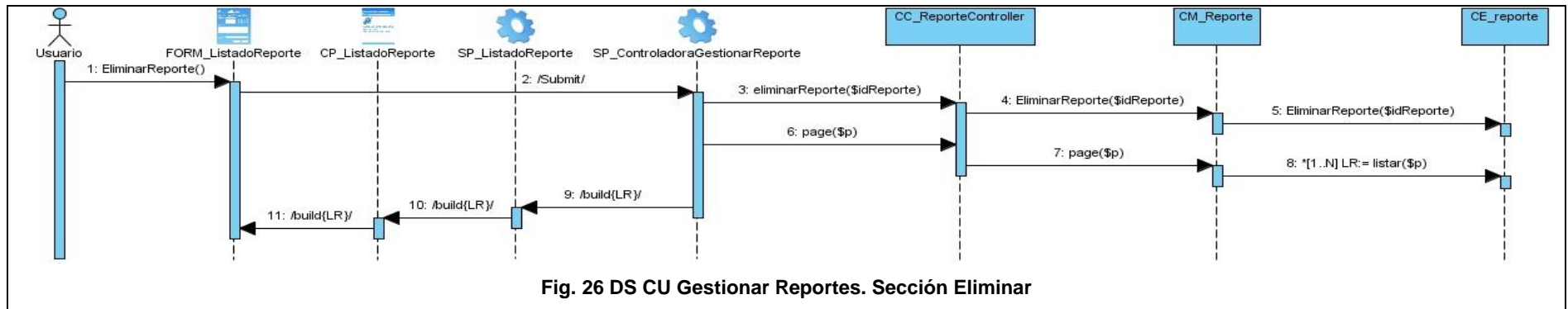
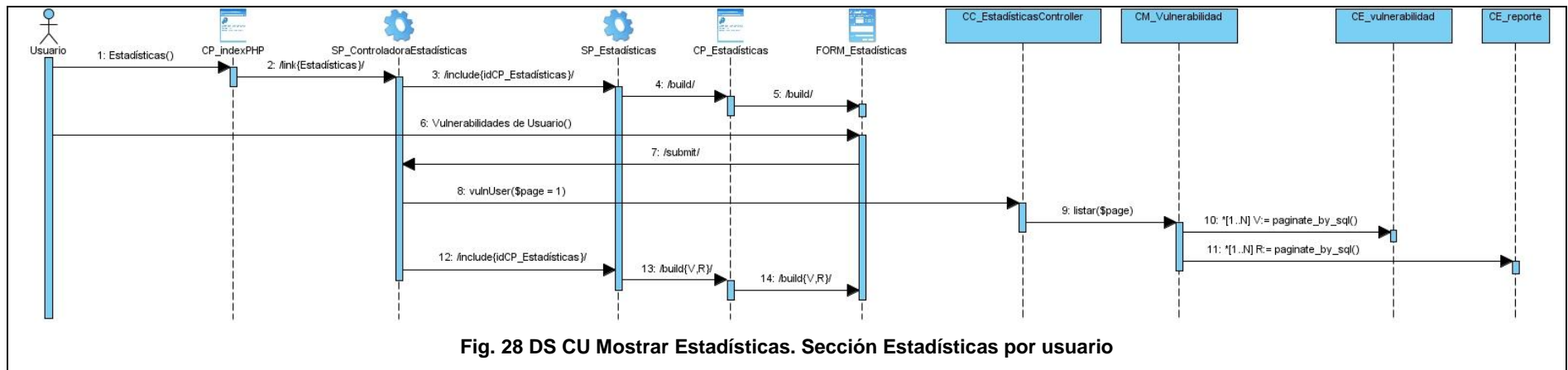
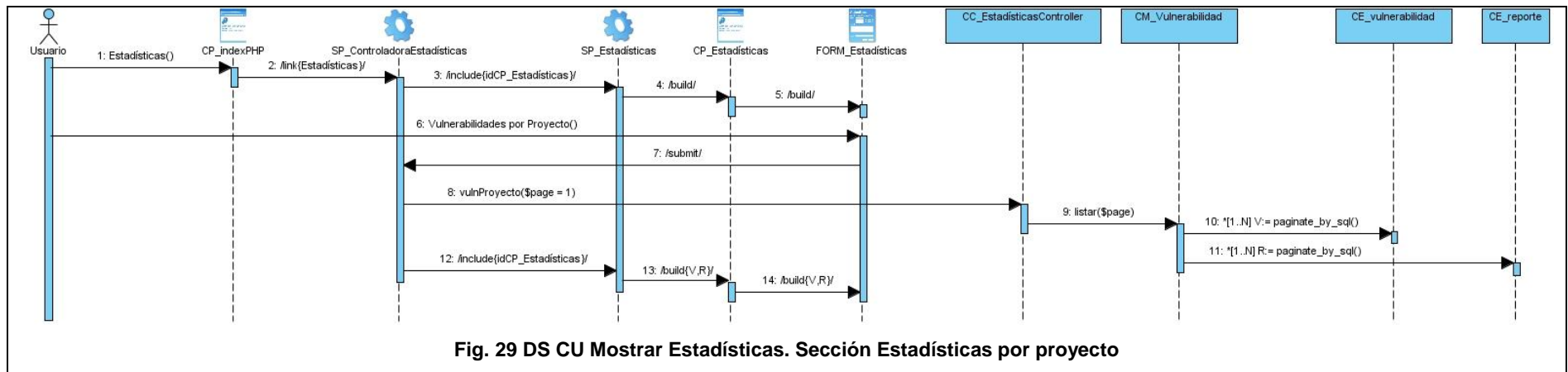


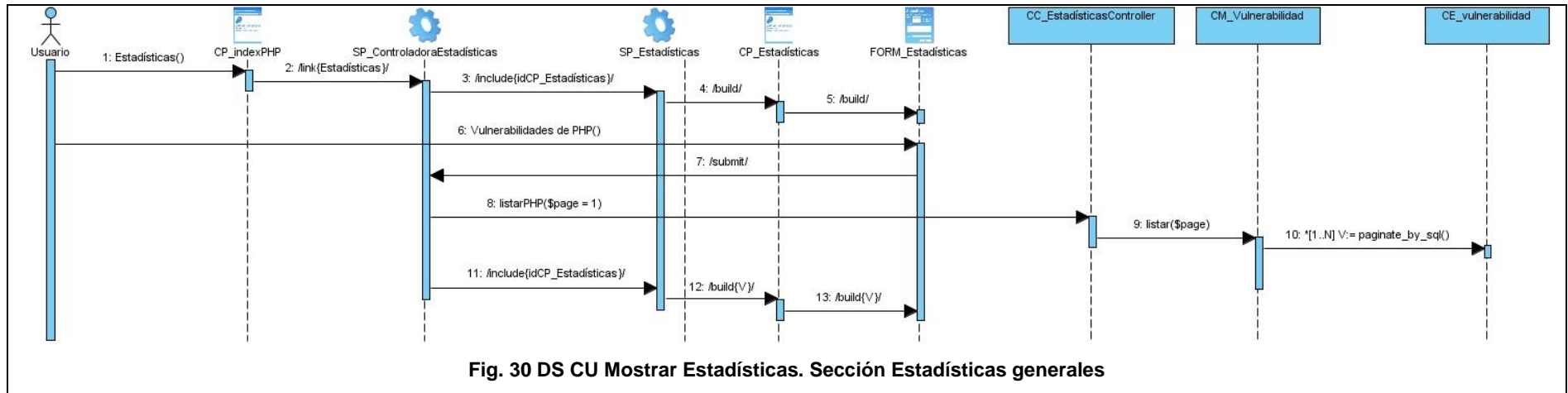
Fig. 23 DS CU Subir Fuente. Sección Subir Código

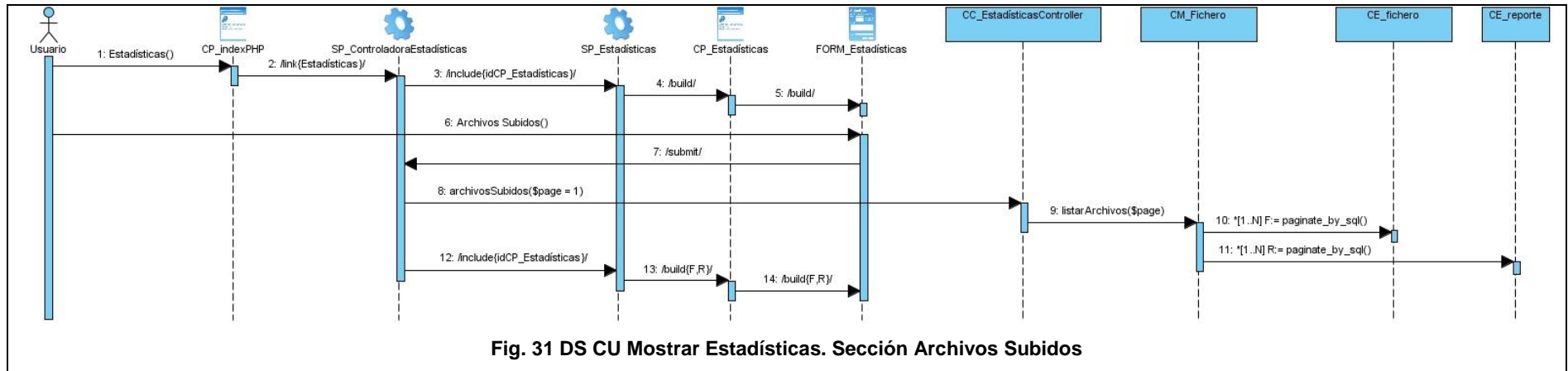












ANEXO 3

DIAGRAMA DE CLASES DEL ANALIZADOR DE CÓDIGO ESTÁTICO PIXY

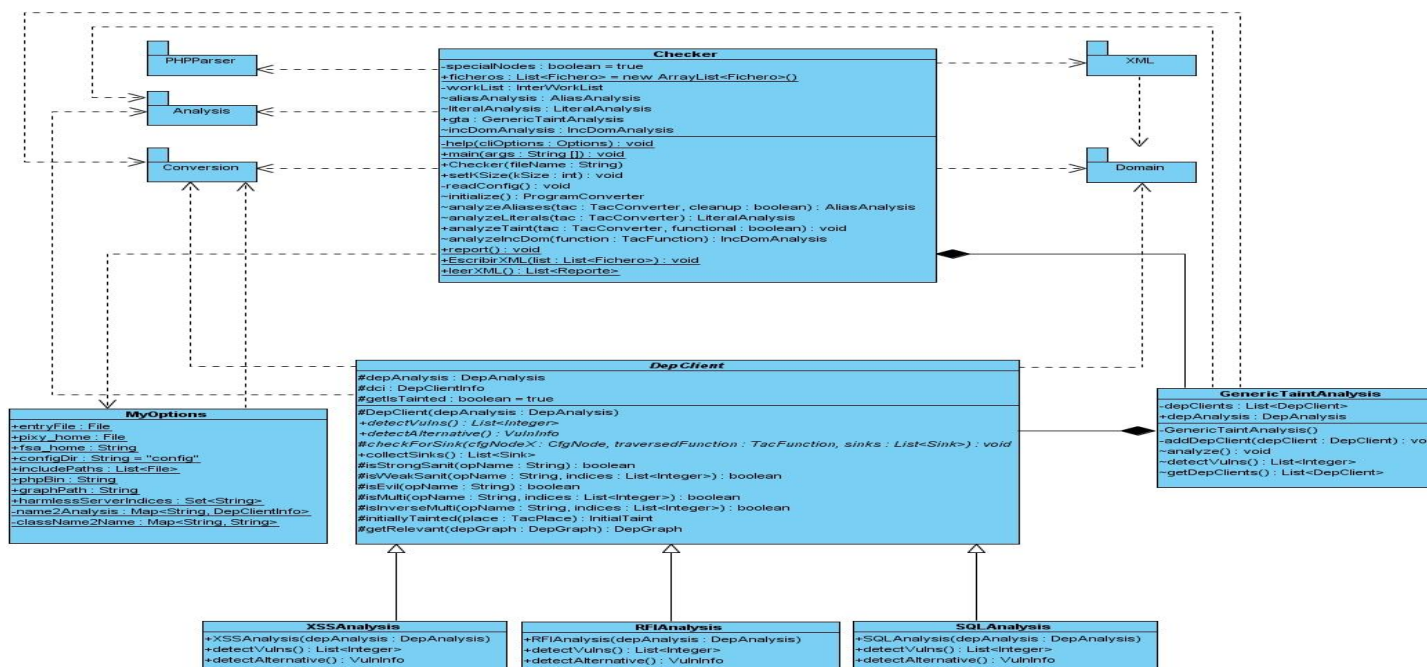


Fig. 32 Diagrama de clases del analizador de código estático Pixy

ANEXO 4

REGLAS ADICIONADAS EN JFLEX Y JCUP

Las reglas que se presentan a continuación representan una ampliación del *parser* que presenta *Pixy*. Las mismas están enfocadas en elementos de *PHP 5*. Estas reglas se dividen en dos categorías: *JFLEX* que permite el reconocimiento léxico y *JCUP* para el análisis sintáctico.

TRY...CATCH:

JFLEX:

```
<ST_IN_SCRIPTING>"try" {  
  
    return symbol(PhpSymbols.T_TRY, "T_TRY");  
  
}
```

```
<ST_IN_SCRIPTING>"catch" {  
  
    return symbol(PhpSymbols.T_CATCH, "T_CATCH");  
  
}
```

JCUP:

```
unticked_statement ::=
```

```
T_TRY statement catch_statement optional_catch_statement {: RESULT =
createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top, prodNumber, prodName,
prodLength); :}
```

catch_statement ::=

```
T_CATCH T_OPEN_BRACES T_STRING cvar T_CLOSE_BRACES statement {: RESULT =
createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top, prodNumber, prodName,
prodLength); :}
```

optional_catch_statement ::=

```
catch_statement optional_catch_statement {: RESULT =
createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top, prodNumber, prodName,
prodLength); :}
```

```
| statement{: RESULT = createRuleNode(CUP$PhpParser$stack,
CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

THROW:

JFLEX:

```
<ST_IN_SCRIPTING>"throw" {
    return symbol(PhpSymbols.T_THROW, "T_THROW");
}
```

JCUP:

unticked_statement ::=

```
T_THROW      throw_possibilities      T_SEMICOLON  {:      RESULT      =
createRuleNode(CUP$PhpParser$stack,  CUP$PhpParser$top,  prodNumber,  prodName,
prodLength); :}
```

throw_possibilities ::=

```
T_NEW T_STRING T_OPEN_BRACES scalar T_CLOSE_BRACES {: RESULT =
createRuleNode(CUP$PhpParser$stack,  CUP$PhpParser$top,  prodNumber,  prodName,
prodLength); :}
```

```
|      T_NEW T_STRING T_OPEN_BRACES T_CLOSE_BRACES {: RESULT =
createRuleNode(CUP$PhpParser$stack,  CUP$PhpParser$top,  prodNumber,  prodName,
prodLength); :}
```

```
|      T_NEW T_STRING T_OPEN_BRACES T_VARIABLE T_CLOSE_BRACES {:
RESULT = createRuleNode(CUP$PhpParser$stack,  CUP$PhpParser$top,  prodNumber,
prodName, prodLength); :}
```

```
|      cvar {: RESULT = createRuleNode(CUP$PhpParser$stack,  CUP$PhpParser$top,
prodNumber, prodName, prodLength); :}
```

Visibilidad de atributos en clases:

JFLEX:

```
<ST_IN_SCRIPTING>"private" {
```

```
    return symbol(PhpSymbols.T_PRIVATE, "T_PRIVATE");
```

```
}
```

```
<ST_IN_SCRIPTING>"protected" {
```

```
    return symbol(PhpSymbols.T_PROTECTED, "T_PROTECTED");
```

```
}  
  
<ST_IN_SCRIPTING>"public" {  
  
    return symbol(PhpSymbols.T_PUBLIC, "T_PUBLIC");  
  
}
```

JCUP:

class_statement ::=

```
    class_visibility    class_variable_declaration    T_SEMICOLON    {:    RESULT    =  
    createRuleNode(CUP$PhpParser$stack,    CUP$PhpParser$top,    prodNumber,    prodName,  
    prodLength); :}
```

class_visibility ::=

```
    T_PRIVATE    T_STATIC    {:    RESULT    =    createRuleNode(CUP$PhpParser$stack,  
    CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

```
|    T_PRIVATE {: RESULT = createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top,  
    prodNumber, prodName, prodLength); :}
```

```
|    T_PUBLIC    T_STATIC    {:    RESULT    =    createRuleNode(CUP$PhpParser$stack,  
    CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

```
|    T_PUBLIC {: RESULT = createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top,  
    prodNumber, prodName, prodLength); :}
```

```
|    T_PROTECTED    T_STATIC    {:    RESULT    =    createRuleNode(CUP$PhpParser$stack,  
    CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

```
|    T_PROTECTED {: RESULT = createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top,  
    prodNumber, prodName, prodLength); :}
```

```
| T_VAR {: RESULT = createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top,
prodNumber, prodName, prodLength); :}
```

Visibilidad de funciones:

JFLEX:

```
<ST_IN_SCRIPTING>"private" {
    return symbol(PhpSymbols.T_PRIVATE, "T_PRIVATE");
}
<ST_IN_SCRIPTING>"protected" {
    return symbol(PhpSymbols.T_PROTECTED, "T_PROTECTED");
}
<ST_IN_SCRIPTING>"public" {
    return symbol(PhpSymbols.T_PUBLIC, "T_PUBLIC");
}
```

JCUP:

```
class_statement ::=
```

```
function_vissibility T_FUNCTION is_reference T_STRING T_OPEN_BRACES parameter_list
T_CLOSE_BRACES T_OPEN_CURLY_BRACES inner_statement_list
T_CLOSE_CURLY_BRACES {: RESULT = createRuleNode(CUP$PhpParser$stack,
CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

```
| function_vissibility T_FUNCTION is_reference T_STRING T_OPEN_BRACES parameter_list  
T_CLOSE_BRACES T_SEMICOLON {: RESULT = createRuleNode(CUP$PhpParser$stack,  
CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

function_vissibility ::=

```
T_PRIVATE  {: RESULT = createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top,  
prodNumber, prodName, prodLength); :}
```

```
| T_FINAL  T_PRIVATE  {: RESULT = createRuleNode(CUP$PhpParser$stack,  
CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

```
| T_PRIVATE  T_STATIC  {: RESULT = createRuleNode(CUP$PhpParser$stack,  
CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

```
| T_FINAL  T_PUBLIC  {: RESULT = createRuleNode(CUP$PhpParser$stack,  
CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

```
| T_PUBLIC  {: RESULT = createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top,  
prodNumber, prodName, prodLength); :}
```

```
| T_PUBLIC  T_STATIC  {: RESULT = createRuleNode(CUP$PhpParser$stack,  
CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

```
| T_PROTECTED  {: RESULT = createRuleNode(CUP$PhpParser$stack,  
CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

```
| T_FINAL  T_PROTECTED  {: RESULT = createRuleNode(CUP$PhpParser$stack,  
CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

```
| T_PROTECTED  T_STATIC  {: RESULT = createRuleNode(CUP$PhpParser$stack,  
CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

```
| T_PROTECTED T_FINAL {: RESULT = createRuleNode(CUP$PhpParser$stack,
CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

```
| T_PRIVATE T_FINAL {: RESULT = createRuleNode(CUP$PhpParser$stack,
CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

```
| T_PUBLIC T_FINAL {: RESULT = createRuleNode(CUP$PhpParser$stack,
CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

```
| T_PUBLIC T_ABSTRACT {: RESULT = createRuleNode(CUP$PhpParser$stack,
CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

```
| T_PROTECTED T_ABSTRACT {: RESULT = createRuleNode(CUP$PhpParser$stack,
CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

```
| T_PRIVATE T_ABSTRACT {: RESULT = createRuleNode(CUP$PhpParser$stack,
CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

```
| /*empty*/ {: RESULT = createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top,
prodNumber, prodName, prodLength); :}
```

Palabras reservadas delante de class:

JFLEX:

```
<ST_IN_SCRIPTING>"abstract" {
    return symbol(PhpSymbols.T_ABSTRACT, "T_ABSTRACT");
}
```

```
<ST_IN_SCRIPTING>"final" {
    return symbol(PhpSymbols.T_FINAL, "T_FINAL");
}
```


}

JCUP:

unticked_declaration_statement ::=

```
optional_words  T_CLASS  T_STRING  T_IMPLEMENTS  T_STRING
T_OPEN_CURLY_BRACES class_statement_list T_CLOSE_CURLY_BRACES {:
RESULT = createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top,
prodNumber, prodName, prodLength); :}
```

optional_words ::=

```
T_FINAL {: RESULT = createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top,
prodNumber, prodName, prodLength); :}

| T_ABSTRACT {: RESULT = createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top,
prodNumber, prodName, prodLength); :}

| /*empty*/ {: RESULT = createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top,
prodNumber, prodName, prodLength); :}
```

Métodos abstractos:

JFLEX:

```
<ST_IN_SCRIPTING>"abstract" {

    return symbol(PhpSymbols.T_ABSTRACT, "T_ABSTRACT");

}
```

JCUP:

function_vissibility ::=

```
T_ABSTRACT function_vissibility T_FUNCTION is_reference T_STRING T_OPEN_BRACES
parameter_list T_CLOSE_BRACES T_SEMICOLON {: RESULT =
createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top, prodNumber, prodName,
prodLength); ;}
```

Métodos final.

JFLEX:

```
<ST_IN_SCRIPTING>"final" {
    return symbol(PhpSymbols.T_FINAL, "T_FINAL");
}
```

JCUP:

function_vissibility ::=

```
T_FINAL T_PROTECTED {: RESULT = createRuleNode(CUP$PhpParser$stack,
CUP$PhpParser$top, prodNumber, prodName, prodLength); ;}
```

Constantes de clases:

JFLEX:

```
<ST_IN_SCRIPTING>"const" {
    return symbol(PhpSymbols.T_CONST, "T_CONST");
}
```

JCUP:

class_statement ::=

```
T_CONST T_STRING T_ASSIGN expr T_SEMICOLON {: RESULT =
createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top, prodNumber, prodName,
prodLength); ;}
```

Interfaces.

JFLEX:

```
<ST_IN_SCRIPTING>"interface" {
    return symbol(PhpSymbols.T_INTERFACE, "T_INTERFACE");
}
```

JCUP:

unticked_declaration_statement ::=

```
T_INTERFACE T_STRING T_OPEN_CURLY_BRACES interface_statement
T_CLOSE_CURLY_BRACES {: RESULT =
createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top, prodNumber,
prodName, prodLength); ;}

| T_INTERFACE T_STRING T_EXTENDS T_STRING
T_OPEN_CURLY_BRACES interface_statement T_CLOSE_CURLY_BRACES {:
RESULT = createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top,
prodNumber, prodName, prodLength); ;}
```

interface_statement ::=

```
interface_statement function_visibility T_FUNCTION is_reference T_STRING
T_OPEN_BRACES parameter_list T_CLOSE_BRACES T_SEMICOLON {: RESULT =
```

```
createRuleNode(CUP$PhpParser$stack, CUP$PhpParser$top, prodNumber, prodName,  
prodLength); ;}
```

```
| /*empty */ {: RESULT = createRuleNode(CUP$PhpParser$stack,  
CUP$PhpParser$top, prodNumber, prodName, prodLength); ;}
```

Clonado de objetos.

JFLEX:

```
<ST_IN_SCRIPTING>"clone" {  
  
    return symbol(PhpSymbols.T_CLONE, "T_CLONE");  
  
}
```

JCUP:

```
expr_without_variable ::=
```

```
cvar T_ASSIGN T_CLONE expr {: RESULT = createRuleNode(CUP$PhpParser$stack,  
CUP$PhpParser$top, prodNumber, prodName, prodLength); ;}
```

Instancias de objetos:

JFLEX:

```
<ST_IN_SCRIPTING>"instanceof" {  
  
    return symbol(PhpSymbols.T_INSTANCEOF, "T_INSTANCEOF");  
  
}
```

JCUP:

expr_without_variable ::=

```
cvar T_INSTANCEOF T_STRING { : RESULT = createRuleNode(CUP$PhpParser$stack,
CUP$PhpParser$top, prodNumber, prodName, prodLength); :}
```

Análisis de vulnerabilidades en código PHP embebido en HTML.

JFLEX:

```
<YYINITIAL>(([^<]|"<[^\s<]*)|'<s'|"<"<?" {
```

```
    return symbol(PhpSymbols.T_INLINE_HTML, "T_INLINE_HTML");
```

```
}
```

ANEXO 5

FUNCIONES AGREGADAS EN PIXY

Función ²	Clasificación	Bloque al que pertenece
<i>strtotime</i>	FS	XSS
<i>intval</i>	FS	XSS, SQL
<i>sha1</i>	FS	XSS, SQL
<i>filter_var</i>	FS	XSS
<i>convert_uuencode</i>	FS	XSS, SQL
<i>bin2hex</i>	FS	XSS, SQL
<i>chr</i>	FS	XSS, SQL
<i>count_chars</i>	FS	XSS, SQL
<i>crypt</i>	FS	XSS, SQL
<i>html_entity_decode</i>	FS	XSS, SQL
<i>money_format</i>	FS	XSS, SQL
<i>ord</i>	FS	XSS, SQL

² La documentación de estas funciones se encuentra en <http://www.php.net>.

<i>stripos</i>	FS	XSS, SQL
<i>floatval</i>	FS	XSS,SQL
Funciones matemáticas: <i>abs, acos, acosh, asin, asinh, atan, atanh, cos, cosh, exp, log10, log, pi, pow, sin, sinh, round, tan, tanh, sqrt.</i>	FS	XSS, SQL
<i>escapeshellarg</i>	FS	RFI
<i>escapeshellcmd</i>	FS	RFI
<i>addslashes</i>	MD	XSS
<i>chop</i>	MD	XSS
<i>str_ireplace</i>	MD	XSS
<i>strchr</i>	MD	XSS
<i>stristr</i>	MD	XSS
<u><i>strchr</i></u>	MD	XSS
<i>substr_replace</i>	MD	XSS
<u><i>wordwrap</i></u>	MD	XSS
<i>fprintf</i>	MI	XSS
<i>htmlspecialchars_decode</i>	M	XSS
<i>shell_exec</i>	S	RFI
<i>system</i>	S	RFI
<i>exec</i>	S	RFI

<i>proc_open</i>	S	RFI
<i>popen</i>	S	RFI
<i>passthru</i>	S	RFI
<i>pcntl_exec</i>	S	RFI
<i>eval</i>	S	RFI
<i>include</i>	S	RFI
<i>require</i>	S	RFI
<i>include_once</i>	S	RFI
<i>require_once</i>	S	RFI
<i>chgrp</i>	S	RFI
<i>chmod</i>	S	RFI
<i>chown</i>	S	RFI
<i>copy</i>	S	RFI
<i>fopen</i>	S	RFI
<i>fsockopen</i>	S	RFI
<i>lstat</i>	S	RFI
<i>stat</i>	S	RFI
<i>mkdir</i>	S	RFI
<i>move_uploaded_file</i>	S	RFI
<i>pathinfo</i>	S	RFI
<i>readfile</i>	S	RFI

<i>rename</i>	S	RFI
<i>rmdir</i>	S	RFI
<i>unlink</i>	S	RFI
<i>is_dir</i>	S	RFI
<i>is_executable</i>	S	RFI
<i>is_file</i>	S	RFI
<i>is_link</i>	S	RFI
<i>is_readable</i>	S	RFI
<i>is_writable</i>	S	RFI
<i>is_writeable</i>	S	RFI
<i>disk_free_space</i>	S	RFI
<i>disk_total_space</i>	S	RFI
<i>dirname</i>	S	RFI
<i>fileperms</i>	S	RFI
<i>fileowner</i>	S	RFI
<i>filesize</i>	S	RFI
<i>filetype</i>	S	RFI
<i>chdir</i>	S	RFI
<i>filegroup</i>	S	RFI
<i>chroot</i>	S	RFI
<i>opendir</i>	S	RFI

<i>scandir</i>	S	RFI
<i>mail</i>	S	RFI
<i>pg_query</i>	S	SQL
<i>pg_insert</i>	S	SQL
<i>pg_update</i>	S	SQL
<i>pg_delete</i>	S	SQL
<i>pg_send_query</i>	S	SQL
<i>pg_send_query_params</i>	S	SQL
<i>mssql_query</i>	S	SQL
<i>odbc_exec</i>	S	SQL
<i>mysqli_multi_query</i>	S	SQL
<i>mysqli_query</i>	S	SQL
<i>mysqli_real_query</i>	S	SQL
<i>mysqli_query<m>*</i>	S	SQL
<i>mysqli_real_query<m>*</i>	S	SQL
<i>mysqli_multi_query<m>*</i>	S	SQL
<i>sqlite_query</i>	S	SQL
<i>sqlite_array_query</i>	S	SQL

<i>pg_escape_bytea</i>	W	SQL
<i>pg_escape_string</i>	W	SQL
<i>mysqli_real_escape_string</i>	W	SQL
<i>mysqli_escape_string</i>	W	SQL
<i>ftp_delete</i>	S	SQL
<i>ftp_exec</i>	S	SQL
<i>ftp_mkdir</i>	S	SQL
<i>ftp_rmdir</i>	S	SQL
<i>ftp_rename</i>	S	SQL
<i>ftp_put</i>	S	SQL
<i>ftp_fput</i>	S	SQL
<i>ftp_nb_fput</i>	S	SQL

Tabla 15 Funciones agregadas en Pixy

Leyenda:**FS:** fuerte sanitizadora.**MD:** multidependencia.**MI:** multidependencia inversa.**M:** maligna.**W:** débil sanitizadora.**S:** sink.*: Representa la extensión de *PHP Mysqli*.

ANEXO 6

DECLARACIONES DE DESARROLLADORES DEL PROYECTOS *PIXY* Y *YASCA*

Christopher Kruegel: es uno de los asesores de *Nenad Jovanovic*, creador del analizador estático *Pixy*.
Pertenece al Laboratorio de Seguridad Informática de la Universidad de Viena.

Entrevista:

Equipo: *We understand you are one of the creators of Pixy. We would like extend this project to PHP 5 but we see no way to make changes to the parser. Can you help us?*

Christopher Kruegel: Well, I was one of the advisors of Nenad, the graduate student who developed Pixy and once he left, detailed knowledge about the tool got lost. Thus you are pretty much on your own with regards to development. Sorry that I don't have better news. You can get access to Pixy's parser here: <http://www.iseclab.org/people/enji/infosys/PhpParser.html>. You will find the grammar files and see that they are for PHP 4. Just update them for PHP 5. Moreover, you will likely need some support for the object oriented features of PHP 5.

Michael V. Scovetta: creador del múltiple analizador estático de código *Yasca*.

Entrevista:

Equipo: *Hello, I was watching Yasca and think it is very interesting. I wonder if there is support for PHP 5.
Thank you.*

Michael V. Scovetta: Thanks! Since I rely on Pixy and PHPLint for most of the PHP scanning, I'm limited to what they can scan. I'll check tonight though to see if either of those handles PHP 5. The grep rules that I wrote should handle PHP 5 just as well as PHP 4, so you definitely get some stuff. I'll let you know what I find out.

ANEXO 7

CÁLCULO DE NIVEL DE RIESGO DEL ANÁLISIS DE FICHEROS

Se parte de la fórmula:

$$\text{Riesgo} = \text{Probabilidad de ocurrencia} * \text{Impacto}$$

Donde:

- Probabilidad de Ocurrencia: es la probabilidad de que una vulnerabilidad se concrete, además de que sea descubierta y explotada por un atacante. En el caso del analizador *Pixy*, la probabilidad de ocurrencia se obtiene por los tipos de funciones, con las siguiente probabilidades:
 - Débiles sanitizadoras: 0.5.
 - Multidependencia e Inversas: 0.6.
 - Malignas: 0.9.
 - Si no existen funciones la probabilidad de ocurrencia de una vulnerabilidad es 1.
- Impacto: se considera en este sentido el impacto técnico (impacto sobre la aplicación) y el impacto sobre el negocio (negocio y compañía de la aplicación). Los niveles de impacto definidos son:
 - Bajo: 1.
 - Medio: 2.

- Alto: 3.
- Si el archivo analizado es de proyecto al impacto se le suma 0.5.

De acuerdo al entorno en que se desarrolla el análisis de vulnerabilidades, la fórmula se modifica de la siguiente forma:

$$\text{Riesgo} = (\text{SUM}(\text{PROM}(\text{Probabilidad de ocurrencia}) * \text{Impacto})/\text{cantVuln})$$

Donde:

- PROM: es el promedio de las probabilidades de ocurrencia de cada función.
- SUM: es la sumatoria del producto de la probabilidad de ocurrencia por el impacto en cada vulnerabilidad.
- cantVuln: es la cantidad de vulnerabilidades del análisis realizado, es un valor distinto de 0.

Las escalas de los valores de riesgo son:

- 0 hasta 1 ---->Bajo.
- Mayor que 1 hasta 2 ---> Medio.
- Mayor que 2 hasta 3 -----> Alto.
- Mayor que 3 ---> Muy Alto.

GLOSARIO DE TÉRMINOS

A

AJAX: Asynchronous JavaScript And XML o JavaScript asíncrono y XML. Es una técnica de desarrollo Web para crear aplicaciones interactivas.

APC: representa un acelerador de aplicaciones *PHP*. Este sistema reserva una zona de la memoria para cachear bytecodes.

C

CPU: Central Processing Unit o Unidad Central de Procesamiento. Es el componente en un ordenador, que interpreta las instrucciones y procesa los datos contenidos en los programas de la computadora.

Cookie: fragmento de información que se almacena en el disco duro del visitante de una página Web a través de su navegador, a petición del servidor de la página.

D

Django: es un Framework de desarrollo Web de código abierto, escrito en Python.

DOM: Document Object Model o Modelo en Objetos para la representación de Documentos, es una interfaz de programación de aplicaciones que proporciona un conjunto estándar de objetos para representar documentos HTML y XML.

E

Falsos negativos: En el análisis de código estático se refiere al término de no detectar vulnerabilidades en el código cuando existen en el mismo.

Falsos Positivos: En el análisis de código estático se refiere al término de detectar una vulnerabilidad en el código cuando no existe en realidad.

Framework: estructura conceptual y tecnológica de soporte definida, normalmente con artefactos de software concretos, mediante la cual otro proyecto de software puede ser organizado y desarrollado.

H

Hacker: Término para designar a alguien con talento, conocimiento, inteligencia e ingenuidad, especialmente relacionada con las operaciones de computadora, redes, seguridad, etc. También se refiere a la persona que disfruta aprendiendo detalles de los sistemas de programación y cómo extender sus capacidades, tan intensamente como, al contrario, muchos usuarios prefieren aprender sólo el mínimo necesario.

Hardware: corresponde a todas las partes físicas y tangibles¹ de una computadora.

Herramientas CASE: herramientas utilizadas para el desarrollo de proyectos de Ingeniería de Software.

HTML: HyperText Markup Language. Lenguaje usado para escribir documentos para servidores World Wide Web.

Http: HyperText Transfer Protocol. Protocolo de Transferencia de Hipertextos. Modo de comunicación para solicitar páginas Web.

Https: es un protocolo de red basado en el protocolo HTTP, destinado a la transferencia segura de datos de hipertexto, es decir, es la versión segura de HTTP.

J

JavaScript: es un lenguaje de scripting basado en objetos, utilizado para acceder a objetos en aplicaciones.

L

Licencia BSD: es la licencia de software otorgada principalmente para los sistemas BSD (Berkeley Software Distribution). Es una licencia de software libre permisiva. Esta licencia permite el uso del código fuente en software no libre.

Licencia GNU: es una licencia creada por la Free Software Foundation en 1989 (la primera versión), y está orientada principalmente a proteger la libre distribución, modificación y uso de software.

Log: en informática es usado para registrar datos o información sobre quién, qué, cuándo, dónde y por qué un evento ocurre para un dispositivo en particular o aplicación.

M

Mod rewrite: es un módulo del servidor Web Apache que permite crear URL alternativas en páginas dinámicas, de forma que sean más fáciles de recordar y también mejor indexadas por los buscadores.

O

OWASP: Open Web Application Security Project o Proyecto de Seguridad de Aplicaciones Web libres. Es un proyecto de código abierto dedicado a determinar y combatir las causas que hacen que el software sea inseguro.

P

Parser: un analizador sintáctico (en inglés parser) es una de las partes de un compilador que transforma su entrada en un árbol de derivación.

php.ini: archivo que contiene la configuración de PHP, con el que se pueden controlar muchos aspectos de su funcionamiento.

Plugin: módulo de hardware o software que añade una característica o un servicio específico a un sistema más grande.

Python: es un lenguaje de programación interpretado.

R

Ruby on Rails: también conocido como RoR o Rails es un Framework de aplicaciones Web de código abierto.

RUP: Rational Unified Process traducido al español, Proceso Unificado de Desarrollo.

S

Sink: se refiere a funciones que imprimen o devuelven resultados en la pantalla de los valores de las variables. Un ejemplo de funciones sink, lo representa *print* en PHP.

Software: se refiere al equipamiento lógico o soporte lógico de una computadora digital.

SQL: Structured Query Language o Lenguaje de consulta estructurado, es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en éstas.

SSL: Secure Socket Layer representa un protocolo criptográfico que proporcionan comunicaciones seguras por una red, comúnmente Internet.

T

TAC: Three Address Code, es una representación de código intermedio usado por compiladores con el objetivo de realizar transformaciones al código provisto y realizar el análisis de forma más sencilla. Cada

instrucción en TAC puede ser descrito por los siguientes elementos: operador, operando 1, operando 2, resultado.

U

UCI: Universidad de las Ciencias Informáticas.

UML: Unified Modeling Language o Lenguaje Unificado de Modelado. Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad.

URL: Uniform Resource Locator o Localizador Uniforme de Recurso. Es una secuencia de caracteres, de acuerdo a un formato estándar, que se usa para nombrar recursos, como documentos e imágenes en Internet, para su localización.

W

Web: World Wide Web o Red Global Mundial. Es un sistema de documentos de hipertexto y/o hipermedios enlazados y accesibles a través de Internet.

Webmetría: Es el estudio de los aspectos cuantitativos de la construcción y uso de los recursos de información, estructuras y tecnologías de una parte concreta de Internet, por regla general a una web o portal, desde perspectivas bibliométricas e informétrica.