Facultad 2

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

Título: Integración de nuevas herramientas de pruebas de seguridad en la Plataforma de Seguridad en las Tecnologías de la Información (PlatSI).

Autores:

Carlos Miguel Medina Aguilera.

Carlos Alejandro Méndez Denis.

Tutores:

Ing. Delís Ise Morales.

Ing. Antonio Hernández Dominguez.

Consultantes:

Ing. Luís Eduardo Gallardo Concepción.

La Habana

Junio 2016

"Pero la juventud tiene que crear.
Una juventud que no crea
es una anomalía realmente."

Declaración de Autoría

Declaramos que somos los únicos autores del trabajo de diploma Integración de nuevas herramientas de pruebas de seguridad en la Plataforma de Seguridad en las Tecnologías de la Información (PlatSI) y autorizamos al Centro de Telemática de la Facultad 2 de la Universidad de 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 año _____.

Autores

Carlos Miguel Medina Aguilera	Carlos Alejandro Méndez Denis

Tutores

Ing. Delís Ise Morales	Ing. Antonio Hernández Dominguez

Consultante

Ing. Luís Eduardo Gallardo Concepción

Datos de Contacto

Carlos Miguel Medina Aguilera

Correo: cmmedina@estudiantes.uci.cu

La Habana, Cuba.

Carlos Alejandro Méndez Denis

Correo: cadenis@estudiantes.uci.cu

La Habana, Cuba.

Delís Ise Morales

Correo: dise@uci.cu

Universidad de las Ciencias Informáticas, La Habana, Cuba.

Antonio Hernández Dominguez

Correo: ahdominguez@uci.cu

Universidad de las Ciencias Informáticas, La Habana, Cuba.

Luís Eduardo Gallardo Concepción

Correo: legallardo@uci.cu

Universidad de las Ciencias Informáticas, La Habana, Cuba.

Agradecimientos

De Carlos Miguel:

Quiero agradecer a mis padres Tania y Miguel Ángel por haberme dado la educación que hoy me convierte en el hombre que soy, por estar ahí presentes para guiarme por el mejor camino posible y por ser los mejores padres del mundo, los quiero mucho.

A mi intranquilo hermano José Ariel, que aunque la mayoría de las veces desatamos una batalla, hay momentos en los que la guerra cesa y nos unimos para lograr nuestros objetivos.

A mis abuelos Aguilera, Lourdes, Eliodoro y Xiomara por ayudar en gran medida a mi crianza, darme todo su cariño y confianza, y guiarme siempre por el buen sendero.

A mis tías y tíos, mis primos y primas, a toda mi familia en general.

A mi compañero de tesis Carlos Alejandro, por ser más que un compañero y un amigo, mi hermano.

A todos mis amigos en general que estuvieron siempre presente en esta larga contienda, en los buenos y malos momentos.

A mis tutores Delís y Antonio por su paciencia y dedicación, sin su guía no hubiéramos podido llegar hasta aquí.

A Gallardo por ayudarnos incondicionalmente, porque aunque estuviera ocupado nunca nos dio un no por respuesta.

A todos los profesores que fueron parte de mi enseñanza y ayudaron a convertirme en el profesional que soy.

A todos muchas gracias.

Agradecimientos

De Carlos Alejandro:

A mi mamá y mi papá por ser los mejores padres del mundo, por estar a mi lado incondicionalmente, gracias por darme la educación que me ha convertido en el hombre que soy hoy, por ser mí modelo a seguir. Son mis héroes, no existen palabras para decirles lo agradecido y orgulloso que estoy de ustedes, los amo.

A mi hermanita Ari, por todo su cariño y preocupación, por ser la mejor hermana del mundo, te quiero mucho.

A mi novia Yanelys por todo el amor que me ha dado, por la paciencia que ha tenido durante estos años de carrera y por acompañarme incondicionalmente en buenos y malos momentos; por sus consejos, por las fuerzas que me ha dado y porque la amo.

A mis abuelos por quererme tanto, y en especial a mi abuelo Pipo por ser como un segundo padre para mí y por aguantar todas mis malacrianzas cuando era un crío, siempre vivirá en mi corazón.

A toda mi familia por su apoyo en todo momento y por estar siempre preocupados por mí.

A mi compañero de tesis Carlos Miguel, por haber compartido conmigo la realización de este trabajo, por ser más que un amigo, mi hermano.

A todos mis amigos, por todos los momentos inolvidables que pasamos juntos y por ayudarme cuando lo necesitaba.

A mis tutores Delís y Antonio por su paciencia y dedicación. Gracias por los conocimientos adquiridos, sin ustedes no podríamos haber llegado hasta aquí.

A Gallardo por ayudarnos incondicionalmente, porque aunque estuviera ocupado nunca nos dio un no por respuesta, muchas gracias.

A mis profesores por ayudar a convertirme en el profesional que soy hoy.

Dedicatoria

De Carlos Miguel:

A mis padres por ser mis ejemplos a seguir.

A mi hermano por los buenos y malos momentos juntos.

A mis abuelos por brindarme todo su cariño y apoyo.

A toda mi familia en general.

A todos mis amigos.

A mí.

De Carlos Alejandro:

A mis padres, mi hermanita, mis abuelos, en general a toda mi familia.

A mi novia, Yanelys.

A todos mis amigos.

A mí.

Resumen

Con el auge de Internet, las aplicaciones, datos e información tienen un grado de exposición cada vez mayor, por lo que es de vital importancia para las instituciones proteger y controlar el uso de la misma. El presente trabajo realiza un estudio sobre las herramientas de pruebas de seguridad a aplicaciones web existentes en la actualidad, con la finalidad de seleccionar las posibles a integrar a PlatSI.

La Plataforma para la realización de auditorías tiene integradas cinco herramientas de pruebas de seguridad, ellas son: “Acunetix”, “Nikto”, “Arachni”, “ZAP” y “W3af”’. De estas herramientas no se cuenta con las versiones actualizadas, limitando así la posibilidad de detectar nuevos tipos de vulnerabilidades, y al no contar con varios tipos de herramientas limita el proceso de correlación, lo que aumenta la probabilidad de ocurrencia de falsos positivos. Además la Plataforma no posee herramientas especialidades en la detección de vulnerabilidades a aplicaciones web desarrolladas en CMS. Con el objetivo de apoyar la detección de nuevas vulnerabilidades de PlatSI, se seleccionaron las herramientas Wapiti y Joomscan para formar parte de la propuesta de solución. En la integración de estas herramientas fue necesario la implementación de un mecanismo de plugins para acoplar y no afectar el funcionamiento de los demás procesos que realiza la Plataforma. Como resultado del presente trabajo se integraron dos herramientas de pruebas de seguridad a aplicaciones web a la Plataforma, que apoyan la detección de nuevas vulnerabilidades, y por consiguiente el proceso de auditorías.

PALABRAS CLAVES

PlatSI, plugins, vulnerabilidades, herramientas, aplicaciones web.

Índice

Índice de Tablas

Índice de Ilustraciones

Introducción

Para el hombre, la seguridad de la información tiene un efecto significativo respecto a su privacidad. En la actualidad, con el desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC), se pone en manos de la sociedad gran cantidad de aplicaciones y sistemas informáticos que apoyan los procesos que se realizan en las empresas, instituciones e incluso en la vida privada de las personas. Con el auge de Internet, las aplicaciones, datos e información tienen un grado de exposición cada vez mayor, esto se evidencia en el informe realizado por la Symantec Corporation[1] donde en el 2015 detectó más de 430 millones de ejemplos nuevos y diferentes de software malicioso (1).

Cuba, a pesar de sus limitaciones tecnológicas, se encuentra inmersa en el proceso de desarrollo de las TIC. Actualmente se lleva a cabo un plan de informatización, en función de desarrollar y modernizar todas las esferas de la sociedad, así como la utilización y acceso masivo a las TIC por parte de los ciudadanos (2). Es de vital importancia para las instituciones garantizar la seguridad de la información, concebida como el conjunto de medidas técnicas, organizativas y legales que permiten a las organizaciones y sistemas tecnológicos proteger la información (3). Por dicha razón en el país existen numerosas entidades dedicadas a desarrollar productos y servicios informáticos capaces de apoyar en un amplio porciento la seguridad de la información.

La Universidad de las Ciencias Informáticas (UCI), es una institución que tiene como misión formar profesionales comprometidos con su Patria y altamente calificados en la rama de la Informática. Producir aplicaciones y servicios informáticos, a partir de la vinculación estudio-trabajo como modelo de formación. Servir de soporte a la industria cubana de la informática (4). La universidad cuenta con un conjunto de áreas vinculadas a las actividades docentes, productivas, investigativas y de servicios.

El Centro de Telemática (TLM) de la Facultad 2 es un centro de desarrollo de sistemas y servicios informáticos en las ramas de las Telecomunicaciones y la Seguridad Informática (5). En dicho centro se desarrolla el proyecto Plataforma de Seguridad en las Tecnologías de la Información (XILEMA-PlatSI), dirigido a la realización de auditorías a aplicaciones web con el fin de detectar brechas de seguridad ante posibles ataques. La Plataforma está concebida para los siguientes módulos:

- **Módulo de Auditorías a Aplicaciones Web (MAAWeb):** Permite realizar pruebas de seguridad a aplicaciones web para detectar vulnerabilidades ante posibles ataques, el cual está integrado a la Plataforma.

[1] Symantec Corporation es una corporación internacional que desarrolla y comercializa software para computadoras, particularmente en el dominio de la seguridad informática.

- **Módulo de Auditorías a Bases de Datos (MABD):** Realiza pruebas de seguridad a bases de datos, el cual está implementado completamente, pero no se ha integrado a la Plataforma.
- **Módulo de Auditorías a Sistemas Operativos (MASO):** Realiza auditorías a sistemas operativos. Aún no se ha implementado y por consiguiente no se encuentra integrado a la Plataforma.

La Plataforma realiza un proceso de correlación que permite verificar y comparar las vulnerabilidades detectadas, con las registradas en la base de datos. Si la brecha de seguridad detectada no se encuentra almacenada, es ubicada en una lista independiente para ser revisada por el especialista, y si representa una amenaza real para la aplicación web a la que se le realizó la prueba, el especialista tiene la posibilidad de editarla e incorporarla de forma manual a la base de conocimientos.

El Módulo de Auditorías a Aplicaciones Web actualmente tiene integradas cinco herramientas de pruebas de seguridad, ellas son: “Acunetix”, “Nikto”, “Arachni”, “ZAP” y “W3af”’. De las mismas no se cuenta con las últimas versiones actualizadas, limitando la posibilidad de detectar nuevos tipos de vulnerabilidades. Además la Plataforma no cuenta con herramientas especializadas en la detección de vulnerabilidades a aplicaciones web desarrolladas en CMS[2].

Al no contar con varios tipos de herramientas se puede ver afectado el proceso de correlación, el mismo permite verificar y comparar las vulnerabilidades detectadas, con las registradas en la base de datos. Por tanto las brechas de seguridad que son detectadas por varias herramientas, disminuye la probabilidad de ocurrencia de falsos positivos[3].

Por lo anteriormente planteado surge como **problema a resolver:** ¿Cómo apoyar el proceso de detección de vulnerabilidades a aplicaciones web de PlatSI?

Como **objeto de estudio** de la presente investigación se plantea: Proceso de integración de herramientas a PlatSI. Enmarcada la investigación en el **campo de acción**: Integración de nuevas herramientas a PlatSI en el MAAWeb.

Para dar solución al problema expuesto anteriormente se traza como **objetivo general**: Integrar nuevas herramientas de pruebas de seguridad a PlatSI mediante plugins[4] para apoyar el proceso de detección de vulnerabilidades a aplicaciones web.

Para dar cumplimiento al objetivo general se plantearon las siguientes **tareas de la investigación**:

[2] Es un sistema de gestión de contenidos por sus siglas en inglés: Content Management System (CMS) es un programa informático que permite crear una estructura de soporte (framework) para la creación y administración de contenidos, principalmente en páginas web.
[3] Vulnerabilidades detectadas por una herramienta que en realidad no representan una amenaza.
[4] Es un complemento generalmente pequeño que se utiliza para brindar funcionalidades específicas a un sistema o herramienta de software.

- Análisis de las características de la Plataforma para guiar el desarrollo de los plugins correspondientes a las herramientas a integrar.
- Caracterización de herramientas de pruebas de seguridad a aplicaciones web para analizar sus principales funcionalidades.
- Comprobación de las funcionalidades que brindan las herramientas para evaluar su posible utilización en la detección de vulnerabilidades.
- Selección de herramientas de pruebas de seguridad a aplicaciones web para utilizarlas en la propuesta de solución.
- Implementación de plugins para integrar a PlatSI las herramientas de pruebas de seguridad seleccionadas.
- Definición de una estrategia de prueba para verificar el funcionamiento de los plugins integrados a la Plataforma.

Para darle cumplimiento a esta investigación fueron utilizados diferentes **métodos de investigación**, los cuales se especifican a continuación:

Métodos Teóricos:

- **Analítico-Sintético**: Se utilizó para el análisis de materiales bibliográficos, herramientas, tecnologías y lenguajes necesarios para desarrollar los plugins para integrar las herramientas a PlatSI. También permitió seleccionar la metodología de desarrollo más apropiada para darle solución a la investigación.

- **Histórico-Lógico:** Se empleó en la investigación para estudiar las herramientas de pruebas de seguridad a aplicaciones web existentes en la actualidad.

- **Modelación:** Se utilizó con el objetivo de realizar una representación de varios procesos que se ejecutan en la Plataforma, a través de modelos y diagramas que apoyan la propuesta de solución.

Métodos Empíricos:

- **Observación**: Se utilizó para observar y analizar el comportamiento de las herramientas de pruebas de seguridad a aplicaciones web seleccionadas en el proceso de investigación para integrar a PlatSI.
- **Entrevista**: Permitió realizar conversaciones con especialistas de Seguridad Informática y de PlatSI para obtener información necesaria acerca de las herramientas de pruebas a aplicaciones web. El tipo de entrevista realizada fue la informal; no se definieron fechas para la misma y las preguntas no fueron planificadas, sino que se desarrollaron de acuerdo a la fluidez de la conversación.

El presente documento está estructurado en 4 capítulos:

Capítulo 1. Fundamentación Teórica:

En este capítulo se presenta la definición conceptual del marco teórico de la investigación. Incluye el estudio del estado del arte realizado sobre las diferentes herramientas de pruebas de seguridad, seleccionando así las que cumplan con las características esenciales propuestas en el proceso de investigación. Además, se exponen la metodología, herramientas, lenguajes y tecnologías que serán utilizadas en la propuesta de solución.

Capítulo 2. Exploración y Planificación:

En este capítulo se presenta la propuesta de solución. Se definen los requerimientos del sistema y la lista de reserva del producto. Además muestra las fases iniciales de la metodología de desarrollo utilizada, en la que se definen las Historias de Usuario (HU) asociada a cada herramienta de prueba de seguridad.

Capítulo 3. Diseño e Implementación:

En este capítulo se presenta la arquitectura de la Plataforma, se definió el patrón arquitectónico y los patrones de diseño utilizados. También se representaron las tarjetas Clase – Responsabilidad – Colaborador (CRC) y se especificaron las Tareas de Ingeniería para describir en detalles cada una de las Historias de Usuario. Además, se definieron los estándares de codificación utilizados en la implementación de los plugins.

Capítulo 4. Pruebas:

En este capítulo se presentan los tipos de pruebas realizados para verificar el funcionamiento de las herramientas integradas a la Plataforma. Se implementaron las pruebas unitarias, pruebas de integración y pruebas de aceptación.

1. Introducción

En el presente capítulo se abordan los principales conceptos utilizados a lo largo de la investigación. Se define la metodología, herramientas y tecnologías utilizadas para dar cumplimiento al objetivo general del presente trabajo. Además, se describen las pruebas realizadas con las herramientas que fueron descargadas y se justifica la selección de las que forman parte de la solución.

1.1. Marco conceptual de la investigación

Con el auge de Internet, la seguridad de las redes es cada vez más indispensable, pero existen diferentes variables que atentan contra la seguridad informática las cuales pueden ser: desconocimiento de los usuarios, gran nivel de interconexión de las redes, software malicioso y hackers.

La seguridad informática es un conjunto de medidas de prevención, detección y corrección, orientadas a proteger los pilares confidencialidad, integridad y disponibilidad de los activos de información (6). Se conoce como confidencialidad a la característica que asegura que los usuarios no tengan acceso a los datos a menos que estén autorizados para ello. Por otro lado, la integridad indica que toda modificación de la información solo es realizada por usuarios autorizados, por medio de procesos autorizados. La disponibilidad garantiza que los recursos del sistema y la información estén disponibles únicamente para usuarios autorizados en el momento que los necesiten (7).

En las diferentes empresas e instituciones se aplican las políticas y medidas de Seguridad Informática necesarias para garantizar los pilares de la misma. Existen diferentes herramientas de pruebas de seguridad a aplicaciones web para el escaneo y detección de vulnerabilidades. Por tanto una vulnerabilidad es considerada una debilidad que posee una aplicación, que puede ser utilizada para causar algún daño (8).

Las vulnerabilidades que mayor impacto ocasionan en aplicaciones web son las siguientes (9):

- **Inyección:** Las fallas de inyección, tales como Structured Query Language (SQL), y Lightweight Directory Access Protocol (LDAP), ocurren cuando datos no confiables son enviados a un intérprete como parte de un comando o consulta.
- **Pérdida de Autenticación y Gestión de Sesiones:** Las funciones de la aplicación relacionadas a autenticación y gestión de sesiones son frecuentemente implementadas incorrectamente.
- **Secuencia de Comandos en Sitios Cruzados (XSS):** Las fallas XSS ocurren cada vez que una aplicación toma datos no confiables y los envía al navegador web sin una validación y codificación apropiada.

- **Referencia Directa Insegura a Objetos:** Una referencia directa a objetos ocurre cuando un desarrollador expone una referencia a un objeto de implementación interno, tal como un fichero, directorio, o base de datos.
- **Configuración de Seguridad Incorrecta:** Una buena seguridad requiere tener definida e implementada una configuración segura para la aplicación, marcos de trabajo, servidor de aplicación, servidor web, base de datos, y plataforma.
- **Exposición de datos sensibles:** Muchas aplicaciones web no protegen adecuadamente datos sensibles tales como números de tarjetas de crédito o credenciales de autenticación.
- **Ausencia de Control de Acceso a Funciones:** La mayoría de aplicaciones web verifican los derechos de acceso a nivel de función antes de hacer visible en la misma interfaz de usuario. A pesar de esto, las aplicaciones necesitan verificar el control de acceso en el servidor cuando se accede a cada función.
- **Falsificación de Peticiones en Sitios Cruzados (CSRF):** Un ataque CSRF obliga al navegador de una víctima autenticada a enviar una petición HTTP (Hypertext Transfer Protocol) falsificado, incluyendo la sesión del usuario y cualquier otra información de autenticación incluida automáticamente, a una aplicación web vulnerable.
- **Utilización de componentes con vulnerabilidades conocidas:** Algunos componentes tales como las librerías, los frameworks y otros módulos de software casi siempre funcionan con todos los privilegios. Si se ataca un componente vulnerable esto podría facilitar la intrusión en el servidor o una perdida seria de datos.
- **Redirecciones y reenvíos no validados:** Las aplicaciones web frecuentemente redirigen y reenvían a los usuarios hacia otras páginas o sitios web, y utilizan datos no confiables para determinar la página de destino.

Las herramientas de pruebas de seguridad pueden detectar estas vulnerabilidades mencionadas y generan un informe con los resultados sin realizar ninguna acción sobre la aplicación web analizada, estos se clasifican en (10):

- **Falsos Positivos:** Cuando una herramienta detecta una vulnerabilidad que en realidad no se considera una amenaza para la aplicación web analizada.
- **Falsos Negativos:** Cuando una herramienta no detecta una vulnerabilidad que realmente es una amenaza para la aplicación web analizada.

Con el objetivo de seleccionar las herramientas más factibles a integrar a la Plataforma y conociendo que cada una de ellas cuenta con características diferentes, el equipo de desarrollo de PlatSI definió cuatro características esenciales para realizar el proceso de selección, ellas son:

- **Descargable:** La herramienta se debe descargar para poder realizar pruebas de seguridad a aplicaciones web de forma práctica y evaluar su integración en PlatSI.

- **Recibir parámetros a través de la vía no estándar:** Permite ejecutar una herramienta a través de otra aplicación estableciéndole a su vez los parámetros de la ejecución.
- **Exportar resultados en formato de texto con una estructura definida:** Ejemplos de estos formatos pueden ser XML[5], JSON[6] y HTML[7].
- **Licencia libre:** De ser posible, el nivel de adquisición de las herramientas no debe ser mediante licencias privativas, se pretende utilizar herramientas de código abierto.

1.2. Herramientas de pruebas de seguridad para aplicaciones web

El proceso de investigación estuvo guiado por las cuatro características definidas por el equipo de desarrollo. En este proceso se estudiaron 22 herramientas, de las cuales se descartaron un total de 12 por ser propietarias lo que impidió la obtención de las mismas, esta información está reflejada en el Anexo I. A continuación se describen las pruebas realizadas con cada una de las herramientas descargadas y además se justifican las que fueron seleccionadas a formar parte de la solución:

1.2.1. WebCruiser

Es una eficaz y poderosa herramienta de pruebas de penetración web que apoya la auditoría de sitios web en busca de vulnerabilidades: Inyección SQL, Cross Site Scripting, inclusión de archivos locales, la inclusión de archivos remotos, redirigida etc. El rasgo más característico de WebCruiser es que se centra en las vulnerabilidades de alto riesgo, y se puede escanear un tipo señalado de vulnerabilidad, o una URL determinada, o una página por separado. Esta herramienta está disponible para el sistema operativo Windows y los resultados se pueden obtener en el formato HTML (11).

Descripción de las pruebas

Para realizar las pruebas con WebCruiser en su versión V3.5.3, se usó una edición libre descargada para el sistema operativo Windows. A continuación se especifican los pasos, donde se describe el proceso y los resultados que se obtuvieron. La herramienta se opera mediante una interfaz gráfica. Para realizar la prueba, se utilizó el ambiente de pruebas: Aplicación Web Vulnerable, en inglés Damn Vulnerable Web App (DVWA).

Dirección IP o URL: http://127.0.0.1/dvwa/

Descripción: Una vez abierta la aplicación, se introdujo la dirección IP o URL en el campo de texto que lo indica, para así abrir a través de la herramienta la aplicación web a la que se le va a realizar

[5] Por sus siglas en inglés de Extensible Markup Language (lenguaje de marcas extensible), es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible.
[6] Acrónimo de JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos.
[7] Por sus siglas en inglés de Hypertext Markup Language (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web.

la prueba. Se inició la prueba mediante la opción ScanSite que aparece en la parte superior derecha de la herramienta. A medida que se fue ejecutando el proceso se fueron visualizando las vulnerabilidades detectadas. Luego de unos minutos a que culminara el ataque, se pudieron observar los resultados obtenidos y su clasificación.

Resultados: Se obtuvo un total de 11 alertas, clasificadas en tipo de vulnerabilidad: 7 SQL INJECTION, 3 Cross-Site Scripting (XSS), 1 Local File Inclusion (LFI), y la dirección URL donde se encuentra cada una de estas.

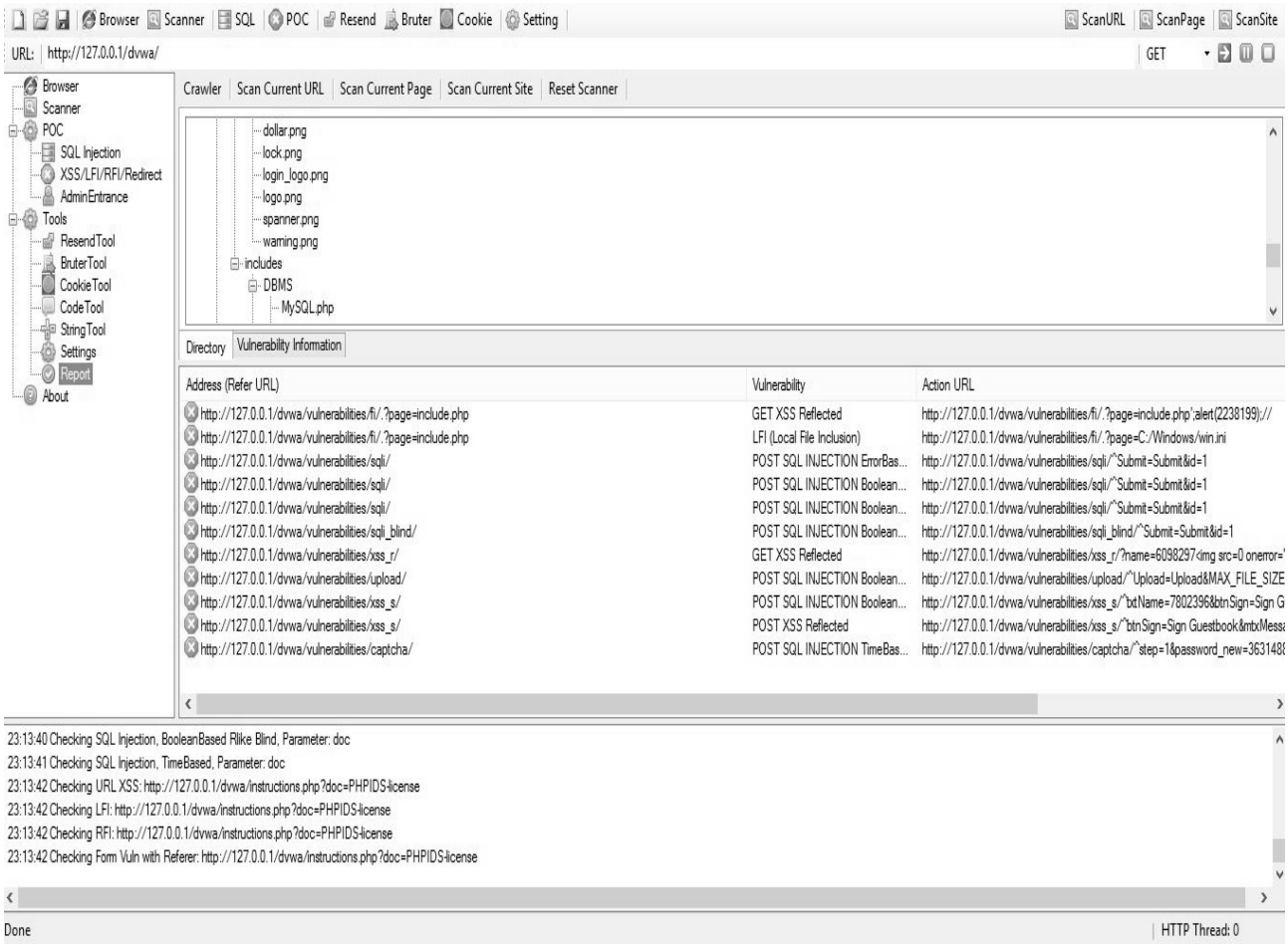


Figura 1: Ejecución de la herramienta WebCruiser.

La figura 1 muestra las vulnerabilidades detectadas por la herramienta WebCruiser, especificando en cada caso el tipo de amenaza y la dirección URL donde se encuentra. Se pudo obtener un reporte en formato HTML en el cual se especifican los resultados obtenidos, este se puede observar en la figura 2.

Vulnerability Result

No.	1
ReferURL	http://127.0.0.1/dvwa/vulnerabilities/fi/.?page=include.php
ActionURL	http://127.0.0.1/dvwa/vulnerabilities/fi/.?page=include.php'; alert(2238199);//
Parameter	page
Vulnerability	GET XSS Reflected

No.	2
ReferURL	http://127.0.0.1/dvwa/vulnerabilities/fi/.?page=include.php
ActionURL	http://127.0.0.1/dvwa/vulnerabilities/fi/.?page=C:/Windows/win.ini
Parameter	page
Vulnerability	LFI (Local File Inclusion)

No.	3
ReferURL	http://127.0.0.1/dvwa/vulnerabilities/sqli/
ActionURL	http://127.0.0.1/dvwa/vulnerabilities/sqli/^Submit=Submit&id=1
Parameter	id
Vulnerability	POST SQL INJECTION ErrorBased Integer

Figura 2: Reporte de la herramienta WebCruiser.

A pesar de ser una excelente herramienta para la detección de vulnerabilidades y exportar los resultados en formato estructurado, no se decide incluir WebCruiser en la solución porque no recibe parámetros por la vía no estándar, lo que implica que no se puede ejecutar la herramienta mediante otra aplicación.

1.2.2. Syhunt Hybrid

Syhunt inyecta de forma dinámica los datos en aplicaciones web y analiza la respuesta de la aplicación con el fin de determinar si el código de la aplicación es vulnerable, la automatización de las pruebas de seguridad de aplicaciones web y la forma proactiva que guarda la infraestructura de Internet de su organización contra varios tipos de amenazas a la seguridad de aplicaciones web. Diseñado para escanear aplicaciones web para varios tipos de problemas, tales como Cross-Site Scripting (XSS), Inclusión, inyección SQL, ejecución de comandos y validación débil, el explorador de código es un complemento perfecto a la ya amplia gama de capacidades de exploración remotos disponibles en el escáner dinámico (12).

Descripción de las pruebas

Para realizar las pruebas con Syhunt Hybrid se obtuvo una distribución de la comunidad en su versión V5.2.0.1 para el sistema operativo Windows. La herramienta se opera mediante una interfaz gráfica y tiene la opción de analizar a través de una consola. Para realizar la prueba, se utilizó el ambiente de pruebas: Aplicación Web Vulnerable, en inglés Damn Vulnerable Web App (DVWA).

Dirección IP o URL: http://127.0.0.1/dvwa/

Descripción: Luego de ejecutada la aplicación se escoge la opción (Sandcat Console) para realizar el escaneo a través de la consola. Se introdujo el comando *help* (Muestra un listado de los comandos disponibles). Seguido el comando *load [url]* (Ej. *load http://127.0.0.1/dvwa/*) para cargar en la herramienta la aplicación web a la que se le va a realizar la prueba y luego se procede al escaneo a través del comando *appscan*. Durante el transcurso de este proceso se fueron visualizando las vulnerabilidades detectadas, la línea donde se encuentran y el riesgo que tiene para la aplicación web.

Resultados: La herramienta se mantuvo analizando 18 horas y no finalizó el proceso de detección de vulnerabilidades. Durante este tiempo se detectaron un total de 18 amenazas, con riesgo medio y ubicado en la línea 200.

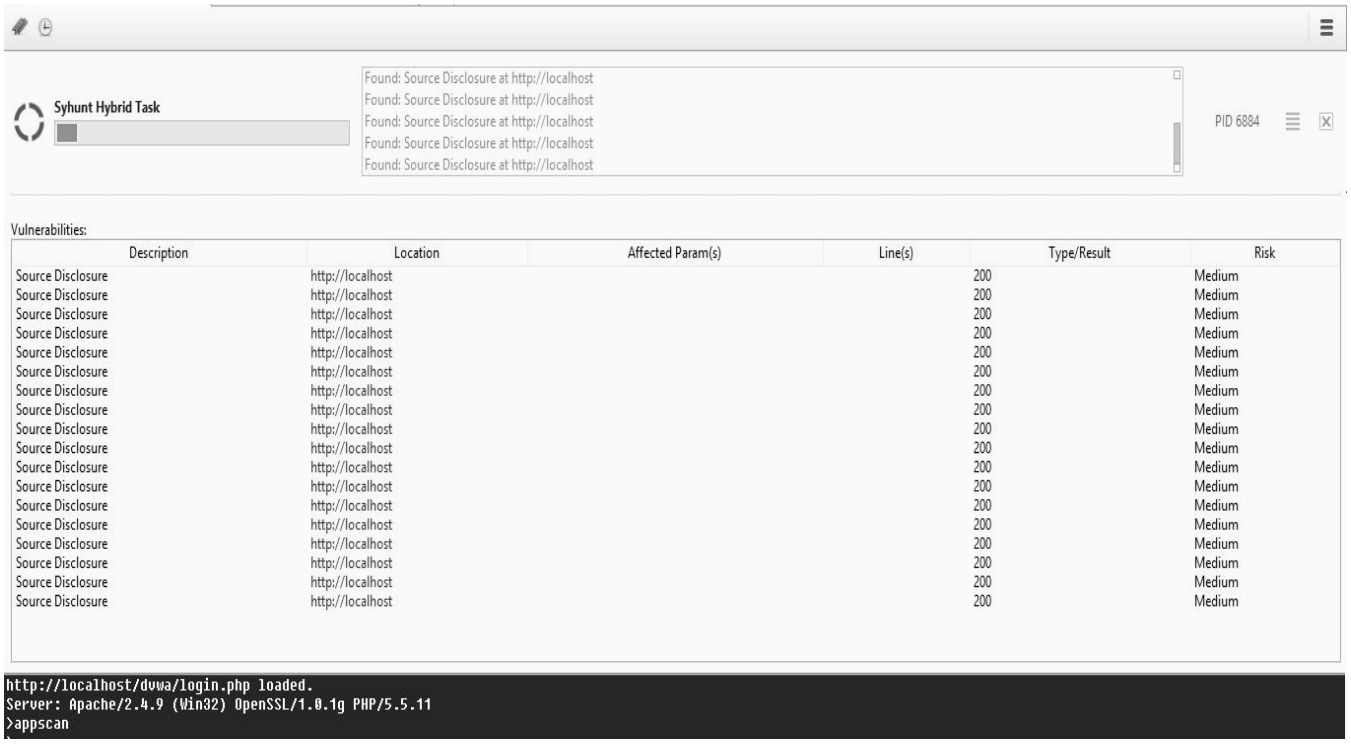


Figura 3: Resultados de la herramienta Syhunt.

La herramienta Syhunt no se incluye en la solución ya que nunca terminó el análisis, como se puede observar en la figura 3.

1.2.3. Netsparker

Desarrollada por una empresa desarrolladora de productos de análisis de vulnerabilidades web, llamada Mavituna Security. Entre las principales características que posee se anuncia que está libre de reportar falsos positivos, o lo que es lo mismo, identificar como vulnerabilidades peticiones que realmente no lo son. Aunque esta característica tiene una especificación, donde las que no puede garantizar como vulnerables las etiqueta como "posibles", para que posteriormente el auditor haga las comprobaciones oportunas. Detecta los riesgos más comunes y los resultados se pueden obtener en diferentes formatos (XML, PDF, RTF). Es una herramienta de uso privativo y solo se conoce de la existencia de versiones en el sistema operativo Windows (13).

Descripción de las pruebas

Para realizar pruebas con Netsparker se utilizó la distribución profesional en su versión v3.2.1.0 para el sistema operativo Windows. La herramienta se opera mediante una interfaz gráfica. Para realizar las pruebas, se utilizó el ambiente de pruebas: Aplicación Web Vulnerable, en inglés Damn Vulnerable Web App (DVWA).

Dirección IP o URL: http://127.0.0.1/dvwa/

Descripción: Una vez abierta la aplicación, se selecciona la opción Start New Scan en la parte superior izquierda de la herramienta. Luego se introduce la dirección IP o URL en el campo de texto que lo indica y se inicia la prueba mediante la opción Start Scan. A medida que se fue completando el proceso se fueron visualizando las vulnerabilidades detectadas. Fue necesario esperar unos minutos a que culminara el ataque. Una vez finalizado se pudieron observar los resultados obtenidos y su clasificación.

Resultados: Se obtuvo un total de 31 alertas clasificadas en: 1 IMPORTANT, 8 LOW y 22 INFORMATION.

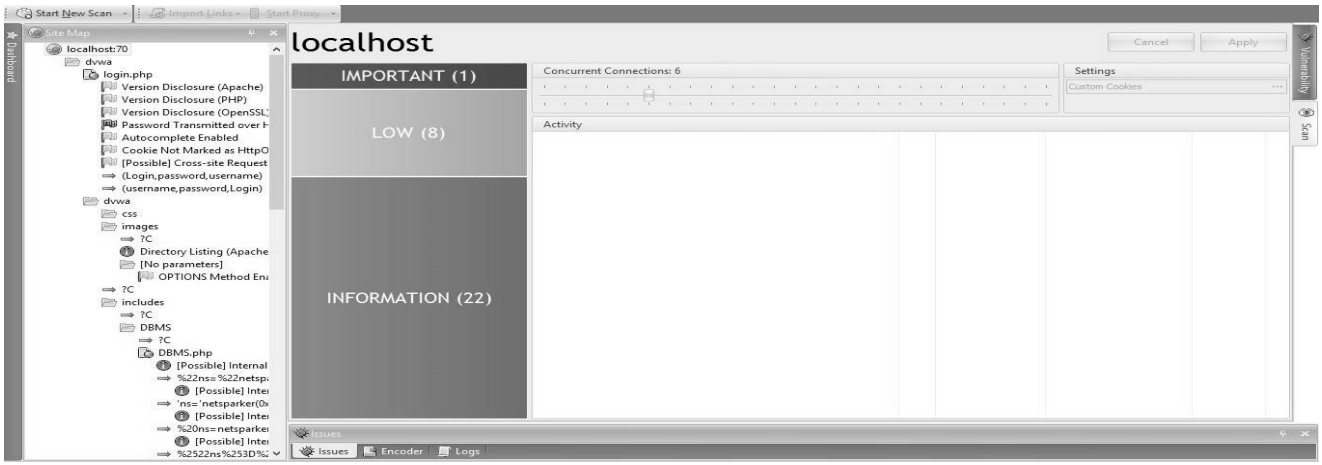


Figura 4: Resultados de la herramienta Netsparker.

La figura 4 muestra las vulnerabilidades detectadas por la herramienta Netsparker, de color rojo la cantidad de alertas de tipo Importante, de amarillo las clasificadas como Baja y las de azul las de tipo Informacionales. Brinda la posibilidad de visualizar y obtener un informe de resultados en los formatos XML, HTML y CSV, el cual se muestra en la figura 5:

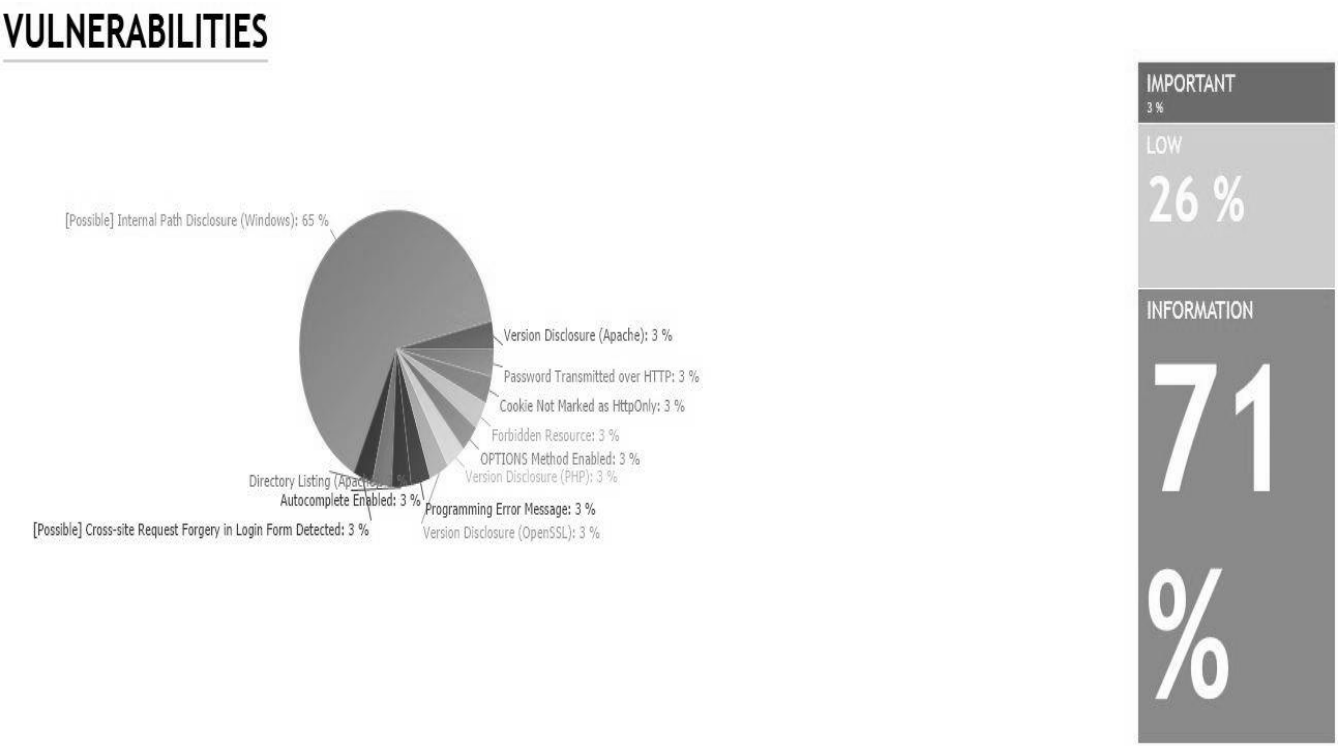


Figura 5: Resultados de la herramienta Netsparker.

A pesar de ser una herramienta desarrollada por una compañía dedicada a la seguridad informática y específicamente a realizar productos de análisis de vulnerabilidades web (Mavituna Security), se decidió no seleccionar a Netsparker porque es una herramienta propietaria y no recibe parámetros por la vía no estándar, lo que implica que no se puede ejecutar la herramienta mediante otra aplicación.

1.2.4. CMS-Explorer

Herramienta desarrollada en Perl[8] que muestra los plugins, módulos, temas y componentes que se encuentran corriendo en una web CMS de tipo Drupal, WordPress, Joomla! y Mambo. Como opciones interesantes permite utilizar un proxy para realizar modificaciones en el envío de solicitudes y realizar una comprobación de posibles vulnerabilidades. Además, CMS-Explorer se puede utilizar para ayudar en las pruebas de seguridad (14).

Descripción de las pruebas

[8] Lenguaje de programación de propósito general originalmente desarrollado para la manipulación de texto y que ahora es utilizado para un amplio rango de tareas incluyendo administración de sistemas, desarrollo web, programación en red.

Para realizar pruebas con CMS-Explorer se utilizó una versión libre para el Sistema Operativo Linux. La herramienta se opera mediante la consola de comandos. A continuación se describe el proceso y los resultados obtenidos luego de realizada la auditoría. Se utilizó para realizar las pruebas un sitio sencillo hecho en el CMS Joomla.

Dirección IP o URL: http://localhost/Joomla/

Descripción: Para realizar la prueba se hizo uso de la consola de comandos, en la que se introdujo antes de realizar la prueba el comando: *./cms-explorer.pl –h* (Muestra una ayuda de cómo pueden ser utilizadas las opciones que posee la herramienta). Luego para realizar la prueba se introdujo en la consola el comando: *./cms-explorer.pl –url http://localhost/Joomla/ -type Joomla* donde se especifica la url a la que se le va a realizar la prueba y el tipo de CMS a analizar, culminando así el análisis en unos pocos minutos.

Resultados: Se visualizan una serie de plugins vulnerables tras la auditoría realizada pero no se obtuvo el reporte de vulnerabilidades ya que la herramienta no cuenta con una opción para mostrar en ningún formato el reporte.

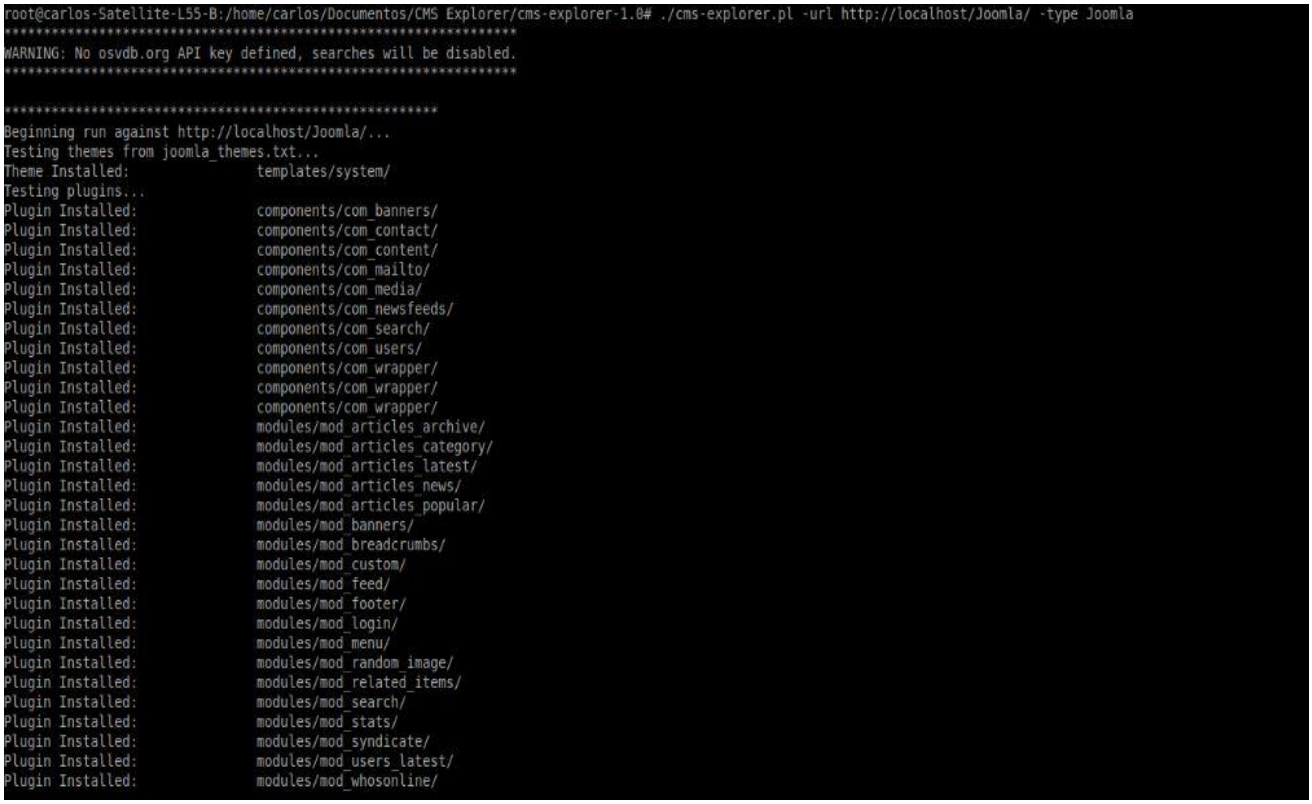


Figura 6: Resultados de la herramienta CMS-Explorer.

La herramienta CMS-Explorer no se incluye en la solución porque no exporta en un formato estructurado el reporte de los plugins vulnerables detectados.

1.2.5. Fulgur Sec - WordPress Security Fingerprinter (FS-WPSFp)

Herramienta utilizada para encontrar plugins instalados vulnerables en WordPress. También encuentra vulnerabilidades según la versión del CMS WordPress analizado. Útil para pruebas de penetración y así comprobar si nuestra página en WordPress es vulnerable para ataques hacking. La versión actual cuenta con 240 pruebas de seguridad (15).

Descripción de las pruebas

Para realizar pruebas con FS-WPSFp se utilizó la versión v1.0 libre para el Sistema Operativo Linux. La herramienta se opera mediante la consola de comandos. A continuación se describe el proceso y los resultados obtenidos luego de realizada la auditoría. Se utilizó para realizar las pruebas un sitio sencillo realizado en el CMS WordPress.

Dirección IP o URL: http://localhost/wordpress/

Descripción: Para realizar la prueba se hizo uso de la consola de comandos, en la que se introdujo para realizar la prueba el comando: *./FS-WPSFp_v1.0_240 http://localhost/wordpress* donde se especifica la url a la que se le va a realizar la prueba.

Resultados: Luego de terminada la auditoría se detectaron una lista de cuatro plugins vulnerables pero no se pudo obtener el reporte del análisis.

```
root@carlos-Satellite-L55-B:/home/carlos/Documentos/FS-WPSFp_v1.0_240# ./FS-WPSFp_v1.0_240 http://localhost/wordpress/
[1;3;J

 FS-WPSFp - v1.0 [public] - 240

-================================================-

  FULGUR SEC - WORDPRESS SECURITY FINGERPRINTER

  BY DR. ALBERTO FONTANELLA - WWW.FULGURSEC.COM

-================================================-


-========== [         INFO         ] ===========-

 [+] HOST: http://localhost/wordpress/
 [+] VULN PLUGINS: 240

 [+] WP VERSION: 4.4.1


-========== [  EXPLOIT WP VERSION  ] ===========-

 [+] EXPLOIT: 0 (None)


-========== [ EXPLOIT VULN PLUGINS ] ===========-

 [+] TESTED PLUGINS:  240 / 240


-========== [       SUMMARY        ] ===========-

 [+] REPORT - [  jue may 26 22:54:13 EDT 2016  ]

 [+] HOST: http://localhost/wordpress/
 [+] VULN PLUGINS: 4
 [+] WP VERSION: 4.4.1

    [0] wp-admin/edit-tags.php
        EXPLOIT: http://www.exploit-db.com/exploits/17465/

    [1] wp-admin/link-manager.php
        EXPLOIT: http://www.exploit-db.com/exploits/17465/

    [2] wp-admin/post.php
        EXPLOIT: http://www.exploit-db.com/exploits/15858/

    [3] wp-includes/comment.php
        EXPLOIT: http://www.exploit-db.com/exploits/15684/
```

Figura 7: Resultados de la herramienta FS-WPSFp.

La herramienta FS-WPSFp no se incluye en la solución porque no se obtiene un reporte en formato estructurado del análisis.

1.2.6. Joomscan

Es un escáner de vulnerabilidades orientado al CMS Joomla que forma parte de OWASP[9] y que ha sido desarrollado en Perl para automatizar la búsqueda de vulnerabilidades SQL injection, LFI, RFI, XSS. Este escáner de seguridad ayuda en gran medida a los desarrolladores y web masters que utilizan este CMS. Gracias a esta herramienta es posible detectar los errores y las vulnerabilidades de una web desarrollada con el gestor de contenidos Joomla (16).

Descripción de las pruebas

Para realizar pruebas con Joomscan se utilizó la versión v0.0.4 libre para el Sistema Operativo Linux. La herramienta se opera mediante la consola de comandos. A continuación se describe el proceso y los resultados obtenidos luego de realizada la auditoría. Se utilizó para realizar las pruebas un sitio sencillo hecho en el CMS Joomla.

Dirección IP o URL: http://localhost/Joomla/

Descripción: Se instalaron varias librerías de Perl que requiere la herramienta y luego fue instalada la herramienta. A continuación se introdujo en la consola el comando: *perl joomscan.pl -h* (Muestra una ayuda de las opciones que se pueden utilizar en la herramienta). Para realizar la auditoría se introdujo en la consola el comando: *perl joomscan.pl –u http://localhost/Joomla/* especificando la url a la que se le va a realizar la prueba. Se esperaron unos minutos a que terminara el proceso de ejecución.

Resultados: Se detectaron un total de 28 posibles ítems vulnerables, y dos ítems vulnerables de alto riesgo. Se obtuvo un reporte de las vulnerabilidades detectadas luego de finalizada la auditoría.

[9]Por sus siglas en inglés Open Web Application Security Project es un Proyecto Abierto de Seguridad de Aplicaciones Web dedicado a determinar y combatir las causas que hacen que el software sea inseguro.

Figura 8: Ejecución de la herramienta Joomscan.

En la figura 8 se muestra la ejecución de la herramienta Joomscan. Esta herramienta brinda la posibilidad de exportar un informe de resultados en el formato HTML, el cual se puede observar en la figura 9 que se muestra a continuación:

Figura 9: Reporte de la herramienta Joomscan.

Se seleccionó Joomscan para la integración por ser una herramienta de licencia libre. Recibe parámetros por la vía no estándar, por lo que puede realizar la prueba desde otra aplicación a través de la consola de comandos. Además está desarrollada por el proyecto OWASP, una comunidad abierta y libre de nivel mundial, enfocada en mejorar la seguridad en las aplicaciones web. Aparte

de su funcionalidad básica de detección de vulnerabilidades, ofrece la posibilidad de generar informes en el formato HTML.

1.2.7. WordPress Security Scanner (WPScan)

Herramienta de seguridad de código abierto distribuida bajo licencia GPLv3 que permite, a los profesionales de la seguridad y a los administradores web, evaluar las condiciones de seguridad de un sitio web mediante la exploración de las diversas vulnerabilidades conocidas dentro de una instalación de WordPress. La herramienta está escrita en Ruby, permite encontrar vulnerabilidades en los plugins instalados, buscando entre los 2220 plugins más populares y lista las vulnerabilidades encontradas en base a las versiones (17).

Descripción de las pruebas

Para realizar pruebas con WPScan se utilizó la versión v2.9 libre para el Sistema Operativo Linux. La herramienta se opera mediante la consola de comandos. A continuación se describe el proceso y los resultados obtenidos luego de realizada la auditoría. Se utilizó para realizar las pruebas un sitio sencillo hecho en el CMS WordPress.

Dirección IP o URL: http://localhost/WordPress/

Descripción: Se introdujo en la consola para comenzar la prueba el comando: *ruby wpscan.rb --url http://localhost/wordpress/ --enumerate p* especificando la dirección url a la que se le va a realizar la auditoría.

Resultados: Se obtuvieron un total de 7 amenazas.

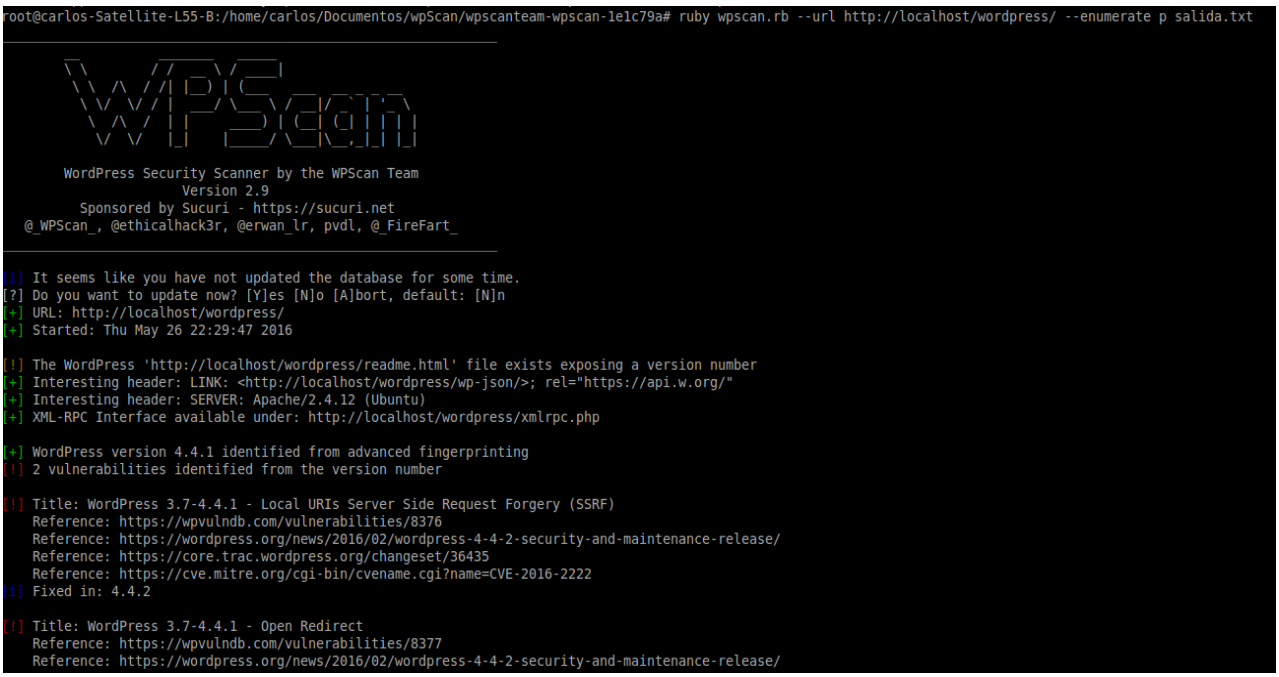


Figura 10: Resultados de la herramienta WPScan.

La herramienta WPScan a pesar de ser de código abierto, licencia libre y que se puede ejecutar por consola no se incluye en la solución porque no exporta los resultados en formato estructurado.

1.2.8. Wapiti

Escáner de vulnerabilidades para aplicaciones web, licenciado bajo GPL v2, busca fallos XSS, inyecciones SQL, inclusiones de archivos (local y remota), ejecución de comandos, inyecciones LDAP, inyecciones CRLF, para poner a prueba la seguridad de las aplicaciones web y poder corregirlas. Esta herramienta está programada en Python y se destaca por sus detallados informes, donde podemos ver la descripción del problema encontrado (18).

Descripción de las pruebas

Para realizar las pruebas con Wapiti, se usó la distribución del sistema Operativo Linux. La versión para realizar la prueba fue la v2.3.0 que se encuentra en el repositorio de Ubuntu. Se utilizó el ambiente de pruebas: Aplicación Web Vulnerable, en inglés Damn Vulnerable Web App (DVWA). A continuación se especifica cada paso, en el que se describe todo el proceso y los resultados que se obtuvieron.

Dirección IP o URL: http://127.0.0.1/dvwa/

Descripción: Antes de realizar el escaneo con Wapiti se introdujo en consola el siguiente comando: *wapiti -h* (Muestra una ayuda de cómo puede ser usada la herramienta para realizar el escaneo). Luego, para realizar el escaneo se introdujo el comando: *wapiti http:///direccion_url/ [options]*. Las opciones restantes se muestran en la ayuda que brinda la herramienta. Utilizando la dirección IP antes mencionada se realizó la prueba, en la que fue necesario esperar unos minutos a que culminara.

Resultados: Se detectaron un total de 17 vulnerabilidades, divididas en: 10 del tipo Inyección SQL a ciegas, 4 Cross Site Scripting y 3 Fichero de backup.

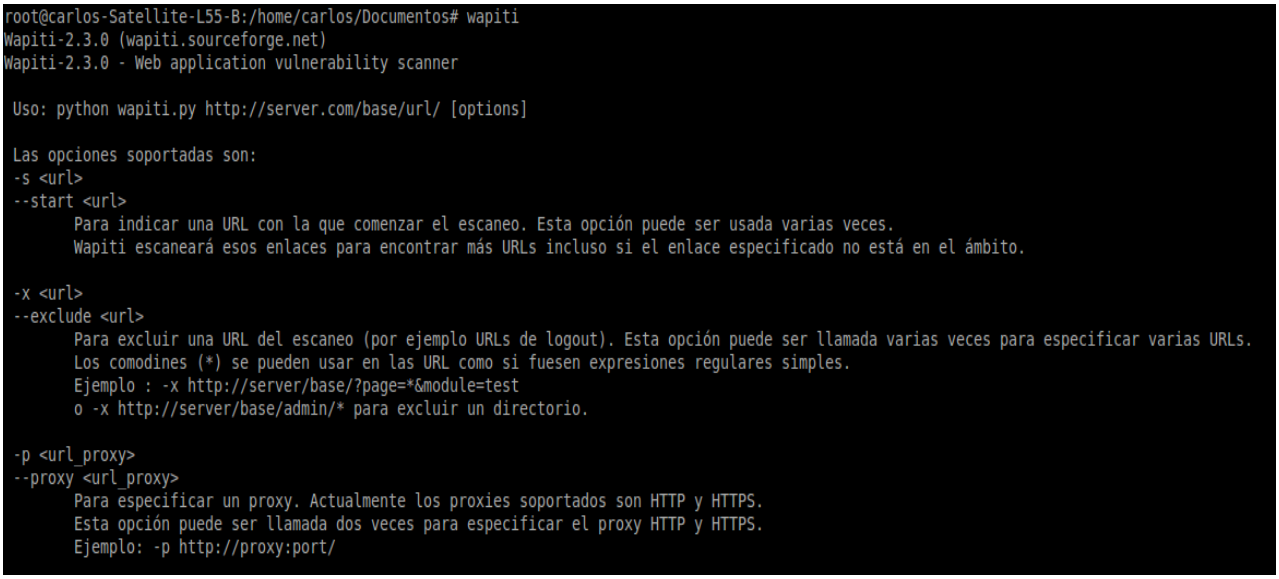


Figura 11: Ejecución de la herramienta Wapiti.

En la figura 11 se muestra la ejecución de la herramienta Wapiti. Se muestra un listado de las opciones que posee la herramienta para su ejecución. Esta herramienta brinda la posibilidad de exportar un informe de resultados en los formatos XML y HTML, en la figura 12 se puede observar un ejemplo de este reporte.

Figura 12: Reporte de la herramienta Wapiti.

Se seleccionó Wapiti para la integración por ser una herramienta de licencia libre. Recibe parámetros por la vía no estándar, por lo que puede realizar la prueba desde otra aplicación a través de la consola de comandos. Aparte de su funcionalidad de detección de vulnerabilidades, ofrece la posibilidad de generar informes en formato estructurado: XML y HTML.

1.2.9. Websecurify Security Testing Framework

Es una solución de pruebas avanzadas construido para identificar rápidamente y con precisión los problemas de seguridad de aplicaciones web. Interfaz de fácil uso para el usuario. La herramienta

está disponible y corre exitosamente en Windows, Mac OS, Linux y otros sistemas operativos (19).

Descripción de las pruebas

Para realizar pruebas con Websecurify se utilizó la versión de la herramienta v2.0 para el Sistema Operativo Windows. La herramienta se opera mediante una interfaz gráfica. Para realizar las pruebas, se utilizó el ambiente de pruebas: Aplicación Web Vulnerable, en inglés Damn Vulnerable Web App (DVWA).

Dirección IP o URL: http://127.0.0.1/dvwa/

Descripción: Una vez abierta la aplicación, se introdujo la dirección IP o URL en el campo de texto que lo indica. Se inició la prueba mediante la opción test. Transcurrido unos segundos, luego de concluida la prueba, la herramienta muestra el reporte de vulnerabilidades detectadas en formato HTML.

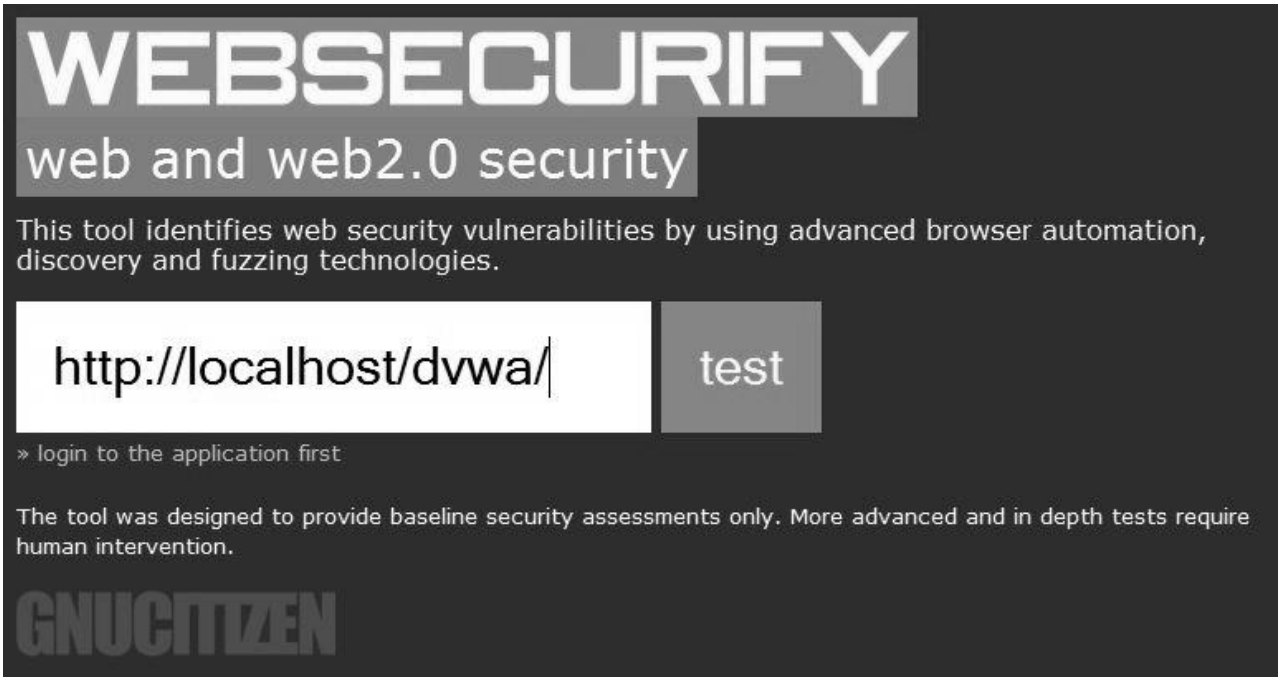


Figura 13: Ejecución de la herramienta Websecurify.

Resultados: Se detectaron un total de 5 vulnerabilidades clasificadas en Low (Baja).

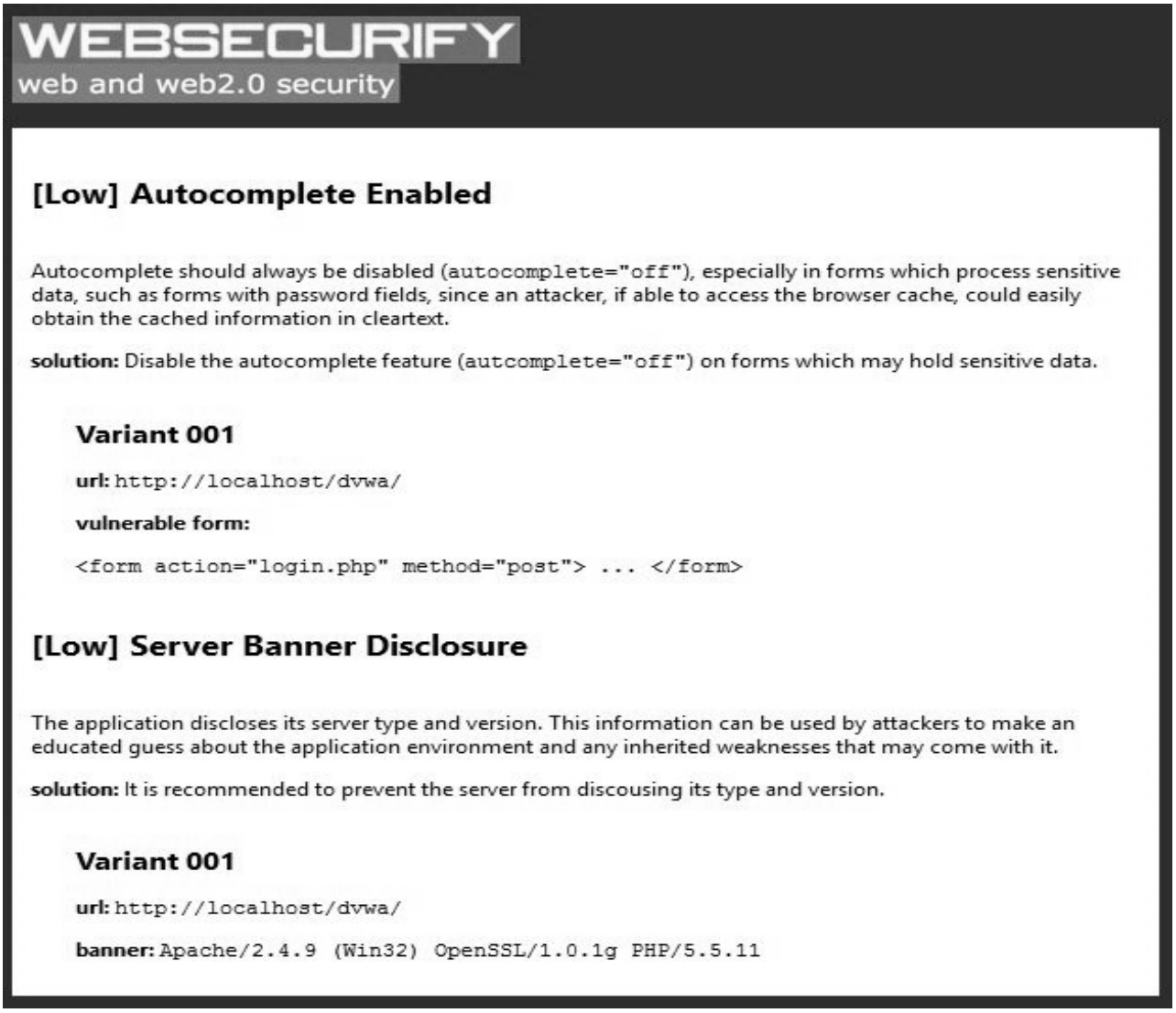


Figura 14: Reporte de la herramienta Websecurify.

La herramienta Websecurify exporta los resultados en formato estructurado (HTML) pero no se incluye en la solución porque no recibe parámetros a través de la vía no estándar, es decir, no se puede realizar la prueba desde otra aplicación a través de la consola de comandos.

1.2.10. Wfuzz

Herramienta diseñada por Edge-Security para la fuerza bruta aplicaciones web, que puede ser utilizado para la búsqueda de recursos no vinculados (directorios, servlets, scripts, etc.), los parámetros para el control de diferentes tipos de inyecciones (SQL, XSS, LDAP, etc.). Tiene como características fundamentales: la recursividad y salida de color (20).

Descripción de las pruebas

Para realizar pruebas con Wfuzz se utilizó la versión v2.1.3 libre para el Sistema Operativo Linux. La herramienta se opera mediante la consola de comandos. A continuación se describe el proceso y los resultados obtenidos luego de realizada la auditoría. Para realizar las pruebas, se utilizó el ambiente de pruebas: Aplicación Web Vulnerable, en inglés Damn Vulnerable Web App (DVWA).

Dirección IP o URL: http://127.0.0.1/dvwa/

Descripción: Para realizar el escaneo se introdujo en la consola el comando: *Python Wfuzz.py –z file,wordlist/general/test.txt http://localhost/prueba/FUZZ* donde se especifica la dirección url a la que se le va a realizar la prueba y los directorios a aplicar fuerza bruta.

Resultados: De un total de 11 peticiones procesadas no se logró filtrar ninguna.

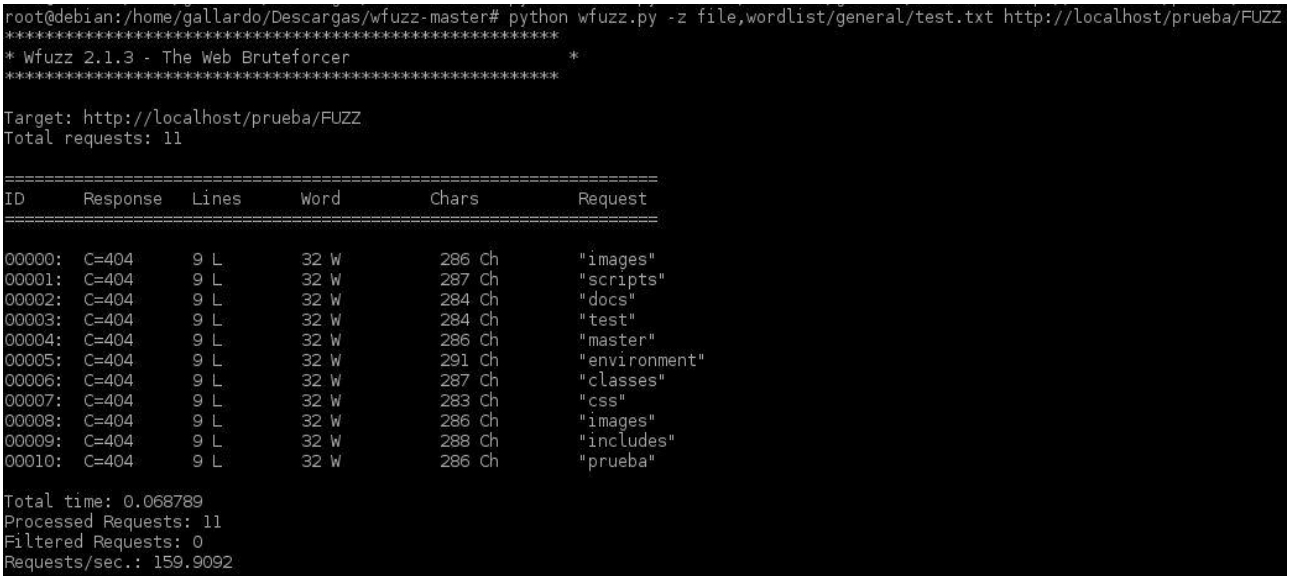


Figura 15: Ejecución de la herramienta Wfuzz.

En la figura 15 se muestra el contenido de los ataques realizados por la aplicación de fuerza bruta, de la cual se obtuvo un reporte que se muestra en la figura 16.

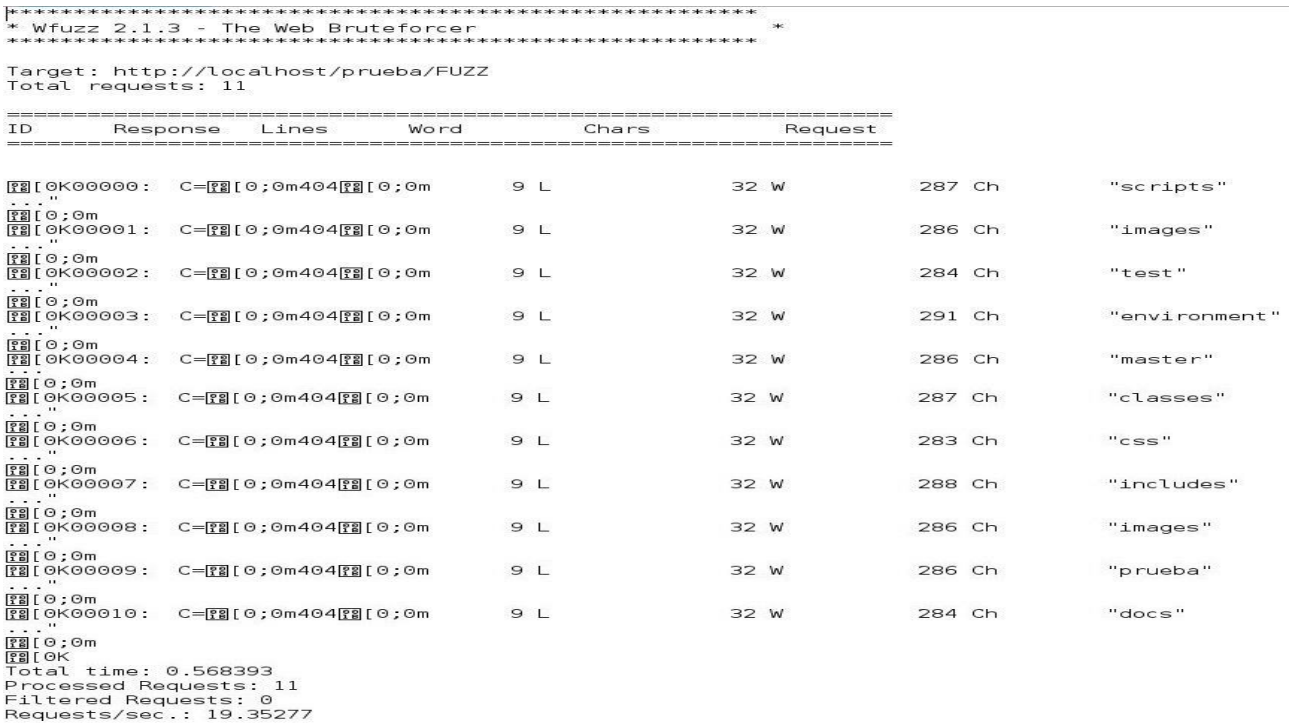


Figura 16: Reporte de la herramienta Wfuzz.

Aunque la herramienta Wfuzz cumple con algunas de las características que se plantean en la investigación como son: código abierto y recibe parámetros por consola, no forma parte de la solución porque los reportes no tienen un formato estructurado.

1.2.11. Conclusiones parciales

Como resultado de las pruebas realizadas y de acuerdo con las características necesarias que debe poseer cada herramienta se seleccionó Wapiti y Joomscan para integrar en XILEMA-PlatSI. Se tuvo en cuenta además las siguientes consideraciones:
Wapiti detectó un grupo de vulnerabilidades que no fueron detectadas por las herramientas que tiene integrada la Plataforma, además sus reportes apoyan el proceso de correlación. Por otra parte, Joomscan es una aplicación creada y mantenida por OWASP, su integración apoya la detección de vulnerabilidades ya que PlatSI no contaba con una herramienta especializada en análisis de vulnerabilidades a aplicaciones desarrolladas en el CMS Joomla.

1.3. Marco de trabajo

El Módulo de Auditorías a Aplicaciones Web basa su funcionamiento en la utilización de herramientas de pruebas de seguridad a aplicaciones web con el objetivo de generar un informe con las vulnerabilidades detectadas en una auditoría. El especialista de seguridad informática autenticado en la Plataforma selecciona la(s) herramienta(s) que considere necesaria(s) para realizar la auditoría, donde introduce los datos que requiera(n). La Plataforma hace uso de Bambú, el cual es un distribuidor de tareas que permite la ejecución de herramientas de forma distribuida, utilizando Internet Comunication Engine (ICE[10]) para la comunicación entre el servidor y los agentes[11]. PlatSI se realizó mediante el marco de trabajo para aplicaciones web Symfony2.
Con el objetivo de mantener uniformidad, se utilizaron en la solución de la problemática las herramientas definidas por el equipo de desarrollo de PlatSI, las cuales se describen a continuación:

1.3.1. Bambú

Bambú fue desarrollado en el centro TLM con el objetivo de ser usado en sistemas que trabajen de forma distribuida. Se implementó en Python, este lenguaje de programación facilita la asimilación y entendimiento de su código fuente.

Por tanto, Bambú es un sistema genérico que permite la ejecución de tareas de forma distribuida, así como la obtención y almacenamiento de sus resultados. Está compuesto por dos subsistemas, el servidor y conectados a este uno o varios agentes. El servidor es el encargado de recibir cada

[10] Estándar desarrollado por ZeroC para el desarrollo de aplicaciones basadas en objetos distribuidos, es utilizado para establecer comunicación entre cliente y servidor.
[11] Son las computadoras donde estarán instaladas las herramientas de pruebas de seguridad.

tarea y asignársela a un agente que pueda realizarla. Por su parte los agentes son las PC que ejecutan finalmente las tareas y envían hacia el servidor el resultado obtenido (21).

1.3.2. Symfony2

Framework basado en la arquitectura Modelo-Vista-Controlador el cual se ha convertido en unos de los más populares de PHP en el desarrollo de aplicaciones web. Proporciona estructura al código fuente, forzando al desarrollador a crear código más legible y fácil de mantener (22). Se incluye en la solución puesto que es la base de la implementación de PlatSI y es usado para implementar las clases de los plugins en el servidor web. Se utilizará en su versión 2.3.

1.4. Tecnologías y herramientas utilizadas

Para dar cumplimiento a los objetivos trazados en la investigación se realizó un estudio de las posibles herramientas a utilizar, se analizaron los lenguajes de programación definidos por los especialistas de PlatSI en la creación de los plugins y se definió la metodología que más se adapta a utilizar en la presente investigación. A continuación se realiza una detallada descripción de estas:

1.4.1. Metodología de desarrollo

Entre las metodologías de desarrollo ágiles más conocidas y usadas se encuentra la Programación Extrema (XP), centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en la realimentación continua entre el cliente y el equipo de desarrollo, la comunicación fluida entre todos los participantes, la simplicidad en las soluciones implementadas y el coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico (23).

Por las razones descritas anteriormente se decide utilizar la metodología de desarrollo XP en el presente trabajo, ya que se adapta adecuadamente a las condiciones en que se realiza el mismo, como se muestra a continuación: es un proyecto orientado directamente a la investigación de nuevas herramientas de pruebas de seguridad a aplicaciones web y la implementación de los plugins que integren dichas herramientas. El proyecto es de corta duración, desarrollado por una pareja de programadores. Esto implica que no sea necesaria la generación de numerosos artefactos, aportando agilidad al proceso de desarrollo del software.

1.4.2. Lenguajes de programación

Python: Es un lenguaje de programación multiparadigma, esto facilita a los desarrolladores adoptar un estilo particular de programación, beneficiando así que el código sea legible y que los programas de diferentes programadores tengan un aspecto similar (24). Se utilizó en la implementación de los plugins necesarios para integrar cada una de las herramientas seleccionadas a la Plataforma. Otra

característica por la que fue utilizado es porque es el lenguaje en que se encuentra desarrollado Bambú. La versión utilizada será la 2.7.

PHP[12]: Es un lenguaje interpretado de alto nivel en páginas HTML y ejecutado en el servidor, permita escribir código ordenado, estructurado y manejable. Es uno de los lenguajes más flexibles, potentes y de alto rendimiento conocido, por lo que cuenta con abundante documentación y una amplia comunidad en el mundo (25). Por sus características este lenguaje forma parte de la solución ya que es utilizado en la Plataforma como lenguaje del lado del servidor para Symfony2. La versión utilizada será la 5.3.8.

1.4.3. Lenguaje de modelado

Notación para el Modelado de Procesos de Negocio, por sus siglas en inglés Business Process Modeling Notation (BPMN), está basada en diagramas de flujo para definir procesos de negocio, desde los más simples hasta los más complejos para dar soporte a la ejecución de procesos. Esta notación ha sido especialmente diseñada para coordinar la secuencia de los procesos y los mensajes que fluyen entre los participantes de las diferentes actividades. Además proporciona un lenguaje común para que las partes involucradas puedan comunicar los procesos de forma clara, completa y eficiente (26). Se emplea en la investigación para apoyar la propuesta de solución y brindar una mayor comprensión del proceso de detección de vulnerabilidades a aplicaciones web.

1.4.4. Herramienta de modelado

Visual Paradigm: Es una herramienta para el desarrollo de software de aplicación, diseñada para la ayuda del desarrollo del software. Es utilizada por distintos usuarios entre los que se incluyen ingenieros de software, analistas de sistemas, analistas de negocios, arquitectos y desarrolladores. Soporta el ciclo de vida completo del desarrollo de software (27). Se utiliza para modelar el diagrama de negocio del proceso de detección de vulnerabilidades, y aunque XP no precisa el diseño de clases, en la presente investigación se diseñaron las clases asociadas a cada plugin para apoyar y permitir un mejor entendimiento de la propuesta de solución. Se utilizó la herramienta en su versión 5.0.

1.4.5. Herramientas de desarrollo

PyCharm: Entorno de Desarrollo Integrado (IDE) multiplataforma utilizado para desarrollar en el lenguaje de programación Python. Proporciona análisis de código, depuración gráfica (28). Para la implementación de la propuesta de solución se utilizó la versión 4.5.4, con el objetivo de escribir un código limpio y fácil de mantener. Además para mantener una uniformidad con respecto al lenguaje de programación que están integradas las anteriores herramientas.

[12]Por sus siglas en inglés Hypertext Pre-Processor.

PHPStorm: Es uno de los Entorno de Desarrollo Integrado (IDE) más completos de la actualidad, editor para el lenguaje PHP, que proporciona terminación inteligente de código, resaltado de sintaxis y la configuración de formato de código extendido (29). Se utilizó como herramienta de desarrollo para el presente trabajo en su versión 9.0.2, ya que apoya la creación de una parte de los plugins de integración en la Plataforma.

1.5. Conclusiones

En este capítulo se definieron los principales conceptos y tecnologías que fueron utilizados en la investigación, logrando un mayor entendimiento de su utilización y el impacto que tienen en el presente trabajo. Se realizaron pruebas con las herramientas descargas y después del análisis de sus características, se llegó a la conclusión de que las herramientas Wapiti y Joomscan forman parte de la solución.

2. Introducción

En el presente capítulo se describe la propuesta de solución. Se definen las funcionalidades del sistema y la lista de reserva del producto. Se muestran las fases de Planificación y Exploración de la metodología utilizada, donde se definen las HU asociadas a cada iteración, se especifica su plan de entrega y estimación de esfuerzo.

2.1. Propuesta de solución

En el proyecto Plataforma de Seguridad en las Tecnologías de la Información (XILEMA-PlatSI), se desarrolla un producto de igual nombre, que está concebido para los tres módulos mencionados anteriormente. Específicamente en el MAAWeb, todo el flujo de trabajo de las herramientas que tiene integradas transita a través de Bambú, servidor que distribuye las diferentes tareas desde que se ejecuta la herramienta hasta que se muestran los reportes de vulnerabilidades. En la figura que se muestra a continuación se define la estructura actual de la Plataforma y se destaca en diferente color donde se encuentra enmarcada la solución:

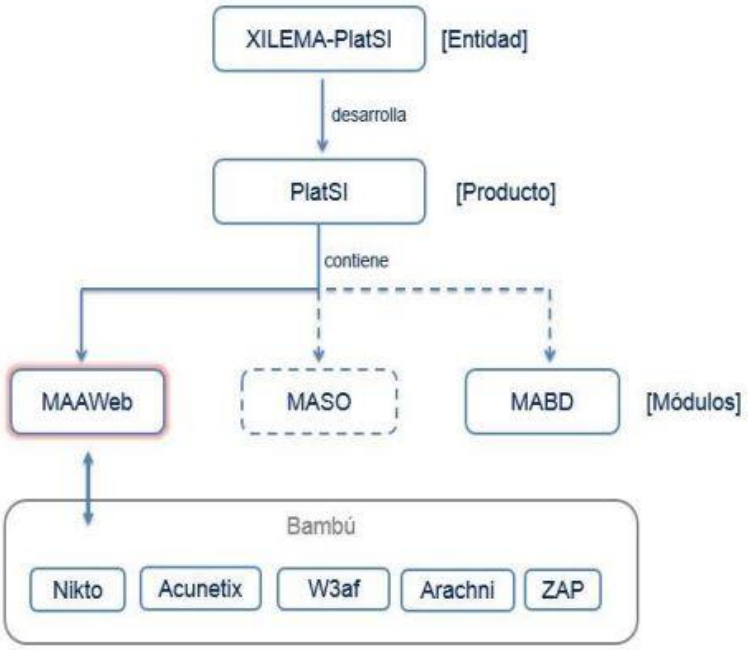


Figura 17: Estructura de PlatSI *(30)*.

Con el objetivo de integrar nuevas herramientas y que no afecten el funcionamiento de los demás procesos que realiza la Plataforma, fue necesario la realización de un mecanismo de plugins. Entonces, por cada herramienta seleccionada a formar parte de la solución es necesario implementar tres plugins, donde en el distribuidor de tareas Bambú se ubican los dos primeros, uno para el subsistema Cliente y otro para el subsistema Servidor, los que se encargan de enviar tareas

a un agente para la ejecución de una herramienta y guardar el resultado en una base de datos para procesarlo y mostrar la información relevante a un especialista. El plugin restante se ubica en el servidor web de Symfony, que permite al especialista realizar el proceso de auditorías mediante la aplicación web de la Plataforma. Para un mayor entendimiento de la propuesta de solución se muestra la siguiente figura:

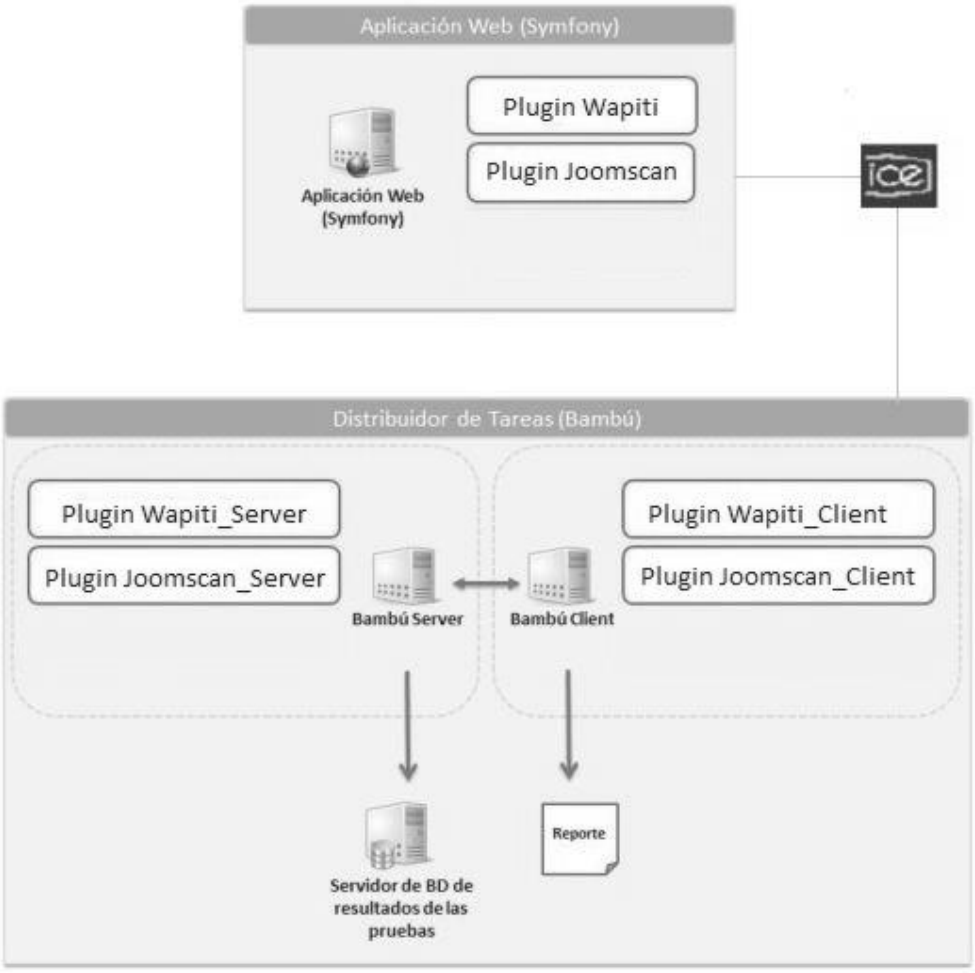


Figura 18: Propuesta de Solución en el MAAWeb.

2.1.1. Detección de vulnerabilidades

El especialista de seguridad informática autenticado en la Plataforma selecciona la(s) herramienta(s) que considere necesaria(s) para realizar la auditoría. Para ello introduce el(los) dato(s) necesarios, los cuales varían en dependencia de la(s) herramienta(s) que se seleccione(n). El servidor web valida los parámetros que introdujo el usuario, si los mismos están correctos el distribuidor de tareas se encarga de enviar los datos para que se ejecute la herramienta correspondiente y en caso de detectar vulnerabilidades, enviar un reporte al usuario. Si los datos iniciales no eran correctos, se informa al usuario. En caso de que la herramienta no se ejecute por algún error, se envía una notificación al especialista. Para un mayor entendimiento, este proceso se muestra en la siguiente ilustración:

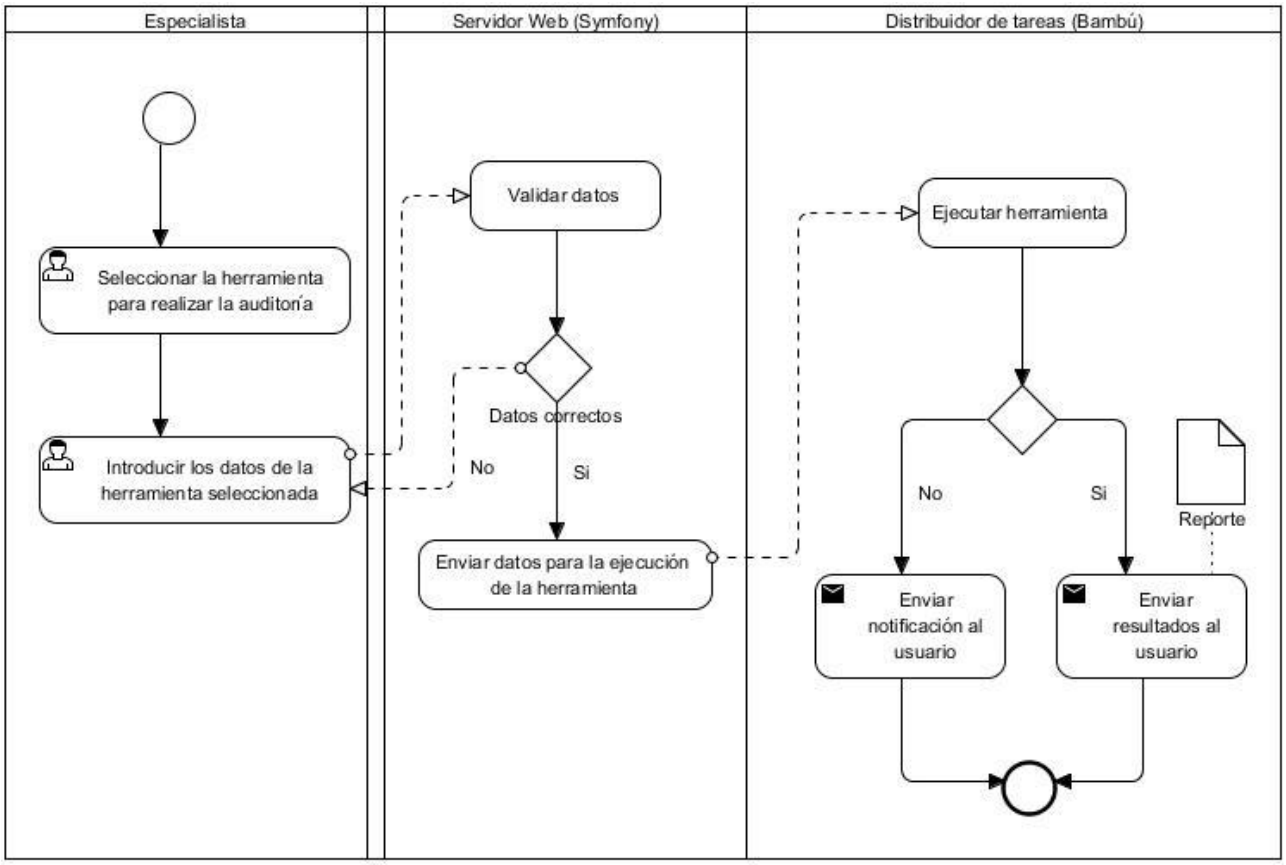


Figura 19: Proceso de negocio de detección de vulnerabilidades.

2.2. Características funcionales del sistema y lista de reserva de producto

A continuación, se muestran los requisitos funcionales del sistema y la lista de reserva del producto donde, estos últimos no apuntan directamente a las funcionalidades del sistema sino a rasgos como software, hardware y usabilidad.

2.2.1. Características funcionales del sistema

1. Realizar pruebas de seguridad a aplicaciones web con la herramienta Wapiti.
2. Realizar pruebas de seguridad a aplicaciones web con la herramienta Joomscan.

2.2.2. Lista de reserva de producto

Hardware:

- Para la instalación de la aplicación es necesario disponer de una computadora con mínimo de 512 MB de RAM, y de espacio en disco duro de al menos de 1 GB.

Software:

- Producto PlatSI.
- Navegador web.

Usabilidad:

- Es necesaria una preparación previa para operar con PlatSI y con herramientas para detectar vulnerabilidades a aplicaciones web.

2.3. Personas relacionadas con el sistema

Las personas relacionadas con el sistema se dividen en dos grupos:

- Administrador: Encargado del mantenimiento, implementar sus funcionalidades y realizar el proceso de integración de las herramientas.
- Especialistas: Encargado de interactuar con las funcionalidades que brinda la Plataforma.

2.4. Fase Exploración

Fase inicial definida por la metodología XP, basada en la creación de Historias de Usuario (HU). Estas historias se encargan de describir las características y funcionalidades del sistema.

2.4.1. Historias de Usuario

Permiten administrar los requisitos sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. Además responden rápidamente a los requisitos cambiantes. Se representan en ellas las funcionalidades que se van a implementar en el sistema (31).

Cada HU está compuesta al menos por los siguientes aspectos (32):
Número: Posee el número asignado a la HU.
Nombre de HU: Atributo que contiene el nombre de la HU.
Usuario: El usuario del sistema que utiliza o protagoniza la HU.
Prioridad en el negocio: Evidencia el nivel de prioridad de la HU en el negocio. Se considera Alta en caso de que la HU sea imprescindible en el negocio, Media en caso de que su realización o no lo afecte considerablemente y Baja cuando no se considera una prioridad para el negocio.
Riesgo de desarrollo: Evidencia el nivel de riesgo en caso de no realizarse la HU. Se considera Alta, cuando el riesgo de no realizar la HU implica en el funcionamiento de la Plataforma. Media cuando el riesgo de no realizarla es medianamente importante y Baja en caso de que no se considere un riesgo el hecho de tardar en la realización de la HU y no implique en el funcionamiento de la Plataforma.
Puntos estimados: Este atributo no es más que una estimación hecha por el equipo de desarrollo del tiempo de duración de la HU. Cuando el valor es 1 equivale a una semana ideal de trabajo. En la metodología XP está definida una semana ideal en 5 días hábiles trabajando 40 horas, es decir, 8 horas diarias. Por lo que cuando el valor de dicho atributo es 0.5 equivale a 2 días y medio de trabajo, lo que se traduce en 20 horas.
Iteración asignada: Se especifica la iteración a la que pertenece la HU correspondiente.
Descripción: Posee una descripción de lo que realizará la HU.

A continuación se muestran las historias de usuarios definidas:

Tabla 1: HU #1: Realizar pruebas de seguridad a aplicaciones web con la herramienta Wapiti.

Historia de Usuario	
Número:1	**Usuario:** Especialista
Nombre: Realizar pruebas de seguridad a aplicaciones web con la herramienta Wapiti.	
Prioridad en negocio: Alta	**Riesgo en Desarrollo:** Media
Puntos estimados: 2.4	**Iteración asignada:** 1
Programadores responsables: Carlos A. Méndez Denis y Carlos M. Medina Aguilera	
Descripción: El especialista selecciona la herramienta con la que va a realizar la prueba, una vez seleccionada la herramienta, introduce los parámetros (url) correspondientes a la misma, envía la petición con los datos introducidos para realizar la prueba. Estos parámetros conforman un comando que es enviado a Bambú mediante el plugin de Wapiti ubicado en Symfony, donde los plugins de Bambú_Client y Bambú_Server correspondientes a Wapiti se encargan de modificar el comando para ejecutar la herramienta ubicada en el agente seleccionado para realizar dicha prueba. Una vez ejecutada la herramienta devuelve un resultado, el cual no es mostrado al usuario. Este resultado se modifica para eliminar de él la información innecesaria detectada por la herramienta, se conforma un nuevo resultado y se envía al experto como el Reporte de vulnerabilidades detectadas.	

Tabla 2: HU #2: Realizar pruebas de seguridad a aplicaciones web con la herramienta Joomscan.

Historia de Usuario	
Número:2	**Usuario:** Especialista
Nombre: Realizar pruebas de seguridad a aplicaciones web con la herramienta Joomscan.	
Prioridad en negocio: Alta	**Riesgo en Desarrollo:** Media
Puntos estimados: 2	**Iteración asignada:** 2
Programadores responsables: Carlos A. Méndez Denis y Carlos M. Medina Aguilera	
Descripción: El especialista selecciona la herramienta con la que va a realizar la prueba, una vez seleccionada la herramienta, introduce los parámetros (url) correspondientes a la misma, envía la petición con los datos introducidos para realizar la prueba. Estos parámetros conforman un comando que es enviado a Bambú mediante el plugin de Joomscan ubicado en Symfony,	

donde los plugins de Bambú_Client y Bambú_Server correspondientes a Joomscan se encargan de modificar el comando para ejecutar la herramienta ubicada en el agente seleccionado para realizar dicha prueba. Una vez ejecutada la herramienta devuelve un resultado, el cual no es mostrado al usuario. Este resultado se modifica para eliminar de él la información innecesaria detectada por la herramienta, se conforma un nuevo resultado y se envía al experto como el Reporte de vulnerabilidades detectadas.

2.5. Fase Planificación

Es la segunda fase definida por la metodología XP, donde se realiza una estimación de esfuerzo para la realización de cada HU especificada, se define el plan de iteraciones y el plan de entregas.

2.5.1. Estimación de esfuerzo por Historias de Usuario

Para la estimación de esfuerzo por HU se utilizó como medida el punto, donde se realiza una escala de uno a tres puntos, siendo un punto equivalente a una semana ideal de programación. Se utilizó el método de analogía, ya que se tuvo en cuenta el tiempo y el esfuerzo que se utilizó para integrar las herramientas con que cuenta actualmente la Plataforma. Esta información se encuentra reflejada en la tabla que se muestra a continuación:

Tabla 3: Estimación de esfuerzo por Historias de Usuario.

No.	Historias de Usuarios	Puntos de Estimación
1	Realizar pruebas de seguridad a aplicaciones web con la herramienta Wapiti.	2.4
2	Realizar pruebas de seguridad a aplicaciones web con la herramienta Joomscan.	2

2.5.2. Plan de iteraciones y entregas

Definidas las HU y estimado el esfuerzo necesario para la realización de cada una de ellas, se dio paso a la planificación de la fase de implementación. En esta fase se establecieron 2 iteraciones.

1. En la iteración #1 se desarrolló la primera HU, que se corresponde con la integración de la herramienta Wapiti.
2. En la iteración #2 se desarrolló la segunda HU, que se corresponde con la integración de la herramienta Joomscan.

Para la realización del plan de entregas se tuvo en cuenta la duración estimada en semanas y días. Se especificó la iteración con su HU correspondiente, así como la fecha de inicio y fin de cada una. Para un mayor entendimiento se muestra la siguiente tabla:

Tabla 4: Plan de duración de las iteraciones.

No. Iteración	Historias de Usuarios	Fecha Inicio	Fecha Fin	Duración (semanas-días)
1	Realizar pruebas de seguridad a aplicaciones web con la herramienta Wapiti.	18 de abril del 2016	3 de mayo del 2016	2 semanas y 2 días
2	Realizar pruebas de seguridad a aplicaciones web con la herramienta Joomscan.	4 de mayo del 2016	17 de mayo del 2016	2 semanas

2.6. Conclusiones

Se llevaron a cabo las fases de Exploración y Planificación definidas por XP, que permitieron describir la propuesta de solución y el proceso de negocio de detección de vulnerabilidades. Además, con la realización de la estimación de esfuerzo y la definición del plan de entregas, se garantizó que el proyecto finalizará en tiempo.

3. Introducción

En el presente capítulo se describe la arquitectura de PlatSI y las fases de diseño e implementación definidas en la metodología de desarrollo utilizada en el presente trabajo. También se define el patrón arquitectónico en el que se basa el desarrollo de los plugins y los patrones de diseño utilizados. Además, se representan las tarjetas Clase – Responsabilidad – Colaborador (CRC), las Tareas de Ingenierías y el estándar de codificación para Python y PHP utilizado en la implementación.

3.1. Arquitectura del Sistema

Una arquitectura de software describe los componentes básicos de un sistema de software y su combinación interna. Para poder funcionar con éxito, debe ser adaptada a los restantes factores del proyecto de software. Una arquitectura de software bien configurada facilita a los usuarios y desarrolladores la comprensión del sistema (33). Para una mayor comprensión del proceso de detección de vulnerabilidades se muestra a continuación una figura donde se define la arquitectura general de la Plataforma:

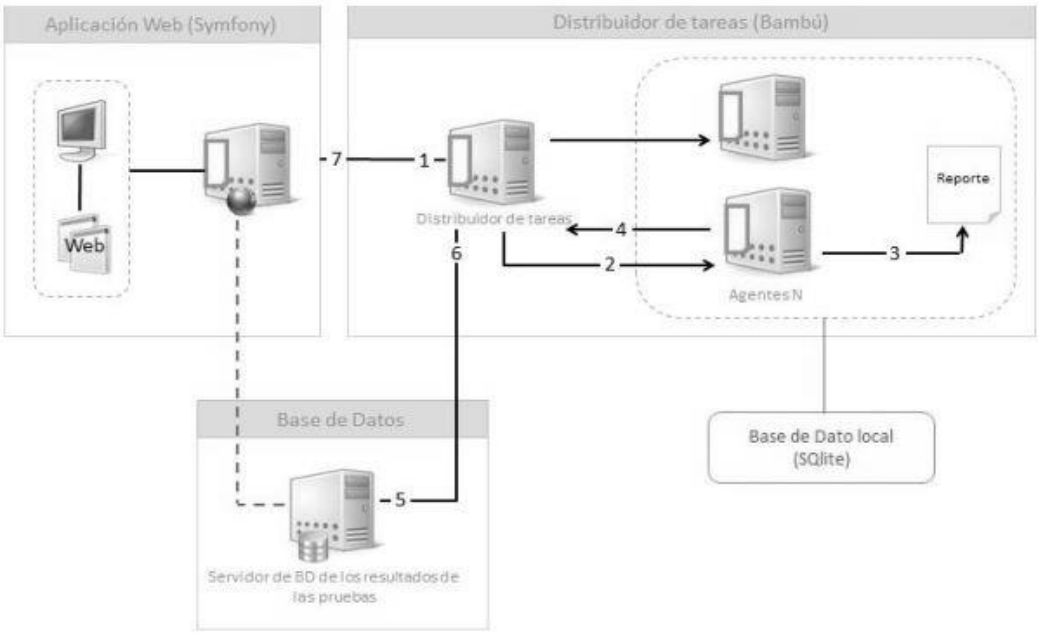


Figura 20: Arquitectura de PlatSI.

Descripción de los componentes utilizados en la representación de la arquitectura:

—#→ Dirección del flujo de procesos que se realizan y el orden en que se realizan (ambas direcciones en caso de no especificar la flecha).

Subdivisión lógica de procesos internos.

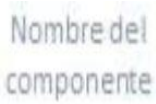
División lógica de los procesos en dependencia de sus características y funciones.

Proceso interno que guarda relación con un componente.

Reportes generados por los agentes.

Plugins a desarrollar ubicadas en los componentes correspondientes.

Relación de un componente con otro.



Nombre del componente.

3.1.1. Flujo de procesos del diagrama

A continuación se describen el flujo de actividades que se realizan:

1: Aplicación web (Symfony) – Distribuidor de tareas (Bambú):

El proceso comienza desde la aplicación web, donde un especialista realiza las configuraciones y especificaciones necesarias para realizar la prueba. De forma estándar los parámetros e informaciones se envían al (los) servidor (es).

2: Distribuidor de tareas (Bambú) (Distribuidor de tareas – Agente N):

El proceso está asociado directamente al componente Bambú-Server, en el cual se encuentran comprendidos los servidores distribuidores que, al recibir la configuración de las pruebas a realizar, el componente Bambú-Server comienza a distribuir las tareas a cada uno de los agentes de pruebas. Cada agente cuenta con el componente Bambú-Client y de esta forma se establece la comunicación. En este proceso se tiene en cuenta el rendimiento de los agentes. El componente Bambú-Server accede a una Base de Datos, donde va a estar registrado el (los) agente (es) que puede utilizar para las tareas a realizar según especificaciones del flujo anterior. Cada agente de prueba, al recibir una tarea, asigna la ejecución de esta a uno de sus hilos de proceso, lo cual debe realizarse de la forma más conveniente posible. Una vez que Bambú-Client recibe una tarea, almacena en una base de datos local (SQlite) información temporal referente a la tarea y a su estado de ejecución. Esta información se actualiza durante la ejecución y es eliminada una vez que Bambú-Client crea el resultado de la ejecución de la tarea.

3: Distribuidor de tareas (Bambú) (Agente N - Reporte):

Una vez concluida la tarea y en dependencia de las pruebas, se genera un fichero en diferentes formatos. El servidor de Bambú-Client es el encargado de normalizar dichas pruebas para obtener una estructura estándar que luego se va a insertar en una base de datos para asegurar la permanencia de los resultados de las pruebas.

4: Distribuidor de tareas (Bambú) (Agente N - Distribuidor de tareas):

Una vez finalizado el proceso de normalización y procesamiento de la información, a través de la comunicación Bambú-Client - Bambú-Server, el fichero obtenido de la tarea ejecutada en el agente, es enviado al (los) servidor (es) distribuidor (es). Luego los datos normalizados son enviados igualmente al servidor distribuidor, donde el conjunto de clases críticas adicionadas al Bambú-Server se encargará de enviarlas a la base de datos correspondiente.

5: Distribuidor de tareas (Bambú) – Base de Datos:

Los datos estandarizados y recibidos de los agentes, son enviados por las clases críticas a la base de datos para ser almacenados.

6: Base de Datos - Distribuidor de tareas (Bambú):

El servidor de base de datos informa sobre la acción de guardar datos, en correspondencia con lo sucedido, informa si hubo un error u ocurrió el proceso eficientemente.

7: Distribuidor de tareas (Bambú) - Aplicación web (Symfony):

El reporte de las vulnerabilidades detectadas es mostrado al experto a través de la aplicación web.

3.2. Patrón Arquitectónico

Los patrones arquitectónicos describen un problema particular y recurrente de diseño, que aparece en contextos de diseño específico, y presenta un esquema genérico demostrado con éxito para su solución. El esquema de solución se especifica mediante la descripción de los componentes que la constituyen, sus responsabilidades y desarrollos, así como también la forma como estos colaboran entre sí (34). El patrón arquitectónico utilizado en el presente trabajo es el Modelo-Vista-Controlador.

3.2.1. Arquitectura Modelo-Vista- Controlador

La arquitectura Modelo-Vista-Controlador (MVC) separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones. El controlador se encarga de aislar al modelo y a la vista de los detalles del protocolo utilizado para las peticiones (HTTP, consola de comandos, email, etc.). El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes. Por último, la vista transforma el modelo en una página web que permite al usuario interactuar con

ella (35). Para un mayor entendimiento de la arquitectura a continuación se muestra la siguiente figura:

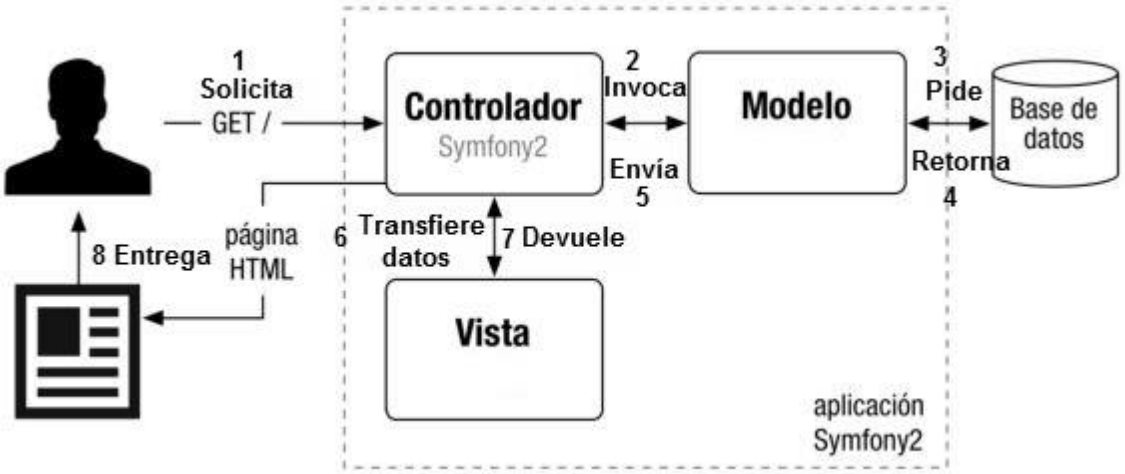


Figura 21: Arquitectura Modelo Vista Controlador *(35)***.**

Esta arquitectura fue utilizada en la presente investigación ya que la Plataforma está desarrollada en Symfony2, framework que está basado en este patrón. Donde solo fue necesario hacer uso de la Vista y del Modelo, ya que el Controlador es genérico para todos los plugins de la Plataforma. En la Vista se generaron los formularios correspondientes a las características de cada herramienta integrada, y el Modelo contiene la lógica de negocio de los plugins.

3.3. Patrones de diseño

Los patrones de diseño comunican los estilos y soluciones consideradas como buenas prácticas, que los expertos en el diseño orientado a objetos utilizan para la creación de sistemas. En el presente trabajo se utilizaron los patrones General Responsibility Assignment Software Pattern (GRASP) y Método Plantilla.

3.3.1. Patrones GRASP

Los patrones GRASP por sus siglas en inglés General Responsibility Assignment Software Patterns describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en formas de patrones (36). A continuación se describe su utilización:

- **Experto en información**

Asigna una responsabilidad al experto en información; la clase que tiene la información necesaria para llevar a cabo la responsabilidad. Se utilizó en la creación de los plugins de Symfony donde en el bundle BamplusPlugins se crearon carpetas con el nombre de las herramientas a integrar y dentro la carpeta correspondiente se creó la clase experta WapitiPlugin y JoomscanPlugin, permiten al especialista realizar el proceso de auditoría mediante la aplicación web. También se empleó en la creación de los plugins de Bambú en el paquete ImplementClass, donde se crearon las clases

expertas Wapiti y Joomscan, encargadas de la ejecución de las herramientas y extraer información relevante de los reportes.

- **Creador**

Crea una nueva instancia por la clase que tiene la información necesaria para realizar la creación del objeto y usa directamente las instancias creadas del objeto. Se evidencia su utilización en el plugin correspondiente para Symfony donde se creó la clase experta WapitiPlugin para la herramienta Wapiti, y a partir de la cual se creó un paquete Form que contienen la clase WapitiFormType y otro Model, que tiene las clases WapitiTool y WapitiParameter. La figura que se muestra a continuación evidencia el uso de este patrón:

```
public function __construct()
{
    $this->name = 'Wapiti';
    $this->tool = new WapitiTool($this->name);
    $this->parameter = new WapitiParameter();
}
```

Figura 22: Ejemplo del patrón creador.

- **Alta cohesión**

Asigna responsabilidades de manera que la información que almacena una clase sea coherente y esté relacionada con la clase. Se empleó en la implementación de las funcionalidades necesarias en cada clase, dicha información se puede observar en la figura 23.

- **Bajo acoplamiento**

Diseñar las clases con el objetivo de que estén lo menos ligadas entre sí. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases. Está reflejado en la estructuración de las clases correspondientes a los plugins para Bambú y Symfony, para un mayor entendimiento observar la figura 23, 24 y 25.

3.3.2. Patrón Método Plantilla

Propone abstraer todo el comportamiento que comparten las entidades en una clase (abstracta) de la que, posteriormente, extenderán dichas entidades. Esta superclase definirá un método que contendrá el esqueleto de ese algoritmo común (método plantilla o Template Method) y delegará determinada responsabilidad en las clases hijas, mediante uno o varios métodos abstractos que deberán implementar (37).

Este patrón fue utilizado en el desarrollo de los plugins de Bambú donde se implementaron métodos declarados en clases abstractas como ITaskClient e ITaskServer, para las clases Bambú- Client y Bambú- Server respectivamente. También se ve reflejado el uso de este patrón de diseño en la

implementación de los plugins en Symfony, donde la clase experta (WapitiPlugin, JoomscanPlugin) implementó la interfaz abstracta PluginInterface, que tiene definidos métodos como getPluginName(), getModelName(), getModelImage(), getModelDescription(), getParameters(), getApplicationName(), getFormType() y runConfig(), donde se utilizaron los mismos de acuerdo a las funcionalidades y parámetros específicos de cada herramienta integrada.

3.4. Diseño de Clases

Aunque la metodología XP no precisa el diseño de clases, en la presente investigación se diseñaron las clases asociadas a cada plugin para apoyar la propuesta de solución y permitir un mejor entendimiento del uso de los patrones de diseño. A continuación se muestran los diagramas de clases correspondientes a cada plugin:

A continuación en la figura 23 se muestra el diagrama de clases correspondiente a los plugins de Symfony, donde quedaron definidas las clases expertas, las clases HerramientaFormType, HerramientaTool, HerramientaParameter, la interfaz PluginInterface y la relación que existen entre ellas.

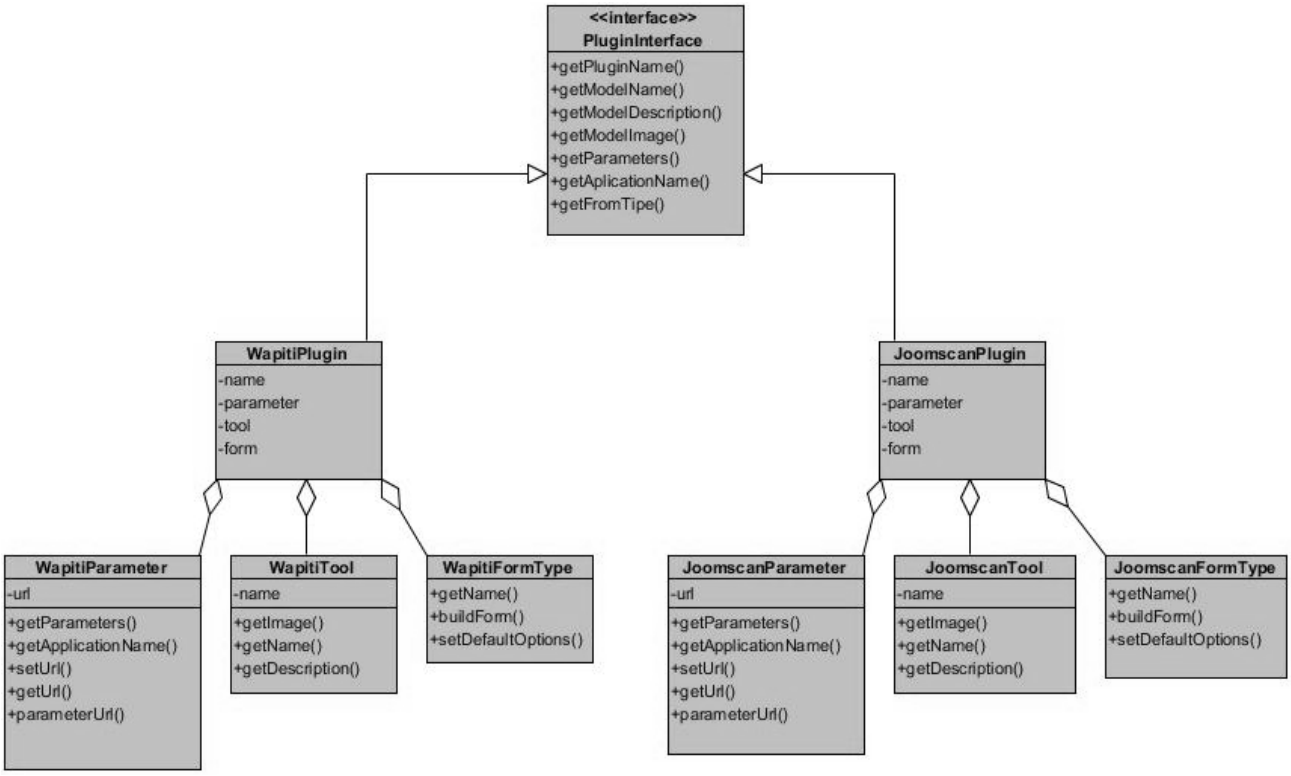


Figura 23: Diseño de clases (Plugins Symfony).

Las figuras siguientes representan las clases correspondiente al plugin de Bambú. La figura 24 muestra las clases diseñadas en Bambú-Client donde las clases de las herramientas implementan los métodos que brinda la clase abstracta ITaskClient y el formato de los reportes generados por cada herramienta.

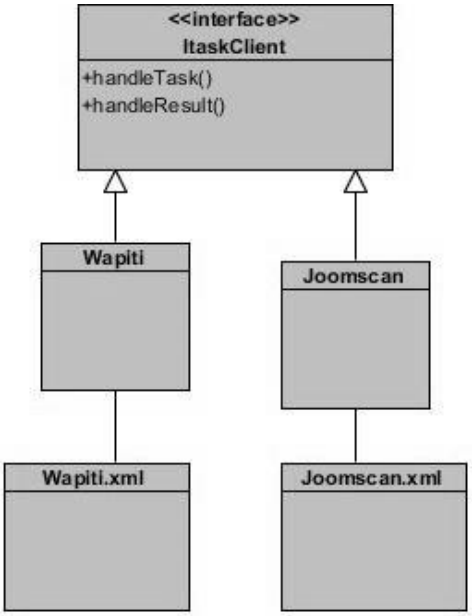


Figura 24: Diseño de clases (Plugins Cliente de Bambú).

Por otra parte, la figura 25 muestra las clases diseñadas en Bambú-Server que implementan los métodos que brinda la clase abstracta ITaskServer.

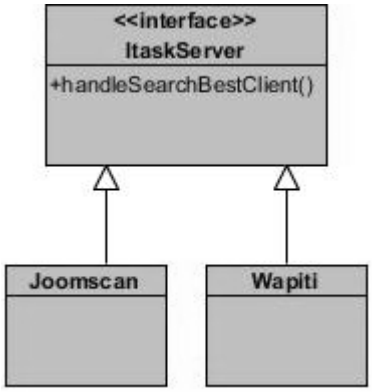


Figura 25: Diseño de clases (Plugins Servidor de Bambú).

3.5. Tarjetas Clase – Responsabilidad – Colaborador

Las tarjetas (CRC) son una técnica para la representación de los sistemas, que tienen como principal objetivo identificar y organizar las clases que se necesitan para implementar el sistema y la relación que guardan entre ellas. La Clase es el objeto que se analiza, la Responsabilidad es cualquier función que la clase sabe o realiza y el Colaborador es el conjunto de clases requeridas para que una clase reciba la información necesaria con el objetivo de completar una responsabilidad. A continuación se muestran las tarjetas CRC generadas para las clases de Symfony:

Tabla 5: Tarjeta 1. Clase WapitiPlugin.

Clase WapitiPlugin

Descripción: Se encarga de controlar el proceso de obtención de parámetro (url) correspondiente a la herramienta de prueba Wapiti.	
Responsabilidades	**Colaboradoras**
Mostrar el nombre, imagen y descripción correspondientes a la herramienta.	WapitiTool
Realiza un comando a partir de los datos especificados por el usuario.	WapitiParameter
Muestra los campos referentes a los parámetros que debe introducir el usuario.	WapitiFormType
Posee la configuración para instalar el plugin.	PluginInterface

Tabla 6: Tarjeta 2. Clase JoomscanPlugin.

Clase JoomscanPlugin	
Descripción: Se encarga de controlar el proceso de obtención de parámetro (url) correspondiente a la herramienta de prueba Joomscan.	
Responsabilidades	**Colaboradoras**
Mostrar el nombre, imagen y descripción correspondientes a la herramienta.	JoomscanTool
Realiza un comando a partir de los datos especificados por el usuario.	JoomscanParameter
Muestra los campos referentes a los parámetros que debe introducir el usuario.	JoomscanFormType
Posee la configuración para instalar el plugin.	PluginInterface

Se pueden encontrar en el Anexo II las tablas restantes correspondiente a las demás clases de Symfony y de Bambú.

3.6. Tareas de Ingeniería

Las tareas de la ingeniería se realizan para especificar las acciones llevadas a cabo por los programadores en cada historia de usuario, haciendo mayor énfasis en los detalles necesarios para lograr una correcta implementación de las mismas. Dichas tareas recogen los siguientes aspectos:

Número de la tarea: Los números deben ser sucesivos.

Número Historia de Usuario: Número de la HU a la que pertenece la tarea.

Nombre Tarea: Nombre que identifica a la tarea.

Tipo de Tarea: Las tareas pueden ser de: Desarrollo, Corrección, Mejora, Otra (Definir).

Puntos Estimados: Tiempo estimado en días que se le asignará a su desarrollo.

Fecha Inicio: Fecha en que comienza el desarrollo de la tarea.

Fecha Fin: Fecha en que termina el desarrollo de la tarea.

Programador Responsable: Nombre y apellidos del programador.

Descripción: Breve descripción de la tarea.

A continuación se muestran las tareas de ingeniería derivadas de la primera HU:

Tabla 7: Tarea 1. Seleccionar herramienta Wapiti.

Tarea de ingeniería	
Número de la tarea: 1	**Número de Historia de Usuario:** 1
Nombre de la Tarea: Seleccionar herramienta Wapiti.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.6
Fecha inicio: 18/4/2016	**Fecha fin:** 20/4/2016
Programador responsable: Carlos A. Méndez Denis y Carlos M. Medina Aguilera	
Descripción: Permite al especialista escoger entre las herramientas que tiene integrada la Plataforma, desde una interfaz gráfica, la herramienta Wapiti para realizar la prueba.	

Tabla 8: Tarea 2. Registrar parámetros (url) de la herramienta Wapiti.

Tarea de ingeniería	
Número de la tarea: 2	**Número de Historia de Usuario:** 1
Nombre de la Tarea: Registrar parámetros (url) de la herramienta Wapiti.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.4
Fecha inicio: 21/4/2016	**Fecha fin:** 22/4/2016
Programador responsable: Carlos A. Méndez Denis y Carlos M. Medina Aguilera	
Descripción: Permite al especialista desde una interfaz gráfica introducir el parámetro (url) necesario para la creación del comando para enviarlo a Bambú.	

Tabla 9: Tarea 3. Ejecutar herramienta Wapiti.

Tarea de ingeniería	
Número de la tarea: 3	**Número de Historia de Usuario:** 1
Nombre de la Tarea: Ejecutar herramienta Wapiti.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.4
Fecha inicio: 25/4/2016	**Fecha fin:** 26/4/2016
Programador responsable: Carlos A. Méndez Denis y Carlos M. Medina Aguilera	

Descripción: El comando creado en el servidor web de Symfony se modifica para ser enviado a Bambú como una tarea, la cual permite que se ejecute la herramienta seleccionada.

Tabla 10: Tarea 4. Crear reporte con las vulnerabilidades encontradas.

Tarea de ingeniería	
Número de la tarea: 4	**Número de Historia de Usuario:** 1
Nombre de la Tarea: Crear reporte con las vulnerabilidades encontradas.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.6
Fecha inicio: 27/4/2016	**Fecha fin:** 29/4/2016
Programador responsable: Carlos A. Méndez Denis y Carlos M. Medina Aguilera	
Descripción: Se crea un reporte sencillo con las vulnerabilidades encontradas, a través de la eliminación de datos innecesarios del resultado de la ejecución de la herramienta.	

Tabla 11: Tarea 5. Mostrar reporte de vulnerabilidades.

Tarea de ingeniería	
Número de la tarea: 5	**Número de Historia de Usuario:** 1
Nombre de la Tarea: Mostrar reporte de vulnerabilidades.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.4
Fecha inicio: 2/5/2016	**Fecha fin:** 3/5/2016
Programador responsable: Carlos A. Méndez Denis y Carlos M. Medina Aguilera	
Descripción: A través de una interfaz gráfica se muestra al especialista el reporte final de vulnerabilidades, detectadas en la ejecución de la herramienta Wapiti.	

Se pueden encontrar en el Anexo III las tareas de ingenierías correspondientes a la HU #2.

3.7. Estándares de Codificación

Los estándares de codificación son utilizados en los proyectos para lograr un código legible y mantener uniformidad en la generación del código por el equipo de desarrollo. A continuación se muestran estándares de codificación utilizados en la implementación de los plugins:

3.7.1. Plugins Symfony

Longitud de línea: Se evitó la utilización de más de 80 caracteres en una línea.

Comentarios: Para comentar en más de una línea se utilizaron los comentarios limitados por /* <Comentario> */.

```
/**
 * Identificador del plugin, debe ser un nombre único
 *
 * @return string
 */
```

Figura 26: Ejemplo de comentarios.

Identación: Se emplearon cuatro espacios como unidad de identación.

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('url', 'text', array(
            'label' => 'URL:', 'required' => false
        ));
}
```

Figura 27: Ejemplo de identación.

Declaraciones: Los métodos fueron declarados con nombres asociados a la función que realizan, donde para los nombres compuestos se definió el primero en minúscula y los demás comenzando con mayúscula. Las variables de clase o método fueron declaradas en minúscula. Las clases se declararon NombHerramientaPlugin donde la inicial del nombre de la herramienta y la inicial de la palabra plugins se especificó en mayúscula.

```
public function getPluginName()
{
    return $this->name;
}
```

```
private $name;

private $tool;

private $parameter;

private $form;
```

```
class WapitiPlugin implements PluginInterface
{
```

Figura 28: Ejemplo de declaraciones.

En la realización de las pruebas unitarias, se añadió la palabra “Test” al final de la declaración del nombre de la clase, y en la declaración de los métodos se le añadió delante la palabra “test”.

Sentencia <if>: Se definió para este tipo de sentencia la siguiente estructura:

```
public function parameterUrl()
{
    if ($this->url != "")
        return " " . $this->url;
    return "";
}
```

Figura 29: Ejemplo de sentencia <if>.

3.7.2. Plugin en Bambú

Comentarios: Se utilizaron los comentarios limitados por #<Comentario> para comentar una línea.

```
# Se debe de implementar de tal forma que arroje como resultado un objeto de tipo Result
```

Figura 30: Ejemplo de comentarios.

Identación: Se emplearon cuatro espacios como unidad de identación.

```
def __init__(self):
    self.__applicationName = 'wapiti'
    self.__applicationPath = ''
    self.__os = OperatingSystem()
```

Figura 31: Ejemplo de identación.

Declaraciones: Los métodos y variables fueron declarados con nombres asociados a la función que realizan, donde para los nombres compuestos, se definió el primero en minúscula y los demás comenzando con mayúscula, en el caso de las variables compuestas por una solo nombre se declararon en minúsculas. Las clases se declararon con la inicial del nombre de la herramienta en mayúscula.

```
def handleTask(self, task):

listFiles = os.getcwd() + '/Resources/ResArachni/' + task.getIdTask() + '.xml'
attributes = Extractor().Extract(listFiles, "xml", "ImplementClass/wapiti.xml")
listFilesToSend = [File(task.getIdTask() + '.xml', os.getcwd() + "/Resources/ResWapiti/")]

class Wapiti(ITaskClient):
```

Figura 32: Ejemplo de declaraciones.

En la realización de las pruebas unitarias, se añadió la palabra “Test_” al comienzo de la declaración del nombre de la clase, y en la declaración de los métodos se le añadió delante la palabra “test_”.

Sentencia <if, else>: Se definieron para estos dos tipos de sentencias la siguiente estructura:

```
if os.path.exists(os.getcwd() + '/Resources/ResWapiti/' + task.getIdTask() + '.xml'):

    listFiles = os.getcwd() + '/Resources/ResArachni/' + task.getIdTask() + '.xml'
    attributes = Extractor().Extract(listFiles, "xml", "ImplementClass/wapiti.xml")
    listFilesToSend = [File(task.getIdTask() + '.xml', os.getcwd() + "/Resources/ResWapiti/")]

    name = "Abortado"
    # try:
    value = self.obtenerStock(task)
    # except:
    # value = "True"
    attr = Attribute(name, "string", value)

    attributes1 = [attr]
    attributes1 += attributes
    attributes = attributes1

    result = Result(str(task.getIdDefinition()) + "-" + str(task.getIdTask()), task.getIdTask(),
                    self.__os.getTimeStamp(), attributes, listFilesToSend)
else:
    name = "Error Wapiti"
    value = "La ejecucion de Wapiti no arrojo ningun resultado."
    attr = Attribute(name, "string", value)
    result = Result(str(task.getIdDefinition()) + "-" + str(task.getIdTask()), task.getIdTask(),
                    self.__os.getTimeStamp(), [attr], [])
```

Figura 33: Ejemplo de sentencia <if, else>.

3.8. Conclusiones

En este capítulo se llevó a cabo la fase de diseño definida por la metodología utilizada en la presente investigación, lo que posibilitó definir la arquitectura del sistema. También se definió el patrón arquitectónico y los patrones de diseño que fueron utilizados en el desarrollo de los plugins necesarios para la integración de nuevas herramientas.

4. Introducción

El proceso de pruebas es uno de los pilares de XP por lo que esta metodología define probar constantemente mientras sea posible, para así aumentar la calidad del sistema. En el presente capítulo se define la estrategia de prueba a utilizar, la cual está compuesta por las pruebas unitarias, pruebas de integración y pruebas de aceptación.

4.1. Estrategia de Prueba

Las pruebas se realizaron para evaluar la calidad de un producto y mejorarlo mediante la identificación de los defectos y los problemas. XP divide las pruebas del sistema en dos grupos: las pruebas unitarias, encargadas de verificar el código, y las pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñadas por el cliente final. La estrategia de prueba utilizada en la presente investigación está compuesta por las pruebas que se describen a continuación:

4.1.1. Pruebas Unitarias

Son consideradas una de las etapas imprescindibles de XP. Este tipo de prueba se le aplica a todos los métodos que se implementen una vez terminados y antes de ser publicados. Que todo el código realizado pase correctamente las pruebas unitarias es lo que garantiza que funcionen en conjunto con los demás. Las pruebas deben ser guardadas en conjunto con el sistema, para que puedan ser utilizadas por otros desarrolladores, en caso de tener que corregir, cambiar o recodificar parte del mismo (38).

Las pruebas unitarias que se le realizaron a los plugins desarrollados para integrar las herramientas son los test unitarios. Para la realización de las mismas se utilizaron las herramientas PyUnit que es un framework para la elaboración de pruebas unitarias en Python y PHPUnit para las de PHP. Estas pruebas fueron llevadas a cabo por los desarrolladores, encargadas de verificar el código de forma automática. Después de una primera iteración se obtuvieron los resultados representados en la figura 37, donde para los plugins de Wapiti y Joomscan se detectaron 7 y 6 no conformidades respectivamente.

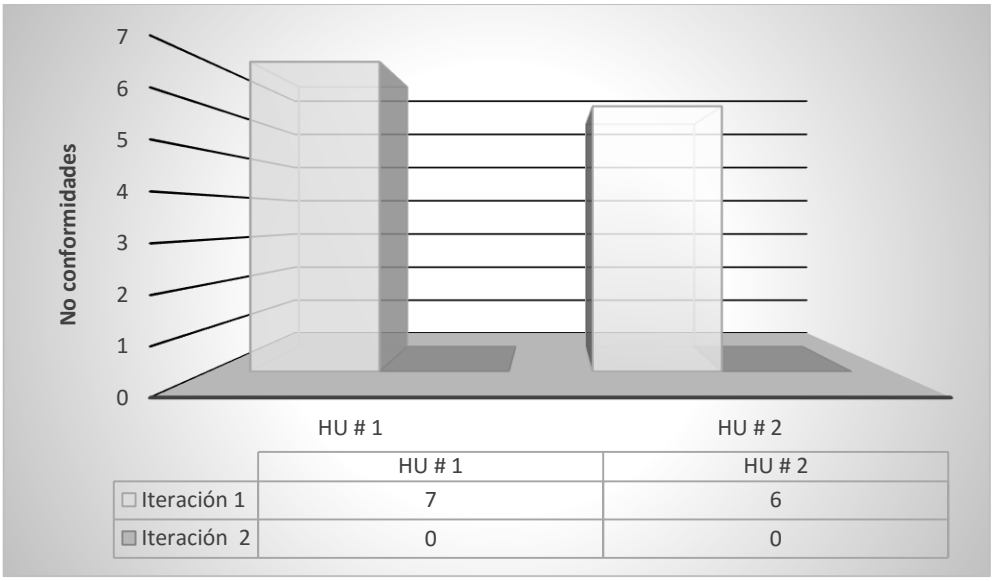


Figura 34: Resultados de las pruebas unitarias.

Después de resolver las no conformidades encontradas, se realizó una segunda iteración, donde los resultados arrojados fueron los esperados. A continuación se muestran dichos resultados:

```
root@debian:/var/www/platsi# phpunit -c app src/LABSI/BamplusPlugins/Wapiti
PHPUnit 4.2.6 by Sebastian Bergmann.

Configuration read from /var/www/platsi/app/phpunit.xml.dist

.......

Time: 42 ms, Memory: 2.50Mb

OK (7 tests, 14 assertions)
```

Figura 35: Resultado de la Prueba Unitaria para el Plugin Wapiti (Symfony).

```
<terminated> /usr/local/bambu/client/test/test_wapiti.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.083s

OK
```

Figura 36: Resultado de la Prueba Unitaria para el Plugin Wapiti (Bambú).

Figura 37: Resultado de la Prueba Unitaria para el Plugin Joomscan (Symfony).

Figura 38: Resultado de la Prueba Unitaria para el Plugin Joomscan (Bambú).

4.1.2. Pruebas de Integración

Se utilizan para asegurar que los componentes en el modelo de implementación operen correctamente cuando son combinados. Estas pruebas descubren errores en las especificaciones de las interfaces de los paquetes y debe ser responsabilidad de desarrolladores. Es el proceso de combinar y probar múltiples componentes juntos. El objetivo es tomar los componentes probados en unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño (39).

Con el objetivo de verificar que la integración de las nuevas herramientas no afectó el funcionamiento de la Plataforma, en el presente trabajo se realizaron pruebas de integración de tipo ascendente, comenzando por los nuevos plugins creados y luego por los que estaban integrados anteriormente. Para la realización de estas pruebas se crearon diferentes niveles, donde el nivel atómico son los nuevos plugins, su nivel superior está compuesto por los tres plugins que fueron integrados en la segunda versión de la Plataforma y como primer nivel están los plugins de las dos primeras herramientas que fueron integradas. Para un mayor entendimiento se muestra la siguiente imagen:

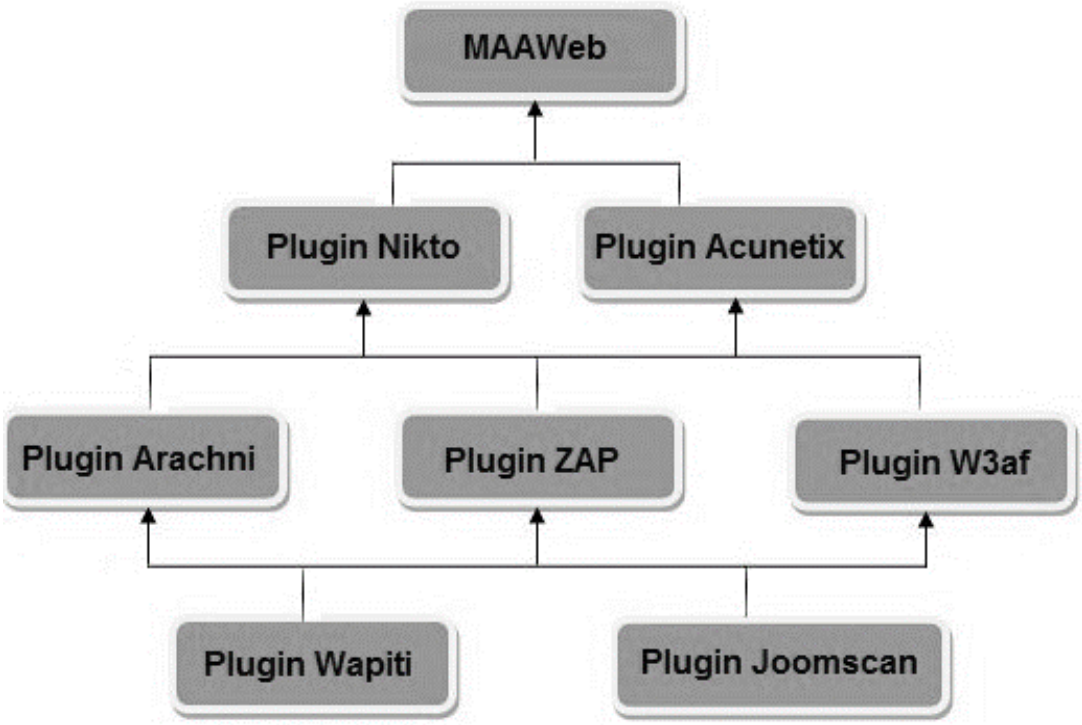


Figura 39: Pruebas de Integración Ascendente.

El resultado de las pruebas de integración fue satisfactorio ya que se comprobó que la integración de los nuevos plugins no afectó el funcionamiento de las herramientas ni el de los demás procesos del MAAWeb. Los resultados obtenidos se muestran a continuación:

Tabla 12: Caso de Pruebas de Integración.

Funcionalidad	Condiciones de Ejecución	Escenario de Prueba	Resultado previsto	Resultado real
Realizar prueba de seguridad a aplicaciones web con la herramienta Nikto.	Introducir los parámetros requeridos por la herramienta seleccionada. Dirección IP o URL: http://127.0.0.1/dvwa/	Envío de parámetros desde la aplicación web en Symfony hasta Bambú, para formar el comando que ejecuta la herramienta Nikto y devuelva el resultado de la ejecución.	Se espera que se muestre en la aplicación web de la Plataforma: "Los datos se han enviado de forma satisfactoria al servidor y al terminar la ejecución de la herramienta se muestre el reporte	El resultado fue: "Los datos se han enviado de forma satisfactoria al servidor y se muestra el reporte con las vulnerabilidades encontradas".

			de vulnerabilidades".	
Realizar prueba de seguridad a aplicaciones web con la herramienta Acunetix.	Introducir los parámetros requeridos por la herramienta seleccionada. Dirección IP o URL: http://127.0.0.1/dvwa/	Envío de parámetros desde la aplicación web en Symfony hasta Bambú, para formar el comando que ejecuta la herramienta Acunetix y devuelva el resultado de la ejecución.	Se espera que se muestre en la aplicación web de la Plataforma: "Los datos se han enviado de forma satisfactoria al servidor y al terminar la ejecución de la herramienta se muestre el reporte de vulnerabilidades".	El resultado fue: "Los datos se han enviado de forma satisfactoria al servidor y se muestra el reporte con las vulnerabilidades encontradas".
Realizar prueba de seguridad a aplicaciones web con la herramienta Arachni.	Introducir los parámetros requeridos por la herramienta seleccionada. Dirección IP o URL: http://127.0.0.1/dvwa/	Envío de parámetros desde la aplicación web en Symfony hasta Bambú, para formar el comando que ejecuta la herramienta Arachni y devuelva el resultado de la ejecución.	Se espera que se muestre en la aplicación web de la Plataforma: "Los datos se han enviado de forma satisfactoria al servidor y al terminar la ejecución de la herramienta se muestre el reporte de vulnerabilidades".	El resultado fue: "Los datos se han enviado de forma satisfactoria al servidor y se muestra el reporte con las vulnerabilidades encontradas".
Realizar prueba de seguridad a aplicaciones web con la herramienta Zap.	Introducir los parámetros requeridos por la herramienta seleccionada. Dirección IP o URL: http://127.0.0.1/dvwa/	Envío de parámetros desde la aplicación web en Symfony hasta Bambú, para formar el comando que ejecuta la	Se espera que se muestre en la aplicación web de la Plataforma: "Los datos se han enviado de forma satisfactoria al	El resultado fue: "Los datos se han enviado de forma satisfactoria al servidor y se muestra el

		herramienta Zap y devuelva el resultado de la ejecución.	servidor y al terminar la ejecución de la herramienta se muestre el reporte de vulnerabilidades".	reporte con las vulnerabilidades encontradas".
Realizar prueba de seguridad a aplicaciones web con la herramienta W3af.	Introducir los parámetros requeridos por la herramienta seleccionada. Dirección IP o URL: http://127.0.0.1/dvwa/	Envío de parámetros desde la aplicación web en Symfony hasta Bambú, para formar el comando que ejecuta la herramienta W3af y devuelva el resultado de la ejecución.	Se espera que se muestre en la aplicación web de la Plataforma: "Los datos se han enviado de forma satisfactoria al servidor y al terminar la ejecución de la herramienta se muestre el reporte de vulnerabilidades".	El resultado fue: "Los datos se han enviado de forma satisfactoria al servidor y se muestra el reporte con las vulnerabilidades encontradas".
Realizar prueba de seguridad a aplicaciones web con la herramienta Wapiti.	Introducir los parámetros requeridos por la herramienta seleccionada. Dirección IP o URL: http://127.0.0.1/dvwa/	Envío de parámetros desde la aplicación web en Symfony hasta Bambú, para formar el comando que ejecuta la herramienta Wapiti y devuelva el resultado de la ejecución.	Se espera que se muestre en la aplicación web de la Plataforma: "Los datos se han enviado de forma satisfactoria al servidor y al terminar la ejecución de la herramienta se muestre el reporte de vulnerabilidades".	El resultado fue: "Los datos se han enviado de forma satisfactoria al servidor y se muestra el reporte con las vulnerabilidades encontradas".
Realizar prueba de seguridad a	Introducir los parámetros requeridos	Envío de parámetros desde	Se espera que se muestre en la	El resultado fue: "Los datos se

aplicaciones web con la herramienta Joomscan.	por la herramienta seleccionada. Dirección IP o URL: http://127.0.0.1/Joomla/	la aplicación web en Symfony hasta Bambú, para formar el comando que ejecuta la herramienta Joomscan y devuelva el resultado de la ejecución.	aplicación web de la Plataforma: "Los datos se han enviado de forma satisfactoria al servidor y al terminar la ejecución de la herramienta se muestre el reporte de vulnerabilidades".	han enviado de forma satisfactoria al servidor y se muestra el reporte con las vulnerabilidades encontradas".

4.1.3. Pruebas de Aceptación

Son creadas en base a las historias de usuarios. El cliente debe especificar uno o diversos escenarios para comprobar que una HU ha sido correctamente implementada. Las pruebas de aceptación son consideradas como Pruebas de Caja Negra. Los clientes son responsables de verificar que los resultados de estas pruebas sean correctos. Así mismo, en caso de que fallen varias pruebas, se debe indicar el orden de prioridad de resolución. Una HU no se puede considerar terminada hasta tanto pase correctamente todas las pruebas de aceptación. Dado que la responsabilidad es grupal, es recomendable publicar los resultados de las pruebas de aceptación, de manera que todo el equipo esté al tanto de esta información (38).

A continuación se muestra un escenario para los casos de pruebas de aceptación realizados. Los restantes se comportaron de forma similar, estos se pueden observar en el Anexo IV.

Tabla 13: Caso de Pruebas de Aceptación: HU1_CP1.

Caso de Prueba de Aceptación	
Código: HU1_CP1	**Historia de Usuario:** 1
Nombre: Insertar parámetro (url) correcto de la herramienta Wapiti.	
Descripción: Se desea probar que la Plataforma permite al especialista introducir el parámetro (url) correcto correspondiente a Wapiti.	
Condiciones de Ejecución: Url válida para Wapiti.	
Entrada/Pasos de Ejecución: • El especialista presiona el link "Lista de Auditorías Web". • El especialista presiona el botón correspondiente a la herramienta seleccionada. • El especialista introduce el parámetro (url) requerido por la herramienta.	

• El especialista presiona el botón Aceptar para comenzar la prueba.
Resultado Esperado: La Plataforma almacena la url introducida por el especialista conformando el comando correspondiente a la herramienta Wapiti.
Evaluación de la Prueba: Satisfactoria.

Tabla 14: Caso de Pruebas de Aceptación: HU1_CP2.

Caso de Prueba de Aceptación	
Código: HU1_CP2	**Historia de Usuario:** 1
Nombre: Insertar parámetro (url) erróneo de la herramienta Wapiti.	
Descripción: Se desea probar que la Plataforma no permite al especialista introducir el parámetro (url) incorrecto correspondiente a Wapiti.	
Condiciones de Ejecución: Url inválida para Wapiti.	
Entrada/Pasos de Ejecución: • El especialista presiona el enlace “Lista de Auditorías Web”. • El especialista presiona el botón correspondiente a la herramienta seleccionada. • El especialista introduce el parámetro (url) requerido por la herramienta. • El especialista presiona el botón Aceptar para comenzar la prueba.	
Resultado Esperado: La Plataforma mostrará una notificación al especialista acerca de los datos erróneos o incompletos.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 15: Caso de Pruebas de Aceptación: HU1_CP3.

Caso de Prueba de Aceptación	
Código: HU1_CP3	**Historia de Usuario:** 1
Nombre: Modificación de los resultados de la prueba.	
Descripción: Probar que la Plataforma muestra el reporte final con las vulnerabilidades detectadas como resultado de la prueba realizada con la herramienta Wapiti.	
Condiciones de Ejecución: Ejecución de la prueba.	
Entrada/Pasos de Ejecución: • Una vez obtenidos los parámetros de la herramienta se conforma el comando para ejecutar. • El comando es enviado al Distribuidor de Tareas (Bambú). • En Bambú, se modifica el comando para ejecutar la herramienta Wapiti. • Se obtiene el reporte generado por la herramienta.	
Resultado Esperado: Se modificará el reporte generado por la herramienta, eliminando los datos innecesarios que se obtienen de la prueba. Se mostrará el reporte final con las vulnerabilidades detectadas.	

Evaluación de la Prueba: Satisfactoria.

Fue necesaria la realización de dos iteraciones. Se obtuvieron los resultados que se observan en la figura 40, donde para las HU 1 y HU 2 se obtuvieron 8 y 9 no conformidades respectivamente. En la segunda iteración se solucionaron las no conformidades detectadas, obteniéndose los resultados esperados.

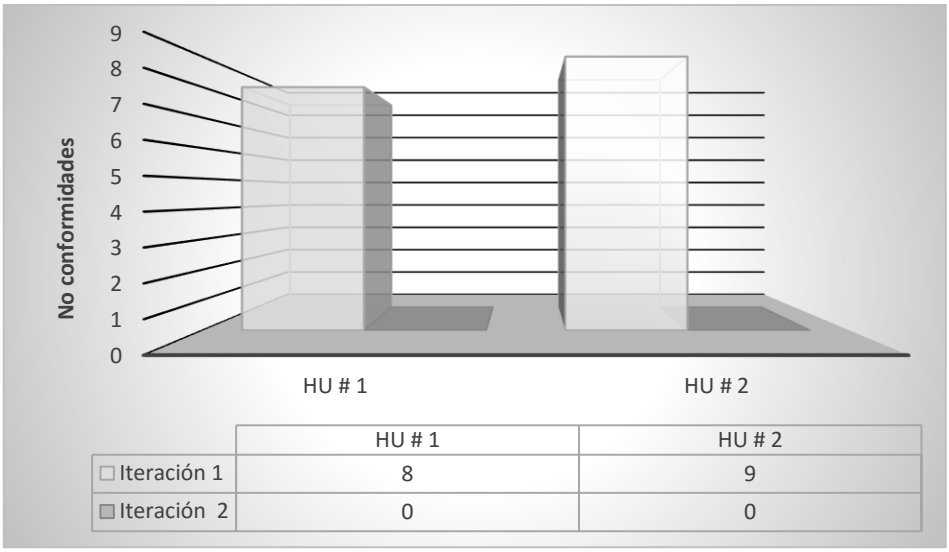


Figura 40: Resultados de las pruebas de aceptación.

4.2. Conclusiones

Con la realización de las pruebas mencionadas anteriormente se detectaron un grupo de no conformidades, las cuales fueron solucionadas con la realización de nuevas iteraciones. El cumplimiento de la estrategia de pruebas definido permitió comprobar que la solución propuesta cumple con el objetivo de la presente investigación.

Conclusiones generales

Finalizada la investigación se le dio solución al problema planteado y se cumple el objetivo del presente trabajo. Por lo expuesto anteriormente, se arriba a las siguientes conclusiones:

- En el proceso de investigación se caracterizaron y seleccionaron las herramientas Wapiti y Joomscan para su integración a la Plataforma, ya que fueron las que cumplieron con las características definidas por el equipo de desarrollo de PlatSI.

- Para la integración de las nuevas herramientas fue necesario la implementación de una serie de plugins que no afectaron los demás proceso que se realizan en la Plataforma.

- Con la realización de la estrategia de prueba definida se verificó que la solución propuesta cumplió con el objetivo de la presente investigación.

- Con la integración de las herramientas mencionadas anteriormente la Plataforma aumentó su capacidad para detectar nuevas vulnerabilidades, por lo que se apoyó el proceso de correlación y se disminuye la probabilidad de ocurrencia de falsos positivos.

A continuación, se listan las recomendaciones para posibles mejoras:

- Dar soporte a las herramientas integradas con las nuevas versiones que se liberen.
- En futuras versiones de la Plataforma, integrar nuevas herramientas que detecten vulnerabilidades en aplicaciones web desarrolladas en los demás CMS.

Referencias Bibliográficas

1. Informe sobre las Amenazas a la Seguridad en Internet. [En línea] [Citado el: 23 de 5 de 2016.] https://www.symantec.com/es/mx/security_response/publications/threatreport.jsp.

2. Las bases estratégicas de la informatización cubana. [En línea] [Citado el: 23 de 5 de 2016.] http://www.mincom.gob.cu/?q=node/803.

3. Mifsud, Elvira. Observatorio Tecnológico. [En línea] 26 de 3 de 2012. [Citado el: 23 de 5 de 2016.] http://recursostic.educacion.es/observatorio/web/ca/software/software-general/1040-introduccion-a-la-seguridad-informatica?start=1.

4. [En línea] [Citado el: 15 de 12 de 2015.] http://www.uci.cu/?q=mision.

5. [En línea] [Citado el: 15 de 12 de 2015.] http://gespro.tlm.prod.uci.cu.

6. Jara, Héctor y G. Pacheco, Federico. Ethical Hacking 2.0. Argentina : s.n., 2012.

7. Benchimol, Daniel I. Hacking desde cero. Argentina : s.n., 2011.

8. González, Eva González de Canales. Trabajo de Fin de Máster especialidad Seguridad en servicios y aplicaciones. Generación de reportes de vulnerabilidades y amenazas para aplicaciones web. UNIVERSITAT OBERTA DE CATALUNYA : s.n., 2014.

9. OWASP Top 10 - Los diez riesgos más críticos en Aplicaciones Web. 2015.

10. Gabriel Diaz Orueta. Departamento de Sistemas de Comunicación y Control. Departamento de Sistemas de Comunicación y Control. [En línea] 22 de Octubre de 2012. [Citado el: 15 de 1 de 2016.] http://www.scc.uned.es/jornadasmaster/pdf/Charla2.pdf.

11. Janusec. [En línea] [Citado el: 22 de 2 de 2016.] http://www.janusec.com/.

12. Syhunt. [En línea] [Citado el: 23 de 2 de 2016.] http://www.syhunt.com/en/index.php.

13. Un vistazo a NetSparker de Mavituna Security. Security by Default. [En línea] 20 de Enero de 2010. [Citado el: 23 de 2 de 2016.] http://www.securitybydefault.com/2010/01/un-vistazo-netsparker-de-mavituna.html.

14. CYBORG. [En línea] 13 de Octubre de 2015. [Citado el: 24 de 2 de 2016.] http://cyborg.ztrela.com/cms-explorer.php/.

15. Security-EC, Revista digital de Seguridad Informática. [En línea] [Citado el: 24 de 2 de 2016.] http://securityec.com/herramienta-para-buscar-vulnerabilidades-en-wordpress-wpsfp/.

16. Blog. [En línea] [Citado el: 24 de 2 de 2016.] http://urlgeek.blogspot.com/2013/05/como-instalar-el-joomscan-en-ubuntu.html.

17. Rabaez, David. WebMaster. [En línea] 10 de Agosto de 2014. [Citado el: 25 de 2 de 2016.] http://www.davidrabaez.com/wpscan-una-magnifica-herramienta-de-seguridad-para-auditar-wordpress/.

18. Quezada, Alonso Caballero. ReYDeS blog. [En línea] 22 de Enero de 2014. [Citado el: 25 de 2 de 2016.] http://www.reydes.com/d/?q=Instalacion_de_Wapiti.

19. Websecurify – entorno de prueba de seguridad Web. [En línea] 23 de Septiembre de 2009. [Citado el: 25 de 2 de 2016.] https://seguinfo.wordpress.com/2009/09/23/websecurify-%E2%80%93-entorno-de-prueba-de-seguridad-web/.

20. Seguridad Informática [Ethical Hacking,Pen-test,Anti Script-kiddies]. [En línea] 17 de 12 de 2012. [Citado el: 26 de 2 de 2016.] http://antisec-security.blogspot.com/2012/12/wfuzz-supongo-que-la-mayoria-que-esta.html.

21. Vázquez, Yamilia Izquierdo y Ramos Medina , Erenio. Trabajo de Diploma para optar por el título de Ingeniería en Ciencias Informáticas: Sistema para la ejecución de pruebas de seguridad y procesamiento de los resultados en una arquitectura distribuida. La Habana, Universidad de las Ciencias Informáticas. : s.n., 2012.

22. Eguiluz, Javier. Desarrollo web ágil con Symfony2. 2013.

23. Ramírez, Ing.Danay Pérez. METODOLOGÍAS ÁGILES.¿CÓMO DESARROLLO UTILIZANDO XP? [En línea] 2008. http://ccia.cujae.edu.cu/index.php/siia/siia2008/paper/viewFile/1174/246.

24. Duque, Raúl González. Python para todos.

25. Sæther Bakken, Stig, y otros. Manual de PHP.

26. A. White, Stephen y Miers, Derek. Guía de Referencia y Modelado BPMN. USA, Florida : s.n., 2009.

27. Sitio Oficial. [En línea] [Citado el: 9 de 3 de 2016.] https://www.visual-paradigm.com/.

28. [En línea] 10 de Junio de 2014. [Citado el: 10 de 3 de 2016.] http://hipertextual.com/archivo/2014/06/pycharm-ide-python/.

29. [En línea] [Citado el: 10 de 3 de 2016.] http://www.editoresdecodigo.com/2014/06/descargar-phpstorm-full-ide-para-php-y-mas.html.

30. Ise Morales, Delís, y otros. CIVITEC PlatSI.

31. Escribano, Gerardo Fernández. Introducción a Extreme Programming. Pág 6.

32. Beck, Kent. Extreme Programming Explained. s.l. : 1 edición: Addison Wesley Pub Co, 1999.

33. Voitgmann. [En línea] [Citado el: 18 de 4 de 2016.] http://www.voigtmann.de/es/desarrollo-de-software.

34. Camacho, Erika, Cardeso, Fabio y Nuñes, Gabriel. Arquitecturas de Software. 2004.

35. Potencier, Fabien y Zaninotto, Francois. Symfony, la guía definitiva. 2008.

36. [En línea] [Citado el: 20 de 4 de 2016.] http://www.practicasdesoftware.com.ar/2011/03/patrones-grasp/.

37. Rodríguez, Miguel Arlandy. [En línea] 4 de Octubre de 2011. [Citado el: 22 de 4 de 2016.] https://www.adictosaltrabajo.com/tutoriales/patron-template-method/.

38. Joskowicz, Ing. José. Reglas Y Prácticas en eXtreme Programing. 2008.

39. Flujo de pruebas de un software. [En línea] [Citado el: 10 de 5 de 2016.] http://www.ecured.cu/Flujo_de_pruebas_de_un_software#Prueba_de_integraci.C3.B3n.

40. [En línea] [Citado el: 5 de 3 de 2016.] http://www.cstl.com/Qualys/QualysGuard/QualysGuard-Web-Applictaion.pdf.

41. Web Site Security Audit - WSSA by Beyond Security. [En línea] [Citado el: 2 de 3 de 2016.] http://www.beyondsecurity.com/vulnerability-scanner.html.

42. [En línea] [Citado el: 2 de 3 de 2016.] https://www.beyondtrust.com/products/retina-web-security-scanning/.

43. [En línea] [Citado el: 3 de 3 de 2016.] http://www.tripwire.com/it-security-software/enterprise-vulnerability-management/web-application-vulnerability-scanning/.

44. [En línea] [Citado el: 3 de 3 de 2016.] http://www.mcafee.com/us/products/vulnerability-manager-end-of-life.aspx.

45. [En línea] [Citado el: 4 de 3 de 2016.] http://www.tenable.com/products/nessus-vulnerability-scanner&dt=-&s=-&r=DQE.

46. [En línea] [Citado el: 4 de 3 de 2016.] http://www.rapid7.com/products/&dt=-&s=-&r=DQE.

47. [En línea] 29 de Abril de 2013. [Citado el: 4 de 3 de 2016.] http://www.tuexperto.com/2013/04/29/hp-webinspect-10-0-analiza-las-aplicaciones-web-en-busca-de-agujeros/.

48. [En línea] [Citado el: 4 de 3 de 2016.] http://www-01.ibm.com/software/bo/rational/security/.

49. [En línea] [Citado el: 5 de 3 de 2016.] http://beefproject.com/&dt=-&s=-&r=DQE.

50. [En línea] [Citado el: 5 de 3 de 2016.] https://portswigger.net/burp/.

51. [En línea] [Citado el: 5 de 3 de 2016.] https://www.owasp.org/index.php/OWASP_Xenotix_XSS_Exploit_Framework.

Bibliografía

1. Filipiak, Joanna María. Herramienta de análisis estático de código para encontrar vulnerabilidades de seguridad en las aplicaciones Web. 2009. UNIVERSIDAD CARLOS III DEMADRID INGENIERÍA INFORMÁTICA.
2. H. Canós, José; Letelier, Patricio; Penadés, María Carmen. Metodologías Ágiles en el Desarrollo de Software. DSIC-Universidad Politécnica de Valencia. Camino de Vera s/n, Valencia.
3. Benchimol, Daniel I. Hacking desde cero. Argentina : s.n., 2011.
4. Escribano, Gerardo Fernández. Introducción a Extreme Programming. 2002.
5. Mestras, Juan Pavón. Estructura de las Aplicaciones Orientadas a Objetos. El patrón Modelo-Vista-Controlador (MVC). Departamento de Ingeniería del Software e Inteligencia Artificial. Universidad Complutense Madrid. 2008-2009.
6. Arauzo Azofra, Antonio y Revilla, Ernesto. Una pequeña introducción a Python. Granada: s.n., 2004.
7. Gutiérrez, J. J., Escalona M. J., Mejías M., Torres J. PRUEBAS DEL SISTEMA ENROGRAMACIÓN EXTREMA. Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla.
8. Aspectos Básicos de la Seguridad en Aplicaciones Web. Se puede encontrar en: http://www.seguridad.unam.mx/documento/?id=17.
9. Joskowicz, Ing. José. Reglas y Prácticas en eXtreme Programming. 2008.
10. Roger S. Pressman. Ingeniería de Software. Un enfoque práctico. (5ta edición).
11. Roger S. Pressman. Ingeniería de Software. Un enfoque práctico. (7ma edición).
12. Raúl González Duque. Python para todos.
13. Sæther Bakken, Stig, y otros. Manual de PHP.
14. Fabien Potencier, Francois Zaninotto. Symfony, la guía definitiva.
15. Informe sobre las Amenazas a la Seguridad en Internet. [En línea] https://www.symantec.com/es/mx/security_response/publications/threatreport.jsp.
16. Las bases estratégicas de la informatización cubana. [En línea] http://www.mincom.gob.cu/?q=node/803.

17. Mifsud, Elvira. Observatorio Tecnológico. [En línea] 26 de 3 de 2012. http://recursostic.educacion.es/observatorio/web/ca/software/software-general/1040-introduccion-a-la-seguridad-informatica?start=1.

18. González, Eva González de Canales. Trabajo de Fin de Máster especialidad Seguridad en servicios y aplicaciones. Generación de reportes de vulnerabilidades y amenazas para aplicaciones web. UNIVERSITAT OBERTA DE CATALUNYA : s.n., 2014.

19. Gabriel Diaz Orueta. Departamento de Sistemas de Comunicación y Control. Departamento de Sistemas de Comunicación y Control. [En línea] 22 de Octubre de 2012. http://www.scc.uned.es/jornadasmaster/pdf/Charla2.pdf.

20. Janusec. [En línea] http://www.janusec.com/.

21. Syhunt. [En línea] http://www.syhunt.com/en/index.php.

22. Un vistazo a NetSparker de Mavituna Security. Security by Default. [En línea] 20 de Enero de 2010. http://www.securitybydefault.com/2010/01/un-vistazo-netsparker-de-mavituna.html.

23. CYBORG. [En línea] 13 de Octubre de 2015. http://cyborg.ztrela.com/cms-explorer.php/.

24. Security-EC, Revista digital de Seguridad Informática. [En línea] http://securityec.com/herramienta-para-buscar-vulnerabilidades-en-wordpress-wpsfp/.

25. Rabaez, David. WebMaster. [En línea] 10 de Agosto de 2014. [http://www.davidrabaez.com/wpscan-una-magnifica-herramienta-de-seguridad-para-auditar-wordpress/.

26. Quezada, Alonso Caballero. ReYDeS blog. [En línea] 22 de Enero de 2014. http://www.reydes.com/d/?q=Instalacion_de_Wapiti.

27. Websecurify – entorno de prueba de seguridad Web. [En línea] 23 de Septiembre de 2009. https://seguinfo.wordpress.com/2009/09/23/websecurify-%E2%80%93-entorno-de-prueba-de-seguridad-web/.

28. Seguridad Informática [Ethical Hacking,Pen-test,Anti Script-kiddies]. [En línea] 17 de 12 de 2012. http://antisec-security.blogspot.com/2012/12/wfuzz-supongo-que-la-mayoria-que-esta.html.

29. Vázquez, Yamilia Izquierdo y Ramos Medina , Erenio. Trabajo de Diploma para optar por el título de Ingeniería en Ciencias Informáticas: Sistema para la ejecución de pruebas de seguridad y procesamiento de los resultados en una arquitectura distribuida. La Habana, Universidad de las Ciencias Informáticas. : s.n., 2012.

30. A. White, Stephen y Miers, Derek. Guía de Referencia y Modelado BPMN. USA, Florida : s.n., 2009.

31. Beck, Kent. Extreme Programming Explained. s.l. : 1 edición: Addison Wesley Pub Co, 1999.

32. Voitgmann. [En línea] http://www.voigtmann.de/es/desarrollo-de-software.

33. Camacho, Erika, Cardeso, Fabio y Nuñes, Gabriel. Arquitecturas de Software. 2004.

34. Flujo de pruebas de un software. [En línea] http://www.ecured.cu/Flujo_de_pruebas_de_un_software#Prueba_de_integraci.C3.B3n.

35. Web Site Security Audit - WSSA by Beyond Security. [En línea] http://www.beyondsecurity.com/vulnerability-scanner.html.

36. [En línea] [Citado el: 15 de 12 de 2015.] http://www.uci.cu/?q=mision.

37. [En línea] [Citado el: 15 de 12 de 2015.] http://gespro.tlm.prod.uci.cu.

38. [En línea] 10 de Junio de 2014. [Citado el: 10 de 3 de 2016.] http://hipertextual.com/archivo/2014/06/pycharm-ide-python/.

39. [En línea] [Citado el: 10 de 3 de 2016.] http://www.editoresdecodigo.com/2014/06/descargar-phpstorm-full-ide-para-php-y-mas.html.

40. [En línea] [Citado el: 20 de 4 de 2016.] http://www.practicasdesoftware.com.ar/2011/03/patrones-grasp/.

41. [En línea] [Citado el: 5 de 3 de 2016.] http://www.cstl.com/Qualys/QualysGuard/QualysGuard-Web-Applictaion.pdf.

42. [En línea] [Citado el: 2 de 3 de 2016.] https://www.beyondtrust.com/products/retina-web-security-scanning/.

43. [En línea] [Citado el: 3 de 3 de 2016.] http://www.tripwire.com/it-security-software/enterprise-vulnerability-management/web-application-vulnerability-scanning/.

44. [En línea] [Citado el: 3 de 3 de 2016.] http://www.mcafee.com/us/products/vulnerability-manager-end-of-life.aspx.

45. [En línea] [Citado el: 4 de 3 de 2016.] http://www.tenable.com/products/nessus-vulnerability-scanner&dt=-&s=-&r=DQE.

46. [En línea] [Citado el: 4 de 3 de 2016.] http://www.rapid7.com/products/&dt=-&s=-&r=DQE.

47. [En línea] 29 de Abril de 2013. [Citado el: 4 de 3 de 2016.] http://www.tuexperto.com/2013/04/29/hp-webinspect-10-0-analiza-las-aplicaciones-web-en-busca-de-agujeros/.

48. [En línea] [Citado el: 4 de 3 de 2016.] http://www-01.ibm.com/software/bo/rational/security/.

49. [En línea] [Citado el: 5 de 3 de 2016.] http://beefproject.com/&dt=-&s=-&r=DQE.

50. [En línea] [Citado el: 5 de 3 de 2016.] https://portswigger.net/burp/.

51. [En línea] [Citado el: 5 de 3 de 2016.] https://www.owasp.org/index.php/OWASP_Xenotix_XSS_Exploit_Framework.

52. Jara, Héctor y G. Pacheco, Federico. Ethical Hacking 2.0. Argentina : s.n., 2012.

53. OWASP Top 10 - Los diez riesgos más críticos en Aplicaciones Web. 2015.

Glosario de Términos

Aplicación Web: Es una aplicación de software que se codifica en un lenguaje soportado por los navegadores Web.

BPMN: Del inglés Business Process Modeling Notation, en español Notación para el Modelado de Procesos de Negocio, se utilizó para modelar el proceso de negocio en la detección de vulnerabilidades.

Bundle: Empaquetado de funcionalidades que tienen una responsabilidad determinada dentro de una aplicación con Symfony2.

Framework: Generadores de aplicación que se relacionan directamente con un dominio específico, es decir, con una familia de problemas relacionados. Además tienen la capacidad para promover la reutilización del código del diseño y el código fuente.

HTML: Del inglés Hyper Text Markup Language, en español Lenguaje de marcación de Hipertexto, se puede definir como el lenguaje que se utiliza para la elaboración de páginas Web.

ICE: Proporciona herramientas, APIs y soporte de bibliotecas para construir aplicaciones cliente-servidor orientadas a objetos.

MVC: Del inglés Model-View-Controller, en español Modelo-Vista-Controlador, patrón utilizado en el diseño y desarrollo web.

PHP: Lenguaje de programación interpretado, diseñado originalmente para la creación de páginas Web dinámicas.

Plugin: Pequeño programa que proporciona alguna funcionalidad específica a otra aplicación mayor o más compleja.

Vulnerabilidad: Debilidad que posee un sistema que puede ser utilizada para causar algún daño, poniendo en riesgo la confidencialidad, integridad, disponibilidad del sistema o de sus datos y aplicaciones. Representan las debilidades o aspectos atacables en un sistema informático.

XML: Del inglés Extensible Markup Language; lenguaje de descripción de páginas de Internet, diseñado con la intención de reemplazar al estándar actual HTML.

XP: Es una metodología de desarrollo ágil centrada en potenciar las relaciones interpersonales como clave para obtener el éxito en el desarrollo de software, promoviendo el trabajo en equipo, la comunicación con el cliente, proporcionando a cada uno de estos un buen clima de trabajo.