PROYECTO FIN DE MÁSTER EN

INGENIERÍA INFORMÁTICA

CURSO 2015-2016

Herramientas de Análisis Dinámico de Vulnerabilidades en Aplicaciones Web

Esteban Alejandro Armas Vega

Directores:

Luis Javier García Villalba

Ana Lucila Sandoval Orozco

Departamento de Ingeniería del Software e Inteligencia Artificial

Convocatoria de Septiembre

Calificación: 10 - Sobresaliente

MÁSTER EN INGENIERÍA INFORMÁTICA

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

El presente Trabajo Fin de Máster se enmarca dentro de un proyecto de investigación titulado RAMSES aprobado por la Comisión Europea dentro del Programa Marco de Investigación e Innovación Horizonte 2020 y en el que participa el Grupo GASS del Departamento de Ingeniería del Software e Inteligencia Artificial de la Facultad de Informática de la Universidad Complutense de Madrid (Grupo de Análisis, Seguridad y Sistemas, http://gass.ucm.es, grupo 910623 del catálogo de grupos de investigación reconocidos por la UCM).

Por razones de confidencialidad del proyecto se ha omitido información del trabajo desarrollado para no infringir la normativa correspondiente

.

_____

Esteban Alejandro Armas Vega

_____ _____

Luis Javier García Villalba Ana Lucila Sandoval Orozco

Abstract

It is common to use automatic detection tools to review the security vulnerabilities of web applications and in the literature there are many studies that compare the detection capabilities of these tools. These studies analyze vulnerable web applications with dynamic analysis tools and then comparing the resulting reports. From the comparative and analysis, scanners are classified according to the number of vulnerabilities found correctly. In this study, unlike other studies, wants to go much further and using a different test model than the typically used in previews works, verify vulnerabilities that the scanners can detect, the tests that they perform and the vulnerabilities that try to detect from those who really are present into the analyzed web application. Through this, it can tell if the performed tests are efficient and know two relevant aspects about these scanners: What tests are not made by the tool even if the scanner are capable to do it and what tests to find vulnerabilities are made but do not report any vulnerability of those that really has the web application. The usual test model has been modified adding an IDS between the detection tools and the vulnerable web applications. From all collected information, was proofed it, in some cases automated tools are not looking for vulnerabilities existing in web applications despite they have the ability to detect it, and in other cases, conduct tests to detect vulnerabilities that actually exist in the web applications analyzed but ultimately do not report their existence.

Keywords

Dynamic analysis, IDS, Security, Vulnerabilities, Web application.

Resumen

Es habitual el uso de escáneres de vulnerabilidades para examinar la seguridad de aplicaciones web y en la literatura existen muchos trabajos que comparan las capacidades de detección de estas herramientas. Estos trabajos analizan aplicaciones web vulnerables con herramientas de análisis dinámico y luego comparan los informes resultantes. De esta comparación y posterior análisis, clasifican las herramientas de acuerdo al número de vulnerabilidades que detectan. En este trabajo, a diferencia de otras investigaciones, se pretende ir más allá y con el uso de un modelo de pruebas distinto al usado tradicionalmente, verificar las vulnerabilidades que detectan las herramientas, las pruebas que ejecutan y las vulnerabilidades que busca, de aquellas que realmente están presentes en la aplicación web analizada. A través de esto, se puede saber si las pruebas realizadas son eficientes y conocer dos aspectos relevantes sobre las herramientas de análisis dinámico de vulnerabilidades en aplicaciones Web: Qué pruebas no realizan las herramientas aunque están capacitadas para ello y qué pruebas realizan pero no reportan vulnerabilidades que realmente tienen las aplicaciones. Se modificó el modelo de pruebas utilizado en muchos trabajos anteriores colocando un IDS en medio de las herramientas de detección y las aplicaciones web vulnerables. De toda esta información recogida se comprobó que algunas de las herramientas no buscan vulnerabilidades conocidas y documentadas que se encuentran en las aplicaciones web analizadas, a pesar de que la herramienta tiene la capacidad de detección, y en otros casos, se realizan pruebas para detectar vulnerabilidades que realmente existen en las aplicaciones web analizadas pero que finalmente no se informa de su existencia.

Palabras clave

Análisis dinámico, Aplicaciones Web, IDS, Seguridad informática, Vulnerabilidades.

Agradecimientos

A Dios y mi familia, en especial mi Madre y Hermanos que siempre han estado a mi lado brindándome el apoyo para salir adelante en todos los proyectos que he emprendido en mi vida, este logro es para ellos. A mi Padre, Carlos Armas, que desde el cielo nos sigue bendiciendo y protegiendo a mí y mis hermanos.

A mis amigos y directores de proyecto Ana Lucila y Luis Javier, que confiaron en mí y han sabido brindarme su ayuda y guía durante todas las etapas en el desarrollo de este trabajo y que gracias a su apoyo y esfuerzo se ha culminado de manera exitosa, a ellos el más grande y profundo de mis agradecimientos. A Fernando Román, que supo compartir su conocimiento e ideas para sacar adelante este proyecto ya que, sin ayuda y visión, este trabajo no hubiese sido posible.

Finalmente, agradezco al Instituto de Fomento al Talento Humano del Gobierno del Ecuador y su Programa de Becas, por haber financiado mis estudios de Máster a través de la beca otorgada en la Convocatoria Abierta 2014 Primera Fase.

Lista de acrónimos

API	Application Programming Interface
CMDEXec	Remote Command Execution
CSRF	Cross-Site Request Forgery
DoS	Denial of Service
DVWA	Damn Vulnerable Web Application
HTML	HyperText Markup Language
IDS	Intrusion Detection System
IPv6	Internet Protocol version 6
JSON	JavaScript Object Notation
LFI	Local File Inclusion
NIST	National Institute of Standards and Technology
NTLM	NT LAN Manager
NVD	National Vulnerability Database
OWASP	Open Web Application Security Project
PCI DSS	Payment Card Industry Data Security Standard
PENTEST	Penetration Testing
PXSS	Persistent Cross-site scripting
REST	Representational State Transfer
RFI	Remote File Inclusion
RPC	Remote Procedure Call
RXSS	Reflected Cross-site scripting
SAMM	Software Assurance Maturity Model
SDK	Software Development Kit

SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSL	Secure Sockets Layer
TLS	Transport Layer Security
URL	Uniform Resource Locator
WASC	Web Application Security Consortium
WASP	Web Application Security testing Platform
WASSEC	Web Application Security Scanner Evaluation Criteria
WIVET	Web Input Vector Extractor Teaser
WSDL	Web Services Description Language
WSDL	Web Services Description Language
WVS	Web Vulnerability Scanner
XML	eXtensible Markup Language
XSS	Cross-site scripting
ZAP	Zed Attack Proxy

ÍNDICE

1. INTRODUCCIÓN 1

1.1. MOTIVACIÓN 2
1.2. OBJETO DE LA INVESTIGACIÓN 3
1.3. ESTRUCTURA DEL TRABAJO 4

2. APLICACIONES 5

2.1. APLICACIÓN 5
2.1.1. Implicación de la Seguridad en el Ciclo de Vida de un Software 5
2.2. APLICACIONES WEB 7
2.2.1. Patrón de Diseño Modelo Vista Controlador para Aplicaciones Web 7
2.3. VULNERABILIDADES EN APLICACIONES WEB 9
2.3.1. Amenazas, Riesgos, Vulnerabilidades 9
2.3.2. Clasificación de Vulnerabilidades 9
2.3.2.1. Clasificación de WASC 9
2.3.2.2. Clasificación de OWASP 10

3. ANÁLISIS DE VULNERABILIDADES 15

3.1. ANÁLISIS ESTÁTICO 15
3.1.1. Técnicas de Análisis Estático 15
3.2. ANÁLISIS DINÁMICO 17
3.2.1. Herramientas de Análisis Dinámico de Vulnerabilidades en Aplicaciones Web 18
3.2.1.1. Etapa Pasiva 18
3.2.1.2. Etapa Activa 20
3.2.2. Herramientas de Análisis Dinámico Usadas 22
3.2.2.1. OWASP ZAP 23
3.2.2.2. Arachni 27
3.2.2.3. Acunetix WVS 29
3.2.2.4. HP WebInspect 33

4. TRABAJOS RELACIONADOS 37

4.1. EVALUACIÓN DE ESCÁNERES 37
4.2. USO DE SISTEMAS DE DETECCIÓN DE INTRUSOS EN EL ANÁLISIS DE VULNERABILIDADES WEB 43

5. DISEÑO DE UN MODELO DE PRUEBAS PARA HERRAMIENTAS DE ANÁLISIS DINÁMICO DE APLICACIONES WEB 45

5.1. MODELO DE PRUEBAS TRADICIONAL 45
5.2. MODELO DE PRUEBAS PROPUESTO 46
5.2.1. Componentes de Modelo Propuesto 50
5.2.1.1. Nodo IDS 50
5.2.1.2. Nodo Servidor Web 51
5.2.1.3. Nodo Atacante 51
5.3. CONFIGURACIÓN Y COMPONENTES DE EXPERIMENTO CON MODELO PROPUESTO 52
5.3.1. Componentes 52
5.3.2. Herramientas de Análisis Dinámico de Vulnerabilidades 53
5.3.3. Aplicaciones Vulnerables 54
5.3.4. Sistema de Detección de Intrusos 56
5.3.5. Software y Hardware Adicional 56
5.3.5.1. Nodo Atacante 57
5.3.5.2. Nodo Servidor Web 57
5.3.5.3. Nodo IDS 58

6. ANÁLISIS RESULTADOS 59

6.1. TIEMPO Y RECURSOS DE RED 59
6.2. PETICIONES GENERADAS 61
6.3. RESULTADOS CON DVWA 61

6.3.1. Falsos Positivos 63
6.4. RESULTADOS CON WACKOPICKO 64
6.4.1.1. Falsos Positivos 65
7. CONCLUSIONES Y TRABAJO FUTURO 69
7.1. CONCLUSIONES 69
7.2. TRABAJOS FUTUROS 70
8. INTRODUCTION 73
8.1. MOTIVATION 74
8.2. PURPOSE OF INVESTIGATION 75
8.3. STRUCTURE OF THIS PAPER 75
9. CONCLUSIONS AND FUTURE WORKS 77
9.1. CONCLUSIONS 77
9.2. FUTURE WORKS 78
REFERENCIAS 79
ANEXO A: EJEMPLO DE VULNERABILIDADES DE OWASP TOP 10 2013 83
ANEXO B: SNORT 89

ÍNDICE DE TABLAS

Tabla 5.1 Vulnerabilidades presentes en DVWA 54
Tabla 5.2 Vulnerabilidades presentes en WackoPicko 55
Tabla 5.3 Características físicas del entorno de pruebas 58
Tabla 6.1 Falsos positivos del análisis de DVWA 63
Tabla 6.2 Falsos positivos del análisis de WackoPicko..... 66
Tabla 6.3 Resultados de analizar la aplicación web DVWA 67
Tabla 6.4 Resultados de analizar aplicación web WackoPicko 68

ÍNDICE DE FIGURAS

Fig. 2.1 Actividades imprescindibles para el ciclo de vida de un software 5
Fig. 2.2 Buenas prácticas de seguridad durante el ciclo de vida de un software 6
Fig. 2.3 Patrón de diseño Modelo Vista Controlador MVC 8
Fig. 3.1 Técnicas de Análisis Estático. 16
Fig. 3.2 Herramientas de Análisis Dinámico (en inglés *Black-box Testing Tools*) 18
Fig. 3.3 Etapa de rastreo del sitio DVWA con Acunetix WVS 20
Fig. 3.4 Etapa activa del análisis de DVWA con Acunetix WVS 21
Fig. 3.5 Análisis de DVWA con Acunetix WVS - Análisis de las respuestas de DVWA 21
Fig. 3.6 Captura de pantalla de la interfaz de usuario de la herramienta OWASP ZAP 24
Fig. 3.7 Captura de Pantalla de la interfaz gráfica de Arachni 27
Fig. 3.8 Captura de Pantalla de línea de comandos de Arachni 28
Fig. 3.9 Captura de Pantalla de la interfaz gráfica de Acunetix WVS 30
Fig. 5.1 Componentes de Modelo Tradicional 46
Fig. 5.2 Diagrama de Modelo Tradicional 46
Fig. 5.3 Componentes de Modelo Propuesto 47
Fig. 5.4 Flujo de Trabajo en Modelo Propuesto 49
Fig. 5.5 Diagrama de Componentes Físicos de Prueba 52
Fig. 6.1 Uso de recursos de red en el análisis de cada herramienta 60
Fig. 6.2 Tiempo de ejecución en el análisis por cada herramienta 60
Fig. 6.3 Sesiones generadas por cada herramienta en el análisis 61
Fig. 0.1 Esquema de Arquitectura de Snort 89

1. INTRODUCCIÓN

El software lo usamos en la gran mayoría de las actividades diarias, incluso sin enterarnos de que lo estamos haciendo. Las aplicaciones web están presentes también en estas actividades, como por ejemplo la gestión de la cuenta bancaria a través de la aplicación web del banco o la gestión de los recibos de los servicios básico o incluso en la forma con la que se interactúa con las personas, a través de redes sociales. Estas aplicaciones en general, almacenan y gestionan muchos datos, datos personales incluso datos sensibles y confidenciales, por lo que estas aplicaciones deben ser protegidas en orden de proteger también estos datos. En el año 2015 se registró un total de 5334 ataques dirigidos hacia aplicaciones web, de estos casos, un total de 908 incidentes confirmaron la exposición de los datos gestionados por la aplicación victima [1]. Las buenas prácticas de seguridad y el uso de herramientas especializadas para fortalecer la aplicación a lo largo de su desarrollo y puesta en producción, reduce significativamente la probabilidad del éxito de un ataque [2][3].

En el mercado existen diversas herramientas que agilizan el análisis y posterior corrección de fallos y vulnerabilidades de una aplicación. Elegir la herramienta adecuada es una tarea importante, pues de su efectividad depende la posibilidad de subsanar cualquier vulnerabilidad de una aplicación. Estas herramientas, de acuerdo a sus capacidades y características, pueden ser de caja blanca ("*White-box Testing Tools*") o de caja negra ("*Black-box Testing Tools*").

El análisis de caja blanca [3][4] consiste en analizar el código fuente y la estructura de la aplicación en búsqueda de fallos que puedan abrir brechas de seguridad en la aplicación al momento de ejecutarse, convirtiéndola en una aplicación vulnerable. El análisis del código se puede hacer mediante el uso de herramientas específicas o también de forma manual [5]. La principal desventaja que existe sobre este tipo de pruebas es el tiempo que lleva realizar el análisis del código de la aplicación.

En el análisis de caja negra [3][4][6], se analiza la seguridad de una aplicación de forma funcional, sin analizar el código fuente, mediante el uso de herramientas automáticas encargadas de buscar las posibles vulnerabilidades en la aplicación realizando ataques de penetración.

El uso de herramientas de penetración y análisis dinámico de aplicaciones permite reducir tiempo y esfuerzo dentro del ciclo de desarrollo de una aplicación y además permite enfocar mayores esfuerzos en tareas de seguridad más complejas [7].

Este tipo de herramientas no son sencillas de configurar para quien no está familiarizado con ellas [5]. Una de sus mayores debilidades es la presencia de falsos positivos en sus resultados [7], por lo que es importante contar con el conocimiento de las capacidades y debilidades que estas herramientas presentan.

1.1. Motivación

Actualmente, en el mercado se puede hallar distintos escáneres de vulnerabilidades para aplicaciones web. Sin embargo, ninguna de estas herramientas son cien por ciento efectiva. Muchas de estas herramientas tienen un alto porcentaje de falsos positivos en sus resultados.

Los trabajos existentes en la literatura se han enfocado en analizar principalmente la precisión de los escáneres utilizando un mismo método para su evaluación. Ninguno ha ido más allá, para observar el comportamiento que tienen las herramientas de análisis de vulnerabilidades con las aplicaciones que analizan y buscar ahí los fallos que resultan en la imprecisión de sus reportes. Es decir, que no se analiza el qué hace la herramienta para llegar al informe que presenta, sino que simplemente se comparan los resultados obtenidos en los reportes de cada una y se extrae un criterio evaluativo únicamente en base a estos.

En este trabajo se considera importante conocer detalles del comportamiento del escáner durante el análisis de una aplicación web vulnerable, las pruebas realizadas, los fallos y aciertos del escáner, entre otros aspectos, para realizar una evaluación completa de las capacidades e insolvencias de este tipo de herramientas.

1.2. Objeto de la investigación

En este trabajo se realiza un enfoque distinto al utilizado en la literatura para obtener un panorama más claro respecto al funcionamiento y precisión de las herramientas. Se introduce, en la estructura típica de la literatura, un Sistema de Detección de Intrusos (IDS), que permite obtener las solicitudes de ataque que cada herramienta realiza en su análisis y así contrastarlo con el reporte correspondiente. De esta manera, se adquiere información que puede determinar la eficiencia de las herramientas al momento de realizar un análisis.

Los objetivos específicos del trabajo son:

- Proponer un modelo para pruebas que permita adquirir más información sobre los resultados del análisis de vulnerabilidades realizados en aplicaciones web vulnerables, con herramientas de análisis dinámico.

- Realizar un análisis automático del tráfico generado por el escáner de vulnerabilidades web para acelerar el proceso de evaluación y disminuir el riesgo de fallo, detectado en trabajos previos, por realizar un proceso de análisis manual.

- Hallar las posibles pruebas que realiza y aquellas que omite un escáner de vulnerabilidades web durante su ejecución.

1.3. Estructura del trabajo

El resto del trabajo está compuesto de 6 capítulos más, con la estructura que se comenta a continuación:

El Capítulo 2 presenta, conceptos básicos de manera general y la importancia de la seguridad en el desarrollo de aplicaciones. Asimismo, se detalla la clasificación de las vulnerabilidades más importantes.

El Capítulo 3 expone las dos categorías principales en las que se clasifican las herramientas de análisis de vulnerabilidades web y detalla las fases de ejecución de las herramientas dinámicas para llevar a cabo su análisis. Se describe también las características principales de las herramientas utilizadas durante los experimentos realizados.

En el Capítulo 4 se realiza una revisión de los principales trabajos relacionados con el análisis de escáneres de vulnerabilidades en aplicaciones web.

El Capítulo 5 detalla la metodología propuesta en este trabajo, describiendo cada uno de los componentes utilizados en los experimentos y su configuración.

El Capítulo 6 describe los resultados obtenidos tras los experimentos para comprobar el funcionamiento de la metodología propuesta, la configuración de las herramientas y las aplicaciones web.

Por último, en el Capítulo 7 se explican las conclusiones del presente trabajo y se mencionan las posibles líneas de trabajo futuro que conduzcan a enriquecer la investigación realizada.

2. APLICACIONES

Para comprender la importancia de la seguridad en el desarrollo de aplicaciones es necesario tener claro algunos conceptos fundamentales.

2.1. Aplicación

Una aplicación es un programa de software que está diseñado para realizar tareas específicas [8]. Hoy en día, hay aplicaciones presentes en distintos procesos de tipo productivo, educativo, entretenimiento y de seguridad.

2.1.1. Implicación de la Seguridad en el Ciclo de Vida de un Software

El ciclo de vida de un software son actividades interrelacionadas que conducen a la elaboración de un producto de software [2]. Existe una amplia variedad de procesos de software para el desarrollo de una aplicación, o de un componente de software, pero todos deben incluir al menos las actividades que se muestran en la Figura 2.1.

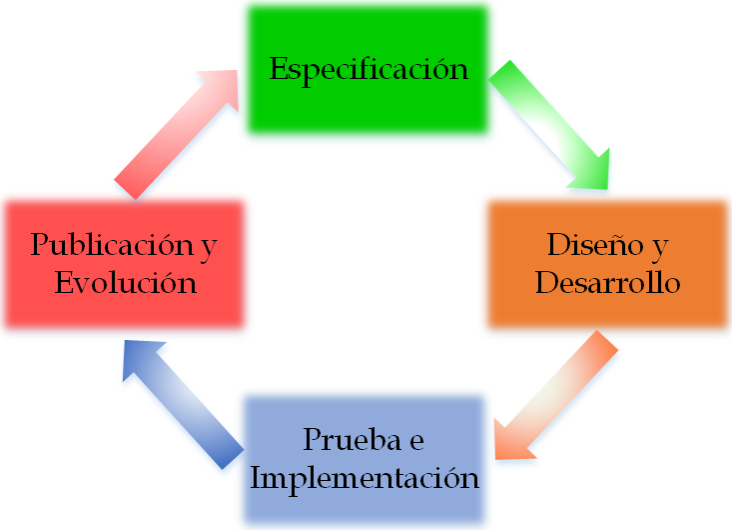

Fig. 2.1 Actividades imprescindibles para el ciclo de vida de un software

Todas las etapas del ciclo de vida de un software se retroalimentan entre sí. Esta información les permite identificar a tiempo posibles problemas y minimizar el coste en el mantenimiento del software [2]. En la Figura 2.2 se presentan las principales recomendaciones de buenas prácticas de seguridad que se advierte aplicar en cada una de las etapas del ciclo de vida de un software.

Fig. 2.2 Buenas prácticas de seguridad durante el ciclo de vida de un software

A continuación se detallan las recomendaciones presentadas en la Figura 2.2.

Especificación: En esta etapa se determinan los requerimientos, funcionalidades, y restricciones del software a desarrollar. Es fundamental la identificación de las características de seguridad que deben cumplir los requerimientos del software, para su adecuado funcionamiento, como por ejemplo: arquitectura, datos a los que tendrá acceso, perfiles de usuario y modos de autenticación.

Diseño y Desarrollo: Se identifican y describen las abstracciones fundamentales del software y las relaciones entre sí. La gran parte de los problemas de seguridad se generan en esta etapa [9]. Es importante considerar requisitos de privacidad, diseñar medidas de seguridad, diseño de casos de abuso, seguir normas de codificación segura, utilizar librerías y software de terceros que sean

seguros, utilizar características de seguridad de los entornos de desarrollo de los lenguajes empleados en la codificación[10].

Prueba e implementación: Se verifica que el funcionamiento del software esté acorde a lo establecido en las etapas de especificación y de diseño. Durante esta fase se pueden utilizar distintos métodos que permitan verificar la seguridad con la que el software se ha desarrollado. El uso de herramientas de análisis dinámico o pruebas de penetración, así como herramientas de análisis estático que verifican la seguridad con la que se ha codificado el software y el control de la información que ingresa y sale de la aplicación.

Publicación y Evolución: El software terminado es puesto en funcionamiento. Esté debe ser adaptable y escalable en el tiempo. De la misma forma que en la etapa anterior, se puede hacer uso de escáneres de vulnerabilidades para identificar fallos de seguridad dentro de un entorno de producción.

2.2. Aplicaciones web

WASC define una aplicación web como "una aplicación de software, ejecutada por un servidor web, la cual responde a solicitudes de páginas web dinámicas sobre HTTP" [11]. Una aplicación web se ejecuta en el navegador del cliente, por lo tanto no requiere ser instalada en el ordenador local para ejecutarse. Una aplicación web consta de una arquitectura multinivel y para facilitar su desarrollo es común el uso de patrones de diseño. Un patrón de diseño muy popular para el desarrollo de aplicaciones web es el Modelo Vista Controlador o (en inglés *Model View Controller) MVC.*

2.2.1. Patrón de Diseño Modelo Vista Controlador para Aplicaciones Web

Este patrón de diseño permite dividir una aplicación, o módulo de ella, en tres partes: Modelo, Vista y Controlador [12]. La Figura 2.3 muestra cómo interactúan estos componentes lógicos [2]:

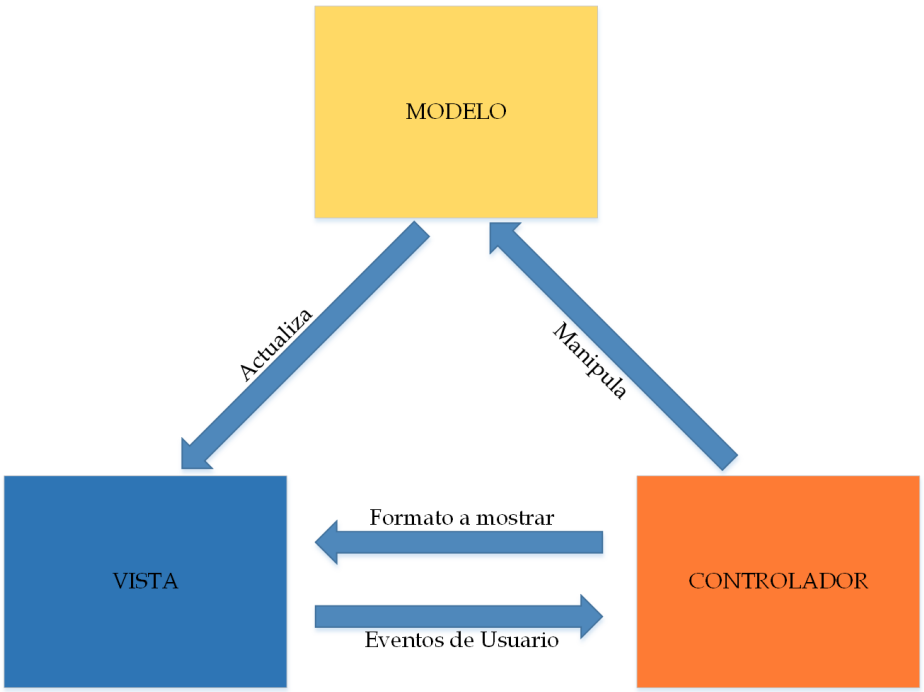

Fig. 2.3 Patrón de diseño Modelo Vista Controlador MVC

- **Modelo**: Gestiona los datos del sistema y las operaciones asociadas al sistema. Se puede considerar como el componente de persistencia en la aplicación.

- **Vista**: Define y gestiona como se muestran los datos de cara al usuario. Se puede considerar como la interfaz gráfica de la aplicación.

- **Controlador**: Gestiona la interacción del usuario con la aplicación, recibe los eventos generados por el usuario y responde a cada uno de ellos. Se lo puede considerar como la lógica de la aplicación.

En el desarrollo de aplicaciones web este patrón de diseño es ampliamente utilizado. Muchas infraestructuras digitales o (en inglés *frameworks*) para el desarrollo de aplicaciones web como AngulaJS o Backbone implementan un patrón MVC [13].

2.3. Vulnerabilidades en Aplicaciones Web

A efectos de comprender mejor esta sección, es importante refrescar conceptos y términos que son usados en este trabajo.

2.3.1. Amenazas, Riesgos, Vulnerabilidades

Estos términos envuelven conceptos distintos pero a pesar de ello están relacionados entre sí [14][15][16].

Las amenazas son los eventos que de suceder, tendrían consecuencias negativas sobre un activo, en este contexto, un activo se considera una aplicación web. Las amenazas se sirven de las debilidades o vulnerabilidades, que poseen las aplicaciones web para afectarla de manera negativa [17].

El riesgo es un indicador de lo que probablemente suceda por causa de las amenazas. Es la combinación de lo probable de un acontecimiento y sus respectivas consecuencias [16].

La existencia de vulnerabilidades implica cierto riesgo para la aplicación. El riesgo se puede medir según el impacto negativo que le pueda causar a la aplicación y la probabilidad o frecuencia de que ocurra [17].

2.3.2. Clasificación de Vulnerabilidades

Varias organizaciones de tipo gubernamentales, comerciales o sin fines de lucro han elaborado distintos tipos de clasificaciones para los diferentes tipos de vulnerabilidades que se pueden hallar en las aplicaciones web. Dentro del marco de desarrollo de este trabajo se consideró las clasificaciones más actualizadas al momento de realizar esta memoria y los respectivos experimentos. Las organizaciones tomadas en cuenta fueron: WASC y OWASP.

2.3.2.1. Clasificación de WASC

La WASC (en inglés *The Web Application Security Consortium* WASC) es una

organización sin fines de lucro, compuesta de expertos, profesionales de la industria y representantes de organizaciones relacionadas con la seguridad del software. Producen estándares ampliamente aceptados y de código abierto sobre seguridad para la Web mundial. [11].

WASC está compuesta por una comunidad altamente activa la cual desarrolla la documentación técnica que organizaciones gubernamentales, comerciales, educativas y de toda índole utiliza como material de apoyo en la mejora de la seguridad de sus aplicaciones web.

La clasificación de amenazas de WASC es un documento desarrollado por la comunidad WASC para organizar y explicar las amenazas presentes en las aplicaciones web. Esta clasificación fue creada con el objetivo de promover un estándar de la terminología a usarse en la industria para describir este tipo de problemas [18]. Esta clasificación contiene 49 vulnerabilidades de aplicaciones web establecidas de acuerdo a su criterio y organizadas de manera alfabética, cada una de estas vulnerabilidades cuenta con la explicación detallada y con ejemplos que ayudan a su comprensión. La versión más actual de esta clasificación es la v2.0 y fue presentada en 2010 [19][18].

2.3.2.2. Clasificación de OWASP

La fundación Open Web Aplicación Security Project OWASP es una organización sin fines de lucro establecida en 2004. OWASP es una comunidad abierta dedicada a la concepción, desarrollo, adquisición, operación y mantenimiento de aplicaciones confiables para la seguridad de sistemas web y para el uso de cualquier organización que lo requiera. Toda la documentación y herramientas están disponibles y pueden ser utilizadas por cualquiera que lo necesite. Su propósito principal según ellos declaran en la página web de su organización es "Ser la comunidad global más prospera que impulse la visión y la evolución de la seguridad y protección en el software mundial" [20].

OWASP realiza periódicamente la clasificación de los fallos de seguridad

más críticos para las aplicaciones web, este documento es conocido como El OWASP Top 10 (en inglés *The OWASP Top 10*) y consiste en una lista de los 10 riesgos de seguridad más críticos para las aplicaciones web según su criterio. Los datos con los que realiza esta clasificación los obtiene de empresas consultoras y de fabricantes de herramientas especializadas en la detección de este tipo de vulnerabilidades [17], actualmente se encuentra en desarrollo la versión 2016 [20] y para este trabajo se ha considerado la información de la versión más actual de este listado, que es del 2013.

El OWASP Top 10 de 2013 está basado en la información obtenida de 7 firmas especializadas en la seguridad de aplicaciones web. La clasificación de las diez categorías de vulnerabilidades se han seleccionado de acuerdo al porcentaje de posibilidad de uso en un ataque y a la estimación del impacto en la aplicación atacada [21].

A continuación se detallan cada una de las vulnerabilidades presentes en la clasificación OWASP Top 10 de 2013.

- **Inyección:** Esta vulnerabilidad puede ser de SQL, OS y LDAP. Este tipo de vulnerabilidades permiten al atacante enviar código malicioso a través de una aplicación hacia otro sistema, logrando que el atacante ejecute comandos mal intencionados en el sistema vulnerable o acceder a datos sin autorización [21].

- **Fallo en autenticación y gestión de sesiones:** Es muy común que las funcionalidades relacionadas con la autenticación y gestión de sesiones de usuario en una aplicación no sean correctamente implementadas. Esta vulnerabilidad explota esos fallos y permite que un atacante pueda suplantar la identidad de cualquier usuario del sistema [21].

- **Secuencia de Comandos en Sitios Cruzados XSS:** Esta vulnerabilidad ocurre cuando datos no confiables se envían a través de una aplicación web sin la adecuada validación y permiten que el atacante ejecute scripts

maliciosos en el navegador de la víctima pudiendo incluso obtener sus credenciales de sesión, alterar la interfaz de la aplicación web o redirigir a la víctima hacia otros sitios maliciosos [21].

- **Referencia Directa de Objetos Insegura:** Esta vulnerabilidad está presente cuando el desarrollador de la aplicación expone en su código referencias hacia uno o varios objetos internos del sistema (fichero, directorio o claves de base de datos) y que el atacante pueda manipular estas referencias para acceder a datos sin autorización a través de ellas [21].

- **Configuraciones de Seguridad mal implementadas:** La seguridad de una aplicación depende también de la correcta configuración de seguridad de cada uno de los componentes del entorno en donde se ejecute y de mantener todo el software actualizado al día para evitar vulnerabilidades derivadas de ello y que se puedan utilizar por parte de un atacante [21].

- **Exposición de datos sensibles:** La correcta implementación de mecanismos seguros de gestión de los datos que administra una aplicación es fundamental y más aún si la aplicación trabaja con datos sensibles como números de tarjetas de crédito, documentos de identidad o credenciales de autenticación del sistema. Un atacante puede robar o modificar estos datos si es que no se encuentran adecuadamente protegidos. Los datos sensibles deben tener una protección adicional, como el cifrado [21].

- **Ausencia de control de acceso a funcionalidades:** Todas las aplicaciones web deberían implementar la verificación de nivel de acceso a sus funciones antes de mostrarlas en la interfaz del usuario y lo mismo del lado del servidor al cual cada funcionalidad accede. Si es que no existe este tipo de control, el sistema es vulnerable a que un atacante pueda

acceder a funcionalidades para las cuales no está autorizado y ejecutarlas [21].

- **Falsificación de Petición en Sitios Cruzados CSRF:** Este tipo de vulnerabilidad obliga a que el navegador de la víctima, envié peticiones HTTP en la cual están incluidos *cookie's* de sesión y cualquier otra información de autenticación hacia una aplicación web vulnerable y que esta a su vez entienda cada petición como legitima de la víctima [21].

- **Uso de componentes vulnerables:** Componentes como infraestructuras digitales, librerías y otro tipo de módulos de terceros suelen ejecutarse con alto nivel de privilegios y si es que cualquiera de estos contiene una vulnerabilidad y que no ha sido corregida, puede ser utilizada por un atacante y a través de ella acceder a datos sensibles, destruir la aplicación o incluso el servidor en el cual se ejecuta. El uso de componentes con vulnerabilidades ya conocidas debilitan la seguridad implementada en la aplicación y aumentan la posibilidad de sufrir ataques por parte de un usuario malicioso [21].

- **Redirección y reenvío sin validación:** Las aplicaciones web frecuentemente reenvían y redirigen sus usuarios hacia páginas y sitios web utilizando parámetros que determinan a donde deben ir, si la aplicación no cuenta con una adecuada validación de estos parámetros, un atacante puede utilizar esta vulnerabilidad de la aplicación web para redirigir al usuario a sitios maliciosos o acceder a datos para los cuales no esté autorizado [21].

En el Anexo A se detallan con un breve ejemplo cada una de las vulnerabilidades presentes en la clasificación OWASP Top 10 2013.

3. ANÁLISIS DE VULNERABILIDADES

En la detección de vulnerabilidades o fallos en aplicaciones web se puede realizar dos tipos de análisis: Dinámico y Estático.

El análisis de vulnerabilidades considerado en este trabajo y con el cual se realizaron los experimentos de los capítulos siguientes, son las herramientas que realizan pruebas de análisis dinámico de vulnerabilidades en aplicaciones web.

3.1. Análisis Estático

El análisis estático de una aplicación es la revisión de los posibles fallos existentes a un nivel interno, utilizando el código fuente de la aplicación web para buscar vulnerabilidades que puedan permitir un fallo en la seguridad de la aplicación al momento de su puesta en funcionamiento [22]. Este tipo de análisis permite la detección y corrección de vulnerabilidades presentes en componentes del código fuente de la aplicación evaluada [23]. La tarea de analizar el código en búsqueda de fallos se puede hacer de forma manual o con la ayuda de herramientas de software que lo faciliten, reduciendo el tiempo de análisis.

Las herramientas que permiten este tipo de análisis son conocidas como herramientas de prueba de caja blanca (en inglés *White-box Testing tolos*).

3.1.1. Técnicas de Análisis Estático

El análisis estático de una aplicación puede ser ejecutado de distintas formas, todas ellas con el objetivo de verificar la integridad y funcionalidad de determinadas partes y características propias de la aplicación evaluada. La Figura 3.1 muestra las técnicas más importantes del análisis estático.

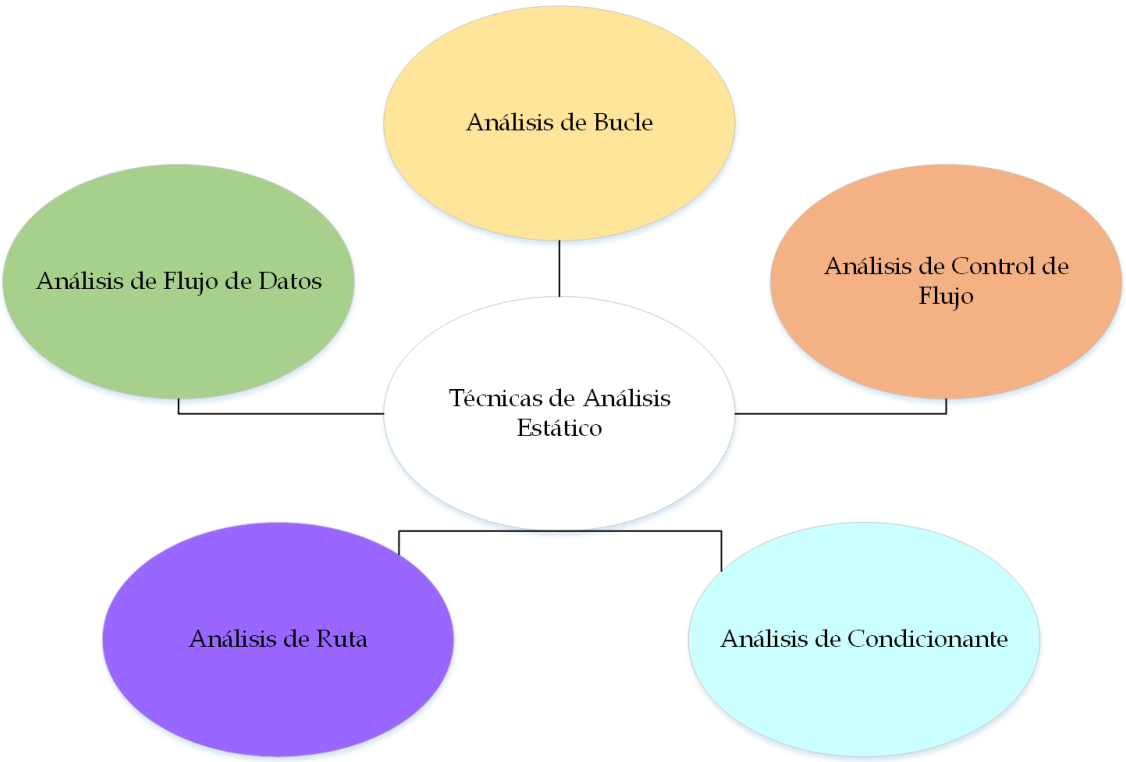

Fig. 3.1 Técnicas de Análisis Estático.

Cada una de estas técnicas se describen brevemente:

- **Análisis de Flujo de Datos**: En esta técnica se comprueba el comportamiento de los datos de prueba introducidos para el análisis y se verifica su paso a través de las funcionalidades de la aplicación. Esta técnica puede descubrir fallos como, declaración de variables que no son utilizadas en ningún momento o el uso de variables que no han sido declaradas previamente [24].

- **Análisis de Control de Flujo**: Esta técnica se utiliza para verificar la estructura de la aplicación y desarrollar casos de prueba en base a esta estructura. Estos casos de prueba son desarrollados para cubrir toda la estructura de control de la aplicación analizada [24].

- **Análisis de Ruta**: Es una técnica que verifica todos los posibles "caminos" que puede tomar la aplicación en su ejecución con un determinado grupo de datos. Todos los caminos que puede tomar la

aplicación deben verificarse al menos una vez. Es una técnica muy efectiva, incluso más que la técnica de análisis de condicionante. Se usa mucho para verificar aplicaciones altamente complejas [24].

- **Análisis de Condicionante**: En esta técnica se verifica la implementación de las condicionantes dentro de la aplicación y los datos de prueba que se utilizan en el análisis deben poder verificar cada posibilidad al menos una vez [25].

- **Análisis de Bucle**: En esta técnica, el análisis se enfoca en la correcta implementación de los bucles dentro de la aplicación [24]. Su verificación es sencilla. Existen distintos tipos de bucles: Simple, Anidado y Concatenado

3.2. Análisis Dinámico

El análisis dinámico de vulnerabilidades es un tipo de prueba que verifica el comportamiento de una aplicación ante distintos parámetros y condiciones [26]. Este tipo de prueba es realizado por software especializado conocido como herramientas de prueba de penetración (en inglés *PenTest Tools*) o herramientas de prueba de caja negra (en inglés *Black-box Testing Tools*). En la Figura 3.2 se puede observar la metodología de ejecución que tiene una herramienta de análisis dinámico de vulnerabilidades en aplicaciones web.

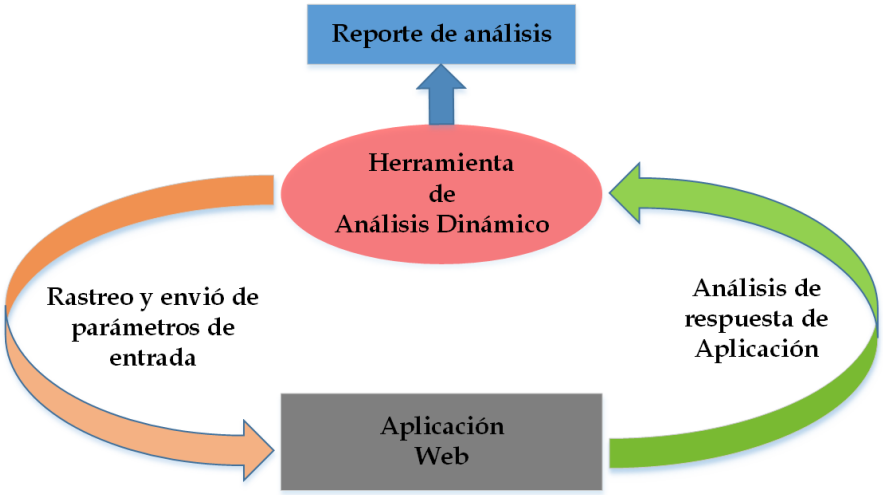

Fig. 3.2 Herramientas de Análisis Dinámico (en inglés *Black-box Testing Tools*)

En este tipo de pruebas no interesa conocer el mecanismo de funcionamiento interno de la aplicación web que analiza. El análisis se realiza sin conocer de antemano la arquitectura, el lenguaje de desarrollo o la plataforma sobre la que se ejecuta la aplicación web. Para el análisis dinámico lo único que interesa es analizar las respuestas de la aplicación web a las solicitudes maliciosas enviadas por el escáner de vulnerabilidades web [25].

3.2.1. Herramientas de Análisis Dinámico de Vulnerabilidades en Aplicaciones Web

Una herramienta de análisis dinámico de vulnerabilidades web o también conocido como escáner de vulnerabilidades web, es un software que analiza de manera funcional una aplicación web en busca de vulnerabilidades que puedan ser explotadas y provocar un comportamiento inadecuado por parte de la aplicación web.

Los escáneres de vulnerabilidades web, realizan el análisis de una aplicación en dos etapas: Pasiva o (en inglés *Crawling*) y Activa.

3.2.1.1. Etapa Pasiva

La etapa pasiva del análisis tiene como tarea navegar a través de la aplicación

web, con el objetivo de localizar todas las paginas, enlaces, formularios y también identificar todos los vectores de entrada asociados a ellos [27].

Esta etapa se ejecuta en dos tareas que se describen:

- **Rastreo (en inglés *Crawling*)**: En esta etapa, la herramienta busca todos los posibles enlaces y directorios que componen la aplicación web, con el fin de adquirir su código HTML. Páginas ocultas o protegidas dentro de la aplicación web dificultan esta tarea y pueden incidir en el análisis que se realiza posteriormente, por ello es necesario que en la configuración del escáner se prevea esta posibilidad, ingresando los parámetros que la herramienta debería utilizar en el caso de no poder acceder a un enlace o formulario por falta de credenciales o parámetros específicos. Al finalizar esta etapa, el escáner debería ya contar con toda la estructura de la aplicación web en formato HTML [28]

- **Identificador de Parámetros de entrada**: En esta tarea el escáner realiza ingeniería inversa sobre el código HTML para identificar formularios y campos de entrada de datos.

En la Figura 3.3 se muestra una captura de pantalla de la herramienta Acunetix WVS durante la etapa de rastreo de la aplicación DVWA y se puede observar la creación de la estructura de ficheros que compone la aplicación web y en la parte inferior se aprecia la navegación que realiza dentro de la aplicación web

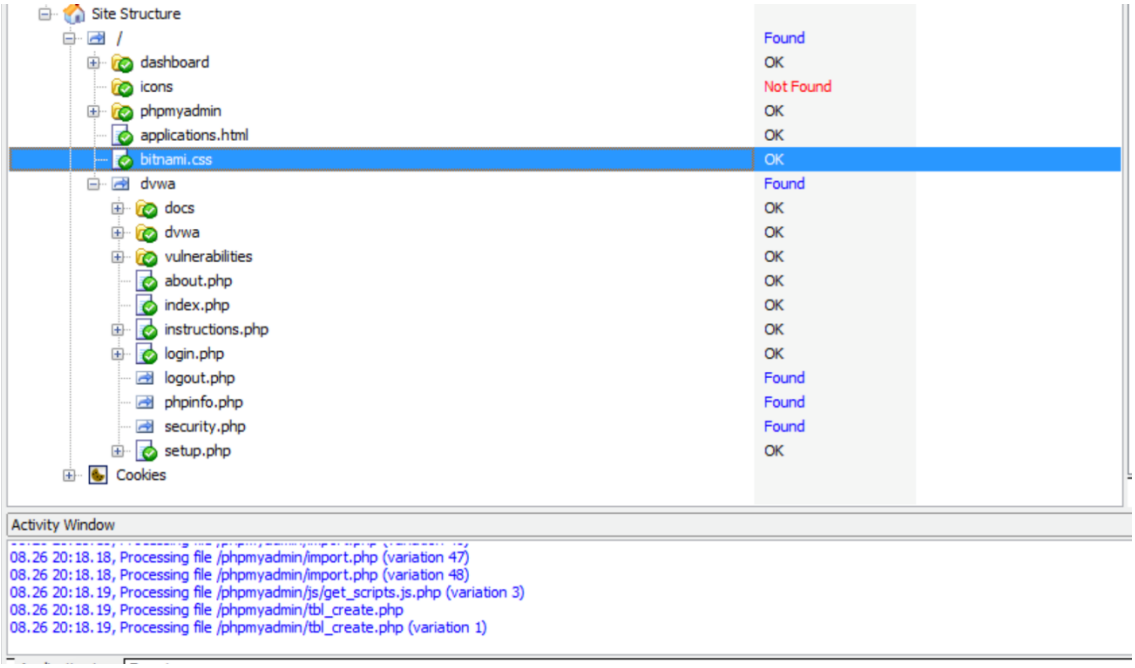

Fig. 3.3 Etapa de rastreo del sitio DVWA con Acunetix WVS

3.2.1.2. Etapa Activa

La etapa activa del análisis de un escáner de vulnerabilidades web, se ejecuta una vez que el escáner ha identificado la estructura y todos los campos de entrada de la aplicación. La herramienta inicia el ataque hacia la aplicación, el cual consiste en la inyección de valores en cada parámetro, para posteriormente analizar las respuestas que recibe por parte de la aplicación frente a lo enviando [27]. En Figura 3.4 y Figura 3.5 se puede observar el comportamiento de Acunetix WVS durante la etapa activa del análisis de la aplicación web DVWA.

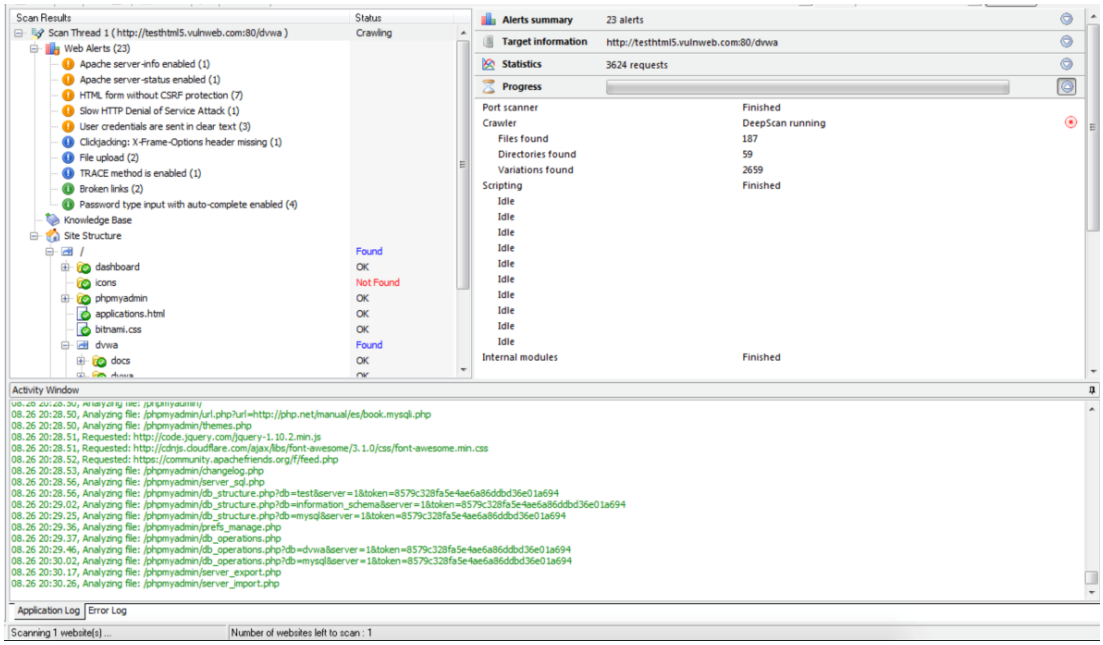

Fig. 3.4 Etapa activa del análisis de DVWA con Acunetix WVS

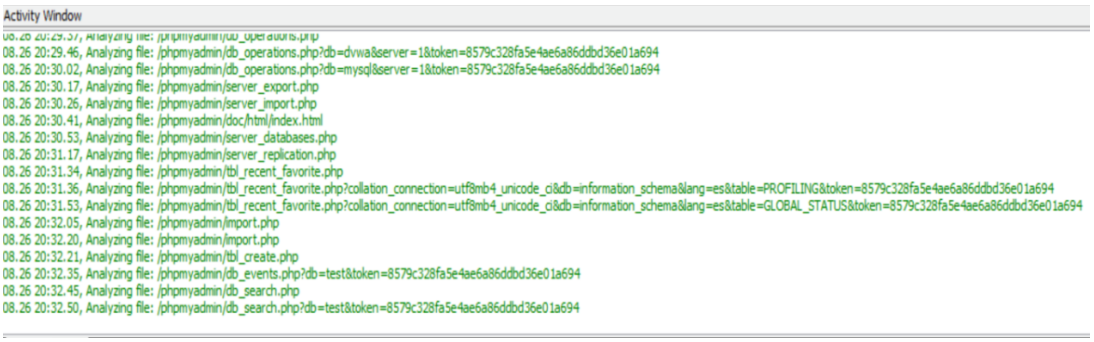

Fig. 3.5 Análisis de DVWA con Acunetix WVS - Análisis de las respuestas de DVWA

Un factor determinante para el óptimo funcionamiento de un escáner dinámico es la capacidad de realizar un reconocimiento completo de todas las paginas, enlaces y campos de entrada de la aplicación, lo más profundo posible para que después se pueda analizar todo sin dejar ninguna página, enlace o campo de lado.

La etapa activa del análisis se completa en dos tareas:

Ataque: En esta tarea el escáner genera los datos que concuerdan con lo requerido en los campos de entrada hallados en la etapa anterior, para enviarlos a la aplicación web. Por lo general estos datos son generados de forma aleatoria

por el escáner, pero también se pueden utilizar datos predefinidos por el usuario, especificados en la configuración de cada escáner antes de ejecutar el análisis. Generados estos datos, el escáner los envía y espera por la respuesta de la aplicación web para analizarla [28].

Análisis de respuestas: En esta tarea el escáner requiere ejecutar ingeniería inversa sobre cada una de las respuestas de la aplicación web, en busca de información valida, según el criterio de la herramienta, para generar el reporte final. El escáner toma la decisión de considerar si existe una vulnerabilidad o no cuando al analizar una respuesta de la aplicación web está contiene un grupo de caracteres que coincide con alguno de los mensajes de error que el escáner mantiene en su base de conocimiento. Esta etapa es la que representa una mayor cantidad de esfuerzo por parte del escáner, ya que las respuestas que son generadas por la aplicación web están elaboradas para ser consumidas por seres humanos [28].

3.2.2. Herramientas de Análisis Dinámico Usadas

Existe una gran cantidad de herramientas de análisis dinámico para búsqueda de vulnerabilidades en aplicaciones web. Estas herramientas se pueden dividir en dos grupos: Gratuitas y Comerciales.

En cada uno de estos grupos existen además herramientas de tipo:

- **Especializadas**: Estas herramientas están diseñadas para buscar determinadas vulnerabilidades. Cuentan con una base de conocimiento específico que les permite generar parámetros de entrada para ataques de vulnerabilidades determinadas y posteriormente, buscar patrones de respuesta concretos de aquellas vulnerabilidades para las cuales fueron concebidas. Por ejemplo vulnerabilidades de tipo: inyección de SQL o Secuencia de Comandos en Sitios Cruzados XSS (en inglés *Cross Site Scripting),* únicamente.

- **Genéricas**: Estas herramientas están diseñadas para detectar la mayoría de vulnerabilidades conocidas y además pueden ofrecer la capacidad de realizar búsquedas específicas. Este tipo de herramientas cuentan con características más amplias que las que presenta una herramienta especializada. Tienen una base de conocimiento mucho más grande que las herramientas especializadas, que les permite hallar una mayor cantidad de vulnerabilidades de distintas clases.

En este trabajo se emplearon cuatro herramientas de análisis dinámico de tipo genéricas: dos escáneres comerciales y dos escáneres gratuitos. Todas las herramientas fueron seleccionadas por los resultados obtenidos en la literatura y por ser además ampliamente utilizada por empresas y profesionales dedicados a la seguridad en software.

A continuación se presentan las características generales de cada una de las herramientas utilizadas en este trabajo.

3.2.2.1. OWASP ZAP

OWASP Zed Attack Proxy (OWASP ZAP) es una herramienta gratuita de análisis dinámico de vulnerabilidades. En la Figura 3.6 se muestra la interfaz de usuario de OWASP ZAP.

ZAP Forma parte del grupo de proyectos de la fundación OWASP. Es ampliamente utilizado alrededor del mundo y la comunidad de voluntarios que la mantiene y desarrolla ofrece gran cantidad de documentación y soporte [29]. Es considerado uno de los escáneres de vulnerabilidades web gratuito más populares del mundo [29]. Su diseño y configuración permite ser usado tanto por usuarios expertos, con un alto nivel de conocimiento de este tipo de herramientas, así como por desarrolladores o usuarios en general, que no están familiarizados o son nuevos en la realización de pruebas de penetración en aplicaciones web. Puede realizar distintos tipos de análisis y ataques al configurar perfiles específicos para ajustarlos a las características de la

aplicación que se desea analizar. Es utilizado como un escáner automático, en su conjunto, pero también presenta una serie de herramientas individuales que facilitan la búsqueda de vulnerabilidades de forma manual [29].

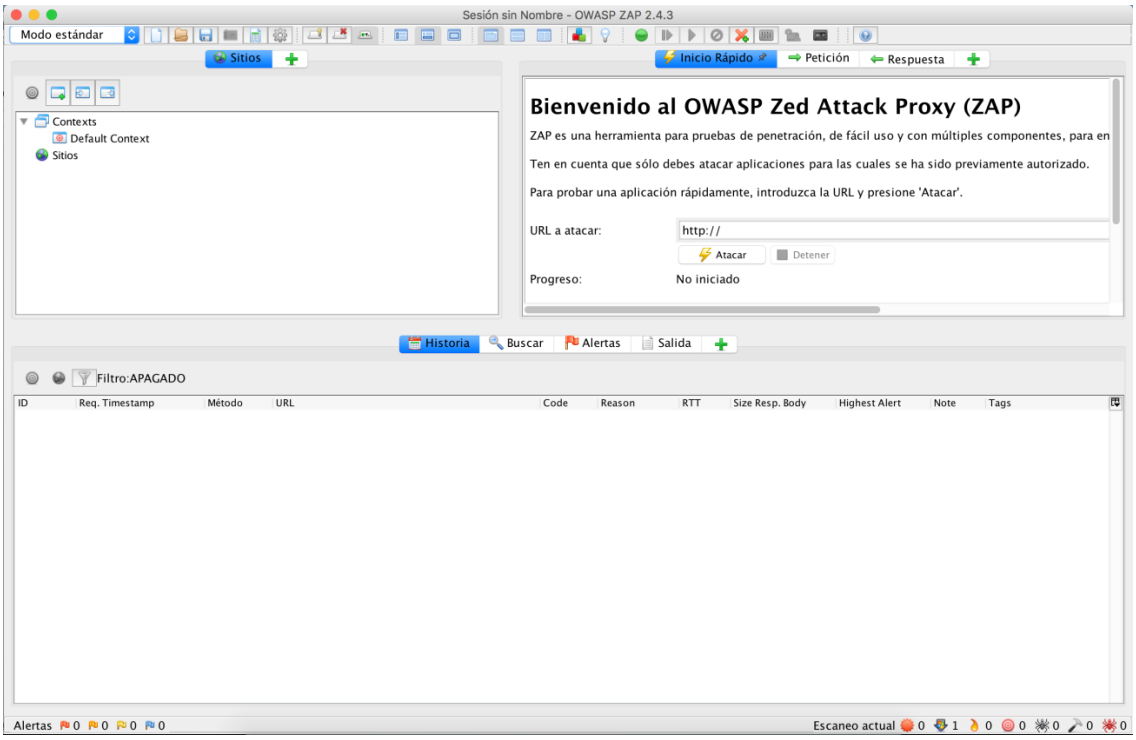

Fig. 3.6 Captura de pantalla de la interfaz de usuario de la herramienta OWASP ZAP.

Con OWASP ZAP se pueden realizar los siguientes dos tipos de análisis de forma independiente o conjunta:

- **Escaneo activo**: Es el modo más utilizado por quienes emplean esta herramienta. El escaneo activo aplica políticas de búsqueda que pueden ser configuradas por el usuario o puede utilizar sencillamente la política por defecto que hace un análisis completo de la aplicación web en busca de vulnerabilidades.

- **Escaneo Pasivo**: En este modo de análisis OWASP ZAP etiqueta de manera automática aquellas respuestas por parte de la aplicación web, en base un grupo de reglas específicas del usuario. La herramienta no realiza ningún análisis intrusivo.

Posee las siguientes características:

- **Complementos**: OWASP ZAP soporta la instalación de complementos para agregar mayor funcionalidad a la herramienta.

- **Alertas**: Diferenciación de alertas de vulnerabilidades mediante el uso de gráficos con colores que determinan los niveles de gravedad de cada vulnerabilidad hallada (Alta, Media, Baja, Información, Falso Positivo)

- **Anti CSRF *Tokens***: Permite la detección del uso de *tokens* en el envío de formularios, aumentando de esta forma la dificultad de realizar ataques CSRF. Con el uso de esta funcionalidad OWASP ZAP puede identificar estos *tokens* y almacenarlos conjuntamente con la URL que lo genero, para usarlo en ataques de este tipo.

- **API**: OWASP ZAP provee de un API a través de la cual se puede interactuar con la herramienta mediante programación. Soporta los formatos JSON, HTML y XML. Con el uso de esta API se puede acceder a funcionalidades principales como Escaneo activo y *Spider*

- **Autenticación**: Soporta distintos métodos de autenticación para el acceso hacia las aplicaciones web objeto de análisis, los principales: Método Manual, permite al usuario ingresar a la aplicación mediante la introducción de sus credenciales durante la fase de captura y navegación proxy. Basada en Formulario, este método es usado cuando la autenticación se realiza mediante el envió de un script a través de una petición GET, el script debe contener el par credencial de Usuario/Contraseña otorgado por el usuario.

- ***Break Points***: Con esta funcionalidad OWASP ZAP puede interceptar todas las peticiones y respuestas para posteriormente modificarlas. Con esta funcionalidad se puede realizar un bypass en la validación

del usuario en la aplicación y obtener sus credenciales.

- **Contexts**: Permite asociar diferentes tipos de URL's con un mismo sitio.

- ***Data Driven Content***: Con la utilización de *Contexts*, se reduce el número de peticiones en una misma página de un sitio.

- **Filtros**: Puede realizar filtros sobre cada petición y respuesta, durante el reconocimiento del sitio, utilizando la función de Proxy.

- **Exclusión de URL's(forma global)**: Con el uso de expresiones regulares ZAP ignora las URL's que coincidan, con estas expresiones, dentro de la aplicación durante todo el proceso de análisis.

- **Sesiones HTTP**: Almacena y utiliza distintas sesiones, mediante *Cookies* dentro de una misma aplicación, forzando a que todas las peticiones se hagan en una sesión particular sin destruir ninguna otra.

- **Proxy interceptor**: Funcionalidad principal de la herramienta que permite ver y almacenar un registro de todas las peticiones que se hacen hacia una aplicación web y sus respectivas respuestas.

- **Interfaz**: ZAP permite cambiar el aspecto de su interfaz para cumplir determinados roles y facilitar el acceso hacia funcionalidades propias de cada rol, de los cuales son: Modo Seguro, Modo Protegido, Modo Estándar, Modo Ataque.

- **Reglas**: OWASP ZAP soporta reglas para análisis activo o pasivo, todas las reglas que utiliza ZAP son complementos propios de la herramienta, su actualización es sencilla a través del administrador de complementos.

- **Alcance**: OWASP ZAP permite determinar el alcance de búsqueda en

la aplicación, utilizando el grupo de URL's relacionadas en la característica de contexto. Por defecto el alcance es nulo, es decir, no tiene límite.

3.2.2.2. Arachni

Es una herramienta gratuita de código abierto de análisis dinámico de vulnerabilidades en aplicaciones web, está desarrollada en *Ruby*. Presenta una interfaz web que permite una fácil configuración, esta característica lo convierte en un escáner multiplataforma (Windows, Linux y Mac OS X) [30]. En la Figura 3.7 se puede observar la interfaz gráfica que presenta Arachni. Adicionalmente, es capaz de realizar análisis de aplicaciones web en colaboración con distintas instancias o "*dispatchers*" de la aplicación, que funcionan como uno solo, para reducir el tiempo de rastreo y análisis. Arachni puede ampliarse y mejorar en funcionalidades. Esto gracias a que es una herramienta de código abierto.

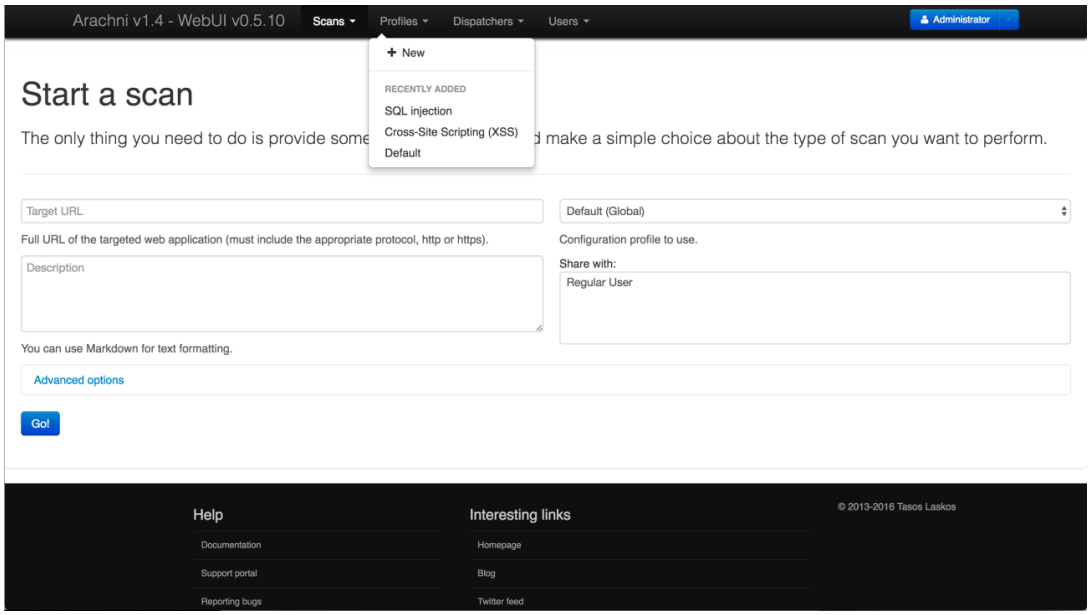

Fig. 3.7 Captura de Pantalla de la interfaz gráfica de Arachni

Sus primeras versiones no presentaban una interfaz gráfica y por lo tanto la configuración y análisis de una aplicación se realizaba utilizando la ventana de comandos. En la Figura 3.8 se puede observar la presentación en la ventana de línea de comandos para ejecutar Arachni.

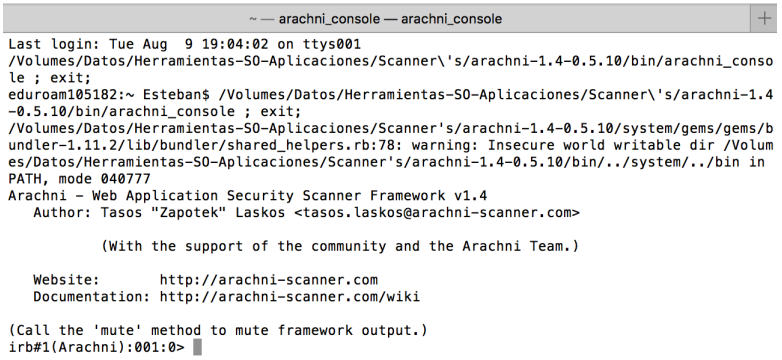

Fig. 3.8 Captura de Pantalla de línea de comandos de Arachni

Arachni cuenta con una estructura altamente configurable, posee distintos complementos específicamente diseñados para ambas etapas del análisis: pasiva y activa. Estos complementos mejoran la precisión de los resultados obtenidos en cada prueba. En la configuración por defecto todos los complementos se encuentran habilitados para ambas etapas del análisis.

Arachni puede ejecutar tres distintos tipos de análisis:

- **Análisis Directo:** Con la configuración establecida y con la URL de la aplicación que se desea analizar, se realiza un análisis completo sin ninguna otra configuración adicional.

- **Análisis Remoto:** Mediante el uso de instancias o "*dispatchers*" se puede asignar a ellos la tarea de análisis de aplicaciones web sin que el tráfico se genere desde el ordenador de origen.

- **Análisis en Distribuido o *"Grid":*** Con un grupo de instancias o "*dispatchers*" de Arachni, se pueden distribuir la carga de trabajo del análisis de aplicaciones web a lo largo de la red de instancias y acelerar los procesos de rastreo y análisis en general.

Las principales características de Arachni son:

- **Autenticación:** Arachni permite la autenticación a la aplicación web basada en *cookies*, SSL, formulario, NTLMv1 y Kerberos entre otros.

- **Detección Automática de Desconexión:** Arachni detecta automáticamente el método de desconexión del usuario de la aplicación web y permite además la reconexión de la sesión para continuar con el análisis sin interrupciones.

- **Abstracción de Interfaz:** Arachni puede ser utilizado directamente por el usuario mediante dos interfaces: Interfaz de ventana de línea de comandos, Interfaz Web.

- **Parámetros de Entrada:** Arachni permite la utilización de parámetros de entrada configurados por el usuario para cada análisis. Utiliza un formato básico de clave/valor para determinar el nombre del campo y el valor que se otorga en ese campo en cada formulario.

- **Características ofrecidas para el desarrollador:** Arachni cuenta con características que permiten a desarrolladores crear aplicaciones que se pueda comunicar con Arachni y también controlar Arachni. Para ello Arachni cuenta con las siguientes API's que facilitan el trabajo:

    - **API REST:** Mediante esta API los desarrolladores pueden interactuar con Arachni. El formato de respuestas pueden ser objetos JSON o XML.

    - **API RPC:** Esta API facilita a los desarrolladores las tareas de comunicación con la herramienta y la administración de sus funcionalidades en modo distribuido.

3.2.2.3. Acunetix WVS

Es una herramienta comercial especializada en auditar aplicaciones web y busca vulnerabilidades y fallos que puedan comprometer la integridad de la aplicación y la información que contiene. En la Figura 3.9 se puede observar la interfaz de usuario que presenta esta herramienta.

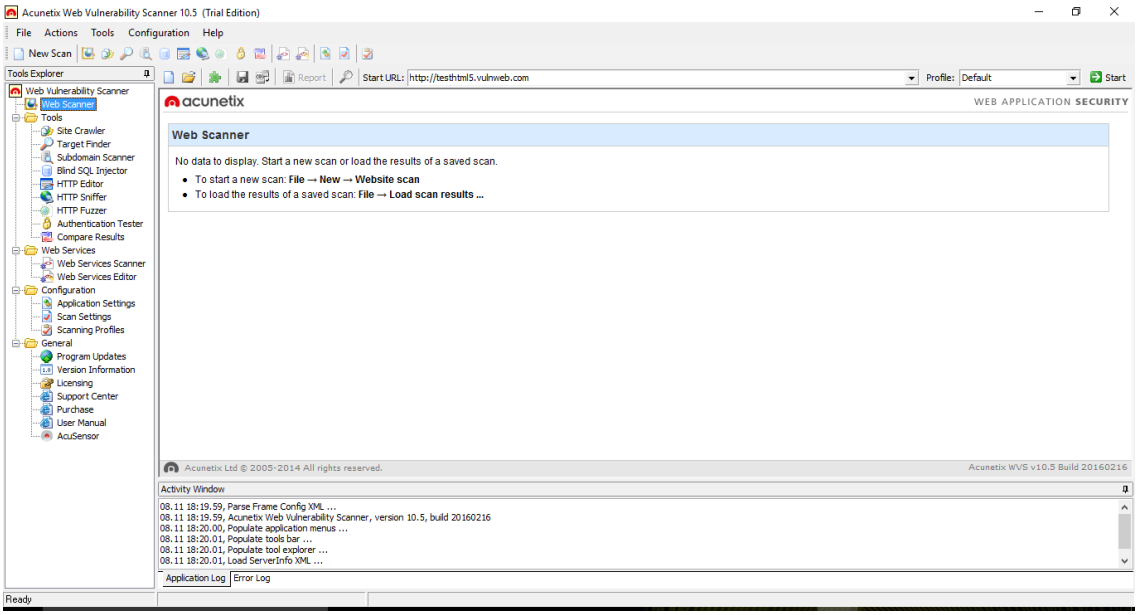

Fig. 3.9 Captura de Pantalla de la interfaz gráfica de Acunetix WVS

Acunetix WVS es muy popular gracias a la gran cantidad de características que ofrece permitiendo configurar la herramienta para aumentar la precisión del análisis, y así, obtener resultados más fiables en su reporte [31]. En su página web ofrece además el servicio gratuito de análisis de vulnerabilidades de aplicaciones que se encuentren en línea.

Las características y funciones más destacadas de Acunetix WVS son:

- **Web Scanner:** Realiza una auditoria automática de un aplicación web, consistente en dos etapas:
    - *Crawiling*: Utilizando *Acunetix DeepScan*, *Acunetix WVS* automáticamente navega dentro de la aplicación, en busca de enlaces y páginas con el fin de establecer una estructura lo más precisa posible de la aplicación que se está analizando. Este proceso enumera todos los archivos dentro de la aplicación y es fundamental para asegurar que todos estos sean analizados.
    - *Scanning*: Con *Acunetix WVS* se ejecuta la búsqueda de vulnerabilidades web dentro de cada fichero presente en la

aplicación, simulando el ataque de un hacker. El resultado es procesado y se muestra en los reportes que se generen.

- **AcuSensor Technology:** Esta tecnología es única de *Acunetix WVS*, permite identificar una mayor cantidad de vulnerabilidades que cualquier análisis dinámico tradicional. Está diseñado para reducir cualquier falso positivo. Esta mejora en la precisión de búsqueda se produce al combinar las técnicas de análisis dinámico con el análisis de código al tiempo en que este se ejecuta. Para lograr esto, es necesario que se instale un agente sensor en la aplicación web permitiendo así la comunicación entre el agente sensor y *Acunetix WVS*. De momento es únicamente posible en aplicaciones desarrolladas en PHP y .NET.

- **PortScanner:** *Acunetix WVS* puede realizar un escaneo en los puertos abiertos del servidor web en el cual se encuentra la aplicación que se desea analizar, para revisar la seguridad del servicio que se ejecuta en el puerto. El usuario puede además configurar sus propios servicios de revisión de seguridad en red mediante un script.

- **Target Finder:** Sirve para que *Acunetix WVS* pueda localizar servidores web (puertos 80 y 443) dentro de un rango IP dado. De cada servidor hallado se muestra la cabecera de respuesta y el software del servidor. Los puertos de búsqueda son configurable.

- **Subdomain Scanner:** Identifica los subdominios activos dentro de un dominio.

- **Blind SQL Injector:** Es una funcionalidad para la extracción de datos de una base de datos. Mediante el uso de técnicas de inyección a ciegas de SQL enumera base de datos y tablas, descarga los datos y también lee archivos específicos de sistema del servidor web. Además permite realizar pruebas manuales de tipo inyección SQL sobre

vulnerabilidades descubiertas en un análisis.

- **HTTP Editor:** Permite la creación, análisis y edición de las peticiones HTTP y también de las respuestas del servidor. Puede codificar y decodificar texto y URL's hacia distintos formatos.

- **HTTP Sniffer:** Actúa como un proxy que captura, examina y modifica el trafico HTTP entre un cliente y el servidor. La información capturada se puede importar a la sesión de análisis de la aplicación y servir de guía para el escaneo automático de la misma. Para activar esta funcionalidad en *Acunetix WVS* es necesario previamente configurar en el navegador web a *Acunetix WVS* como proxy HTTP.

- **Authentication Tester:** Permite ejecutar ataques hacia las páginas de autenticación de las aplicaciones, utilizando diccionarios predefinidos. Los diccionarios pueden ser modificados y agregar nombres de usuario y contraseñas personalizadas.

- **Web Services Scanner and Web Services Editor:** Puede ejecutar un análisis automático de vulnerabilidades sobre servicios Web WSDL. El editor puede importar un WSDL local o en línea para editar y ejecutar operaciones de servicios web sobre distintos puertos y obtener un análisis profundo de las respuestas y solicitudes WSDL.

- **Acunetix Web Vulnerability Scanner SDK:** Se puede desarrollar características adicionales para el análisis que hace *Acunetix WVS*. Estas características deben ser escritas en JavaScript y requiere de la instalación de las herramientas para el desarrollo propio de *Acunetix WVS*.

- **Reportes:** Los reportes en *Acunetix WVS* son generados por la aplicación *Acunetix WVS Reporter*, esta aplicación se ejecuta después de haberse completado el escaneo o desde la aplicación principal de

Acunetix WVS y admite la generación de distintos tipos de reportes incluyendo reportes de desarrollo, reportes ejecutivos, reportes completos estándar y reportes comparativos de dos escaneos distintos.

3.2.2.4. HP WebInspect

Es una herramienta desarrollada por HP para el análisis dinámico de vulnerabilidades en aplicaciones web. En la Figura 3.10 se muestra la interfaz de HP WebInspect.

En HP WebInspect la configuración inicial requiere de poca aportación por parte del usuario y los reportes que genera cuando ha terminado el análisis son muy completos [32].

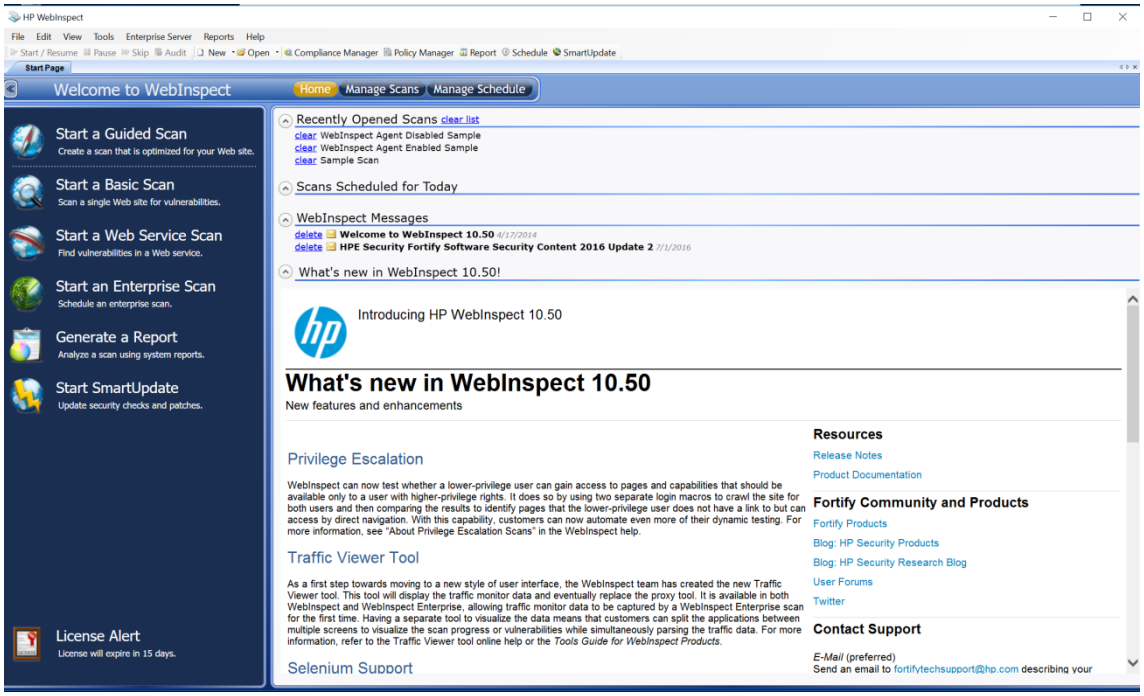

Fig. 3.10 Captura de Pantalla de la interfaz gráfica de HP WebInspect

HP WebInspect utiliza dos modos básicos para ejecutar sus análisis:

- ***Crawling***: Es el proceso en donde HP WebInspect identifica la estructura que es objeto de análisis, en busca de enlaces, formularios, páginas y toda la estructura que compone a la aplicación.

- **Auditoria**: Este proceso realiza el escaneo de las vulnerabilidades en todas las páginas de la aplicación web recopiladas en el modo *crawling*.

HP WebInspect ofrece las siguientes características para realizar análisis de vulnerabilidades:

- **Reportes:** Los reportes en HP WebInspect pueden ser personalizados, se pueden generar reportes orientados a determinadas audiencias indicando el nivel de información que se mostrara en cada caso. Los reportes se presentan en diferentes formatos PDF, HTML, Excel, Raw, RTF o simplemente en formato texto. Además se pueden incluir en los reportes el resumen en graficas del análisis de una aplicación.

- **Hacking Manual:** HP WebInspecte contiene una herramienta que captura las peticiones enviadas y las respuestas recibidas de la aplicación analizada, a través de esto el usuario puede modificarlas y realizar ataques de forma manual.

- **Resumen y Reparacion:** La información de vulnerabilidades que se muestran en la interfaz de HP WebInspect durante un análisis contiene las recomendaciones de reparación de cada una y también incluye instrucciones para evitar que se produzcan esos fallos a futuro. Toda esta información que se incluye es actualizada constantemente y de manera automática por parte de HP WebInspect.

- **Políticas de escaneo:** En HP WebInspect se pueden modificar y adaptar las políticas de escaneo que tiene predefinida la herramienta. De esta forma se reduce el tiempo de análisis.

- **Detalles del análisis todo el tiempo:** En la interfaz de usuario de HP WebInspect se muestra durante todo el tiempo de análisis paneles que indican aspectos como:

- **Sitio:** muestra la estructura jerárquica de los ficheros que componen la aplicación, también muestra por cada fichero las vulnerabilidades halladas en él y el código de estado HTTP de la respuesta del servidor.

- **Secuencia**: muestra la información de los recursos del servidor hallados en la primera etapa(*crawling)* del análisis conducido por HP WebInspect.

- **Análisis de Servicios Web:** HP WebInspect también puede realizar búsquedas de vulnerabilidades en servicios web. Permite evaluar las solicitudes que contengan objetos Servicios Web/SOAP.

- **Exportación de datos:** HP WebInspect cuenta con un asistente que permite exportar en formato XML, toda la información recopilada durante un análisis incluyendo comentarios, archivos ocultos, cookies, formularios, URL's, peticiones, sesiones. El usuario es el encargado de determinar la información que se incurra en el archivo exportado.

- **Configuración de Parámetros de Entrada:** Con la herramienta para el diseño de archivos de prueba HP WebInspect permite al usuario crear ficheros *.WSD que contienen los valores que HP WebInspect introducirá en los campos de los formularios que halle durante el análisis de una aplicación.

- **Agentes de Análisis especializados:** HP WebInspect cuenta con agentes de búsqueda especialmente diseñados para realizar análisis sobre aplicaciones desarrolladas en entornos como IBM WebSphere, Adobe ColdFusion, Microsoft.NET, Lotus Domino, BEA Weblogic, Macromedia JRun, Oracle Application Server, Jakarta Tomcat.

4. TRABAJOS RELACIONADOS

La literatura relacionada con la investigación de este trabajo es escasa. No obstante, la experiencia de estos trabajos ha servido de guía para obtener las herramientas de escaneo de vulnerabilidades web y las aplicaciones web adecuadas para realizar los experimentos conducidos y expuestos en este documento.

Se han considerado dos tipos de literatura: a) Trabajos que realizan comparativas de efectividad de las herramientas de análisis de escáneres de vulnerabilidades en aplicaciones web y b) Trabajos que utilizan un IDS de red para detectar ataques hacia vulnerabilidades de aplicaciones web.

4.1. Evaluación de Escáneres

En [3], se hizo el análisis de vulnerabilidades en cinco aplicaciones web públicas en Internet. Su selección se basó en la popularidad de su uso en Indonesia y el tipo de información que circula y se almacena en cada aplicación web (en algunos casos información de tipo confidencial). La comunicación hacia cada una de las aplicaciones web se hizo directamente a través de una conexión de internet de un proveedor local.

Se utilizaron dos herramientas de análisis de vulnerabilidades para realizar las pruebas: Acunetix WVS y Vega WVS. En Acunetix WVS, herramienta comercial, se utilizó una configuración personalizada para la búsqueda de vulnerabilidades en cada aplicación web analizada. En la configuración de la herramienta gratuita, Vega WVS, no se hicieron ajustes adicionales y se utilizó la configuración por defecto del escáner. Los resultados del análisis mostraron diferencias en cuanto al número de vulnerabilidades de alto y bajo riesgo que cada escáner presentó en sus respectivos informes. En el número de vulnerabilidades halladas de mediano riesgo las dos herramientas de análisis obtuvieron similares resultados. El autor concluye explicando que las

diferencias que se mostraron en las vulnerabilidades identificadas se debe a que cada herramienta realiza su análisis de forma distinta y que a pesar de ello son herramientas muy útiles para los desarrolladores que desean verificar y mejorar la seguridad de su aplicación.

En [5] se realiza una evaluación de dos herramientas de análisis dinámico: Zed Attack Proxy (OWASP ZAP) y Skipfish. Ambas herramientas son gratuitas, multiplataforma y muy utilizadas, características consideradas muy importantes para la realización de las pruebas, por parte del autor. Se utilizaron DVWA Damn Vulnerable Web Application y WAVSEP Web Application Vulnerability Scanner Evaluation Project como aplicaciones web vulnerables. En la metodología utilizada por los investigadores, solo consideran las vulnerabilidades detectadas para la evaluación de cada herramienta. Con esta información se calcula la precisión de cada escáner. El cálculo se realizó de la siguiente forma:

$$Porcentaje\ de\ Falsos\ Psitivos = \frac{N_b}{N_b + N_c}\ \text{X}\ 100$$

N_b: $Número\ de\ vulnerabilidades\ correctamente\ identificadas$

N_c: $Número\ de\ vulnerabilidades\ mal\ identificadas$

La herramienta OWASP ZAP obtuvo el mejor porcentaje de aciertos. El número de falsos positivos también fue analizado, siendo la herramienta OWASP ZAP la que obtuvo el mayor número de falsos positivos. Finalmente, los resultados muestran que ambas herramientas tienen dificultad para identificar vulnerabilidades de tipo RFI (en inglés *Remote File Inclusion*)..

En [33], se evaluaron 13 herramientas comerciales de análisis dinámico de vulnerabilidades. La metodología de evaluación utilizada en el estudio fue el Criterio de Evaluación para Herramientas de Análisis de Seguridad en Aplicaciones Web WASSEC (en inglés *Web Application Security Scanner Evaluation Criteria*). Para la evaluación se utilizaron 6 de sus principales

criterios, sin tener en cuenta los criterios de control y administración y el criterio de reporte. Para clasificar las herramientas se calculó el promedio de criterios cubiertos por cada herramienta, mediante la fórmula:

$$Promedio\ de\ criterios\ cubiertos = \frac{\sum C_H}{\sum C_W}$$

$$C_H: Criterios\ que\ cumple\ la\ herramienta$$

$$C_W: Criterios\ de\ WASSEC$$

Las 3 herramientas resultantes con mejor promedio de criterios cubiertos fueron: Acunetix WVS, Ammonite y Burp Suite Professional. Las tres herramientas obtuvieron los mismos resultados en la evaluación del criterio de rastreo (*crawling*). Sin embargo, el puntaje obtenido en la evaluación del criterio de Análisis Inicial (*Parsing*) fue el más bajo para la tres herramientas. Acunetix WVS fue la herramienta que mayor promedio de criterios cubiertos obtuvo, a pesar de que hubo criterios para los cuales tuvo el menor puntaje de las otras 2 herramientas del top 3.

Posteriormente en [34], el autor extendió su trabajo realizado en [33] y ejecutó una comparación de 32 herramientas de análisis dinámico gratuitas, continuando con la metodología de evaluación utilizada en su trabajo anterior. Entre las 32 herramientas consideradas hay escáneres especializados en la búsqueda tipos de vulnerabilidades específicas, como son SQLMap o XSSploit v0.5. Esta es una de las diferencias con su trabajo anterior, donde se utilizaron escáneres comerciales genéricos que detectaban todo tipo de posibles vulnerabilidades. Las herramientas especializadas obtuvieron un bajo promedio de criterios evaluados, en relación al resto de escáneres. Las herramientas que mejor promedio de criterios cubiertos lograron fueron W3AF 1.2-rev, OWASP Zed Attack Proxy (OWASP ZAP), IranWASP v0.9.1.0, Arachni WVS y Skipfish 2.07b.

Algunos trabajos han analizado escáneres de seguridad sobre determinadas vulnerabilidades, por lo general sobre las más comunes.

En [35] las herramientas Acunetix WVS, Rational AppScan Enterprise y QualysGuard Express Suite se configuraron para realizar búsqueda de vulnerabilidades de tipo Inyección de SQL sobre tres aplicaciones vulnerables PCI, WackoPicko y MatchIt. Cada herramienta fue configurada con tres diferentes perfiles de búsqueda: 1) Búsqueda de vulnerabilidades de tipo Blind/Reflected, 2) Búsqueda de objetivos Stored y 3) Búsqueda de cualquier tipo de inyección de SQL existente en la aplicación web. Adicionalmente, para realizar una búsqueda de todas las vulnerabilidades existentes en una aplicación web, se configuró un cuarto perfil de búsqueda general. Finalmente, se utilizó Wireshark como herramienta para el análisis de protocolos de red y capturar los paquetes enviados durante la etapa de rastreo o *crawling* de cada escáner, analizarlo e identificar los ataques, el contenido de formularios enviados al servidor y sus respuestas. Los resultados de este estudio muestran la deficiencia que tienen todas las herramientas para identificar vulnerabilidades de tipo inyección de SQL principalmente la de tipo Almacenado (en inglés *Stored)*. El mejor resultado se obtuvo utilizando un perfil de búsqueda general en lugar de uno diferenciado en cada herramienta.

En [36] se evalúan seis escáneres de vulnerabilidades web gratuitas y/o de código abierto: NetSparker Community Edition, N-Stalker Free 2012, OWASP ZAP, W3Af, Iron WASP y Vega. Como aplicación vulnerable para el análisis se utilizó *WackoPicko*. Se realizaron dos pruebas con las herramientas de análisis que permiten el uso de credenciales de autenticación para el acceso a la aplicación web, una con credenciales de autenticación y otra sin ellas. Los reportes de ambas pruebas fueron comparadas para verificar cambios en los resultados de análisis del escáner. La información fue obtenida del análisis realizado sin parámetros de autenticación. Todas las herramientas utilizadas mostraron un alto porcentaje de falsos positivos en su reporte. Asimismo, se encontró que Iron WASP fue la herramienta que tardó más en culminar su análisis.

En [6] se elaboró un estado del arte sobre escáneres de vulnerabilidades en

aplicaciones web. El trabajo quería responder tres preguntas: 1) ¿Qué vulnerabilidades prueba el escáner?, 2) ¿Cuán importantes son las pruebas realizadas por los escáneres para detectar vulnerabilidades en aplicaciones no intencionalmente vulnerables?, 3) ¿Cuán efectivos son los escáneres? Con el fin de contestar estas preguntas, se utilizaron ocho herramientas de análisis dinámico de vulnerabilidades web y las tareas de evaluación se realizaron en dos etapas: 1) Evaluar el desempeño de cada herramienta analizando versiones antiguas y vulnerables de Drupal, phpBB y Wordpress con cada escáner y 2) Evaluar la precisión que cada escáner en un análisis ejecutado sobre una aplicación web vulnerable "real" y en un ambiente controlado. Se utilizó una aplicación web intencionalmente vulnerable, desarrollada específicamente para este trabajo. Los resultados obtenidos por el autor demostraron que ningún escáner fue capaz de hallar todas las vulnerabilidades en ninguna de las aplicaciones analizadas y además se comprobó que las herramientas utilizadas no lograron identificar vulnerabilidades de segundo orden de las categorías inyecciones de SQL y *XSS*, específicamente del tipo almacenadas. Esto se debe a que las herramientas no son capaces de comprobar si el código malicioso enviado es almacenado en la aplicación.

En [37] se realizó un análisis comparativo de tres escáneres de vulnerabilidades en aplicaciones web con el objetivo de encontrar la herramienta más adecuada a las necesidades de los autores. Los escáneres comparados fueron Nessus, Acunetix Web Vulnerability Scanner y OWASP ZAP. Se utilizaron dos aplicaciones web con diferente arquitectura, en este estudio.

- Proyecto A: Aplicación web ejecutada en un entorno CentOS Release 6.3 con servidor web Apache.

- Proyecto B: Aplicación web que contiene distintos métodos de autenticación ejecutada en un entorno Ubuntu 12,04 LTS con servidor web Apache.

En el estudio se realizaron dos experimentos, el primero consistía en analizar los proyectos A y B con cada una de las herramientas y comparar los resultados obtenidos individualmente. El segundo, consistió en realizar un análisis de ambos proyectos con la herramienta Nessus, utilizando los distintos modos de búsqueda y configuraciones de la herramienta para comparar los reportes. Los resultados del estudio fueron distintos en cuanto al número de vulnerabilidades detectadas. El autor concluyó en que estas diferencias se deben a las características propias de cada herramienta y a los tipos de búsqueda realizadas en cada aplicación.

En [38] se condujo la evaluación de once escáneres de vulnerabilidades web, entre comerciales y gratuitas. Con estas herramientas se analizó las vulnerabilidades presentes en la aplicación web WackoPicko, desarrollada específicamente para este propósito, concebida como una aplicación real, pero con vulnerabilidades intencionalmente dispuestas en distintas partes de la misma. También se utilizó la aplicación web WIVET (en inglés *Web Input Vector Extractor Teaser)* para comparar la capacidad de rastreo de enlaces y la extracción de vectores de entrada en distintos formatos por cada herramienta. Cada escáner tuvo 3 configuraciones: Una básica en la que no se otorgó ninguna credencial o parámetros básicos de entrada; una intermedia solo con los datos de autenticación de la aplicación y una avanzada en la que el escáner se inicializó en modo proxy y el usuario se encargó de navegar la aplicación para que el escáner almacenara todos los campos y enlaces de cada página recorrida. Posteriormente, se realizó el ataque con esta información. Los resultados de la evaluación mostraron que ocho vulnerabilidades no fueron detectadas por ningún escáner y que los escáneres con mayor número de vulnerabilidades detectadas fueron Acunetix, HP WebInspect y Burp.

4.2. Uso de Sistemas de Detección de Intrusos en el Análisis de Vulnerabilidades Web

Algunas de las propuestas recientes han enfocado su investigación en definir reglas que permitan mejorar la eficiencia de los sistemas de detección de intrusos en la identificación de patrones de determinados tipos de ataques a vulnerabilidades en aplicaciones web. Estas propuestas se basan en Snort ya que es uno de los sistemas de detección de intrusos más utilizado en el mercado.

En [39] se propuso filtrar ataques de tipo Inyecciones de SQL en aplicaciones web utilizando Snort. Se utilizaron cinco reglas nuevas para que Snort pudiera detectar ataques de Inyección de SQL. Se utilizó DVWA como aplicación vulnerable para ser analizada. El resultado del experimento demostró un alto nivel de precisión en las nuevas reglas propuestas. Posteriormente, estos resultados fueron comparados con técnicas similares propuestas en trabajos anteriores para demostrar lo extensible que puede ser este conjunto de reglas propuestas. Los ataques realizados para evaluar las reglas propuestas fueron realizados de forma manual, sin intervención de herramientas automáticas de ataque.

En [40] se propuso el uso de tres reglas nuevas para mejorar la precisión en la detección de vulnerabilidades de tipo Inyección de SQL, XSS y *Command execution*. Estas reglas probaron utilizando la aplicación DVWA y los ataques se realizaron de forma manual, sin intervención de herramientas automáticas de ataque.

5. DISEÑO DE UN MODELO DE PRUEBAS PARA HERRAMIENTAS DE ANÁLISIS DINÁMICO DE APLICACIONES WEB

Para alcanzar los objetivos propuestos en el Capítulo 1, Se propone un modelo distinto a los utilizados en la literatura para evaluar una herramienta de análisis dinámico de vulnerabilidades en aplicaciones web. Con este nuevo enfoque, la evaluación del escáner se podrá realizar contando con más datos que los utilizados en modelos anteriores.

La metodología utilizada en la literatura se apoya únicamente en los resultados del reporte obtenido por los escáneres de vulnerabilidades, sobre en aplicaciones web, para hacer una comparativa sobre la efectividad de este tipo de herramientas y posteriormente emitir un criterio al respecto. A pesar de que este modelo es una metodología válida y ampliamente utilizada en la literatura, el modelo propuesto en este trabajo, cuenta con un elemento que no se encuentra presente en trabajos anteriores y que permite obtener datos adicionales que son procesados automáticamente y que mejoran el estudio evaluativo de las herramientas de análisis dinámico.

En este capítulo se explica detalladamente los componentes del modelo propuesto, el cual posteriormente se utiliza para ejecutar experimentos y verificar si éste es funcional y permite adquirir la información adicional para alcanzar los resultados esperados. Los componentes que conforman el modelo propuesto son: Nodo Atacante, Nodo Servidor Web y Nodo IDS.

5.1. Modelo de Pruebas Tradicional

El modelo tradicional de pruebas utilizado en la literatura para evaluar un escáner de vulnerabilidades, consta de dos componentes: Una herramienta de análisis dinámico y una aplicación web vulnerable. Se revisa el reporte generado al finalizar el análisis de la aplicación y se verifica la cantidad de

aciertos reportados. Posteriormente, el número de aciertos se compara con el resto de herramientas evaluadas. La arquitectura de componentes para la evaluación resulta simple, como se observa en la Figura 5.1, donde se presentan los componentes que conforman este modelo tradicional.

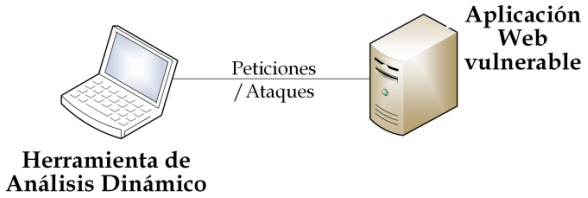

Fig. 5.1 Componentes de Modelo Tradicional

En la Figura 5.2 se presenta el flujo de trabajo utilizado en trabajos previos para realizar la evaluación de este tipo de herramientas.

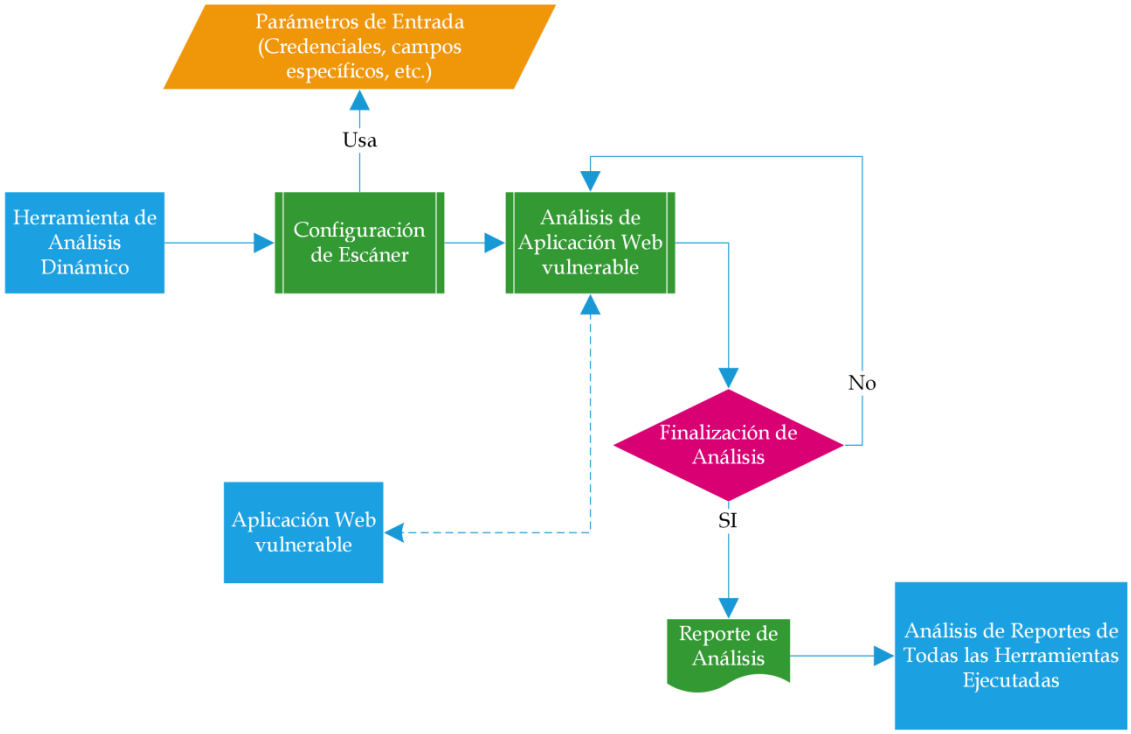

Fig. 5.2 Diagrama de Modelo Tradicional

5.2. Modelo de Pruebas Propuesto

El modelo de pruebas que se propone en este trabajo, consta de una estructura

con tres nodos principales: un Nodo Atacante, un Nodo Servidor Web y un Nodo IDS. Esto difiere del modelo tradicional que solo considera el atacante y la aplicación web atacada. En la Figura 5.3 se muestran los componentes del modelo propuesto y la interacción entre ellos.

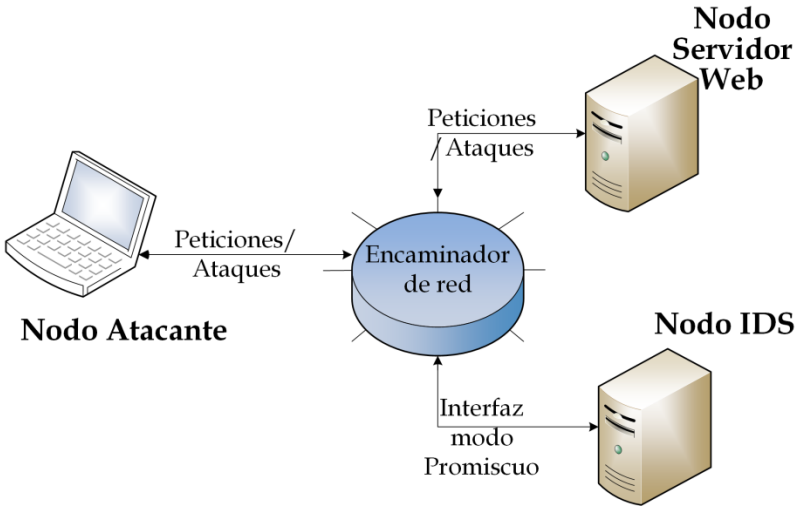

Fig. 5.3 Componentes de Modelo Propuesto

El modelo propuesto adiciona como componente, un sistema de detección de intrusos de red, que interviene durante todo el tiempo de ejecución de la herramienta de análisis dinámico, capturando los paquetes que se generan del escáner hacia la aplicación web y analizando cada uno de ellos en busca de patrones que sugieran el intento de un ataque hacia cualquiera de las vulnerabilidades que contenga la aplicación web.

En este modelo se generan 2 reportes distintos que deben ser analizados:

- **Reporte de vulnerabilidades halladas en la aplicación web**: Este reporte procede de la herramienta de análisis dinámico utilizada.

- **Reporte de alertas de ataque**: Es el reporte generado por el IDS durante la ejecución del escáner, este reporte presenta las alertas de ataque identificadas.

Con estos dos reportes y la documentación de las vulnerabilidades existentes

en la aplicación web, se revisa cada alerta de ataque del reporte del IDS y se compara con las vulnerabilidades reportadas por parte de la herramienta de análisis dinámico. De esta forma, se conoce si se ejecutó algún ataque por parte del escáner hacia alguna vulnerabilidad existente en la aplicación web y que no fue reportada en el informe final del escáner. Además, con el uso del reporte del sistema de detección de intrusos, se puede conocer si la herramienta utilizada realiza ataques que buscan, dentro de la aplicación web, al menos las vulnerabilidades presentes en el listado del OWASP Top 10 de 2013.

Del uso de la documentación de las vulnerabilidades de la aplicación web y el reporte detallado de la herramienta de análisis dinámico se obtienen los falsos positivos de las vulnerabilidades reportadas como ciertas por parte del escáner. El escáner debe obtener una mínima cantidad de falsos positivos posibles.

En la Figura 5.4 se puede observar el flujo de trabajo dentro del modelo propuesto.

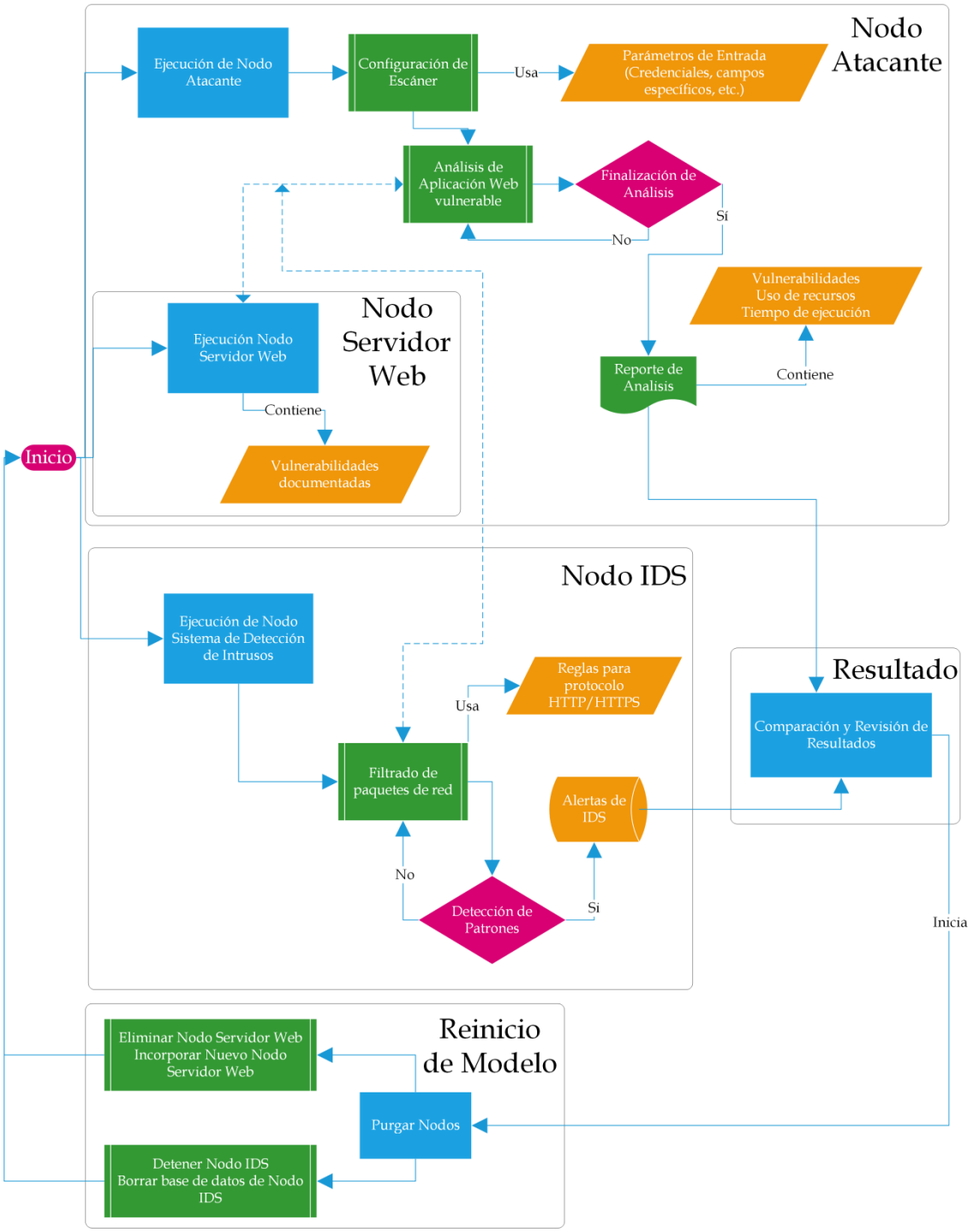

Fig. 5.4 Flujo de Trabajo en Modelo Propuesto

5.2.1. Componentes de Modelo Propuesto

Este modelo consta de tres componentes que se denominan "nodos", cada uno de ellos cumple una funciona especifica detallada a continuación

5.2.1.1. Nodo IDS

Este nodo es el encargado de hospedar y ejecutar el sistema detector de intrusos seleccionado para analizar el tráfico generado durante el ataque de cada escáner hacia cada aplicación vulnerable. En la metodología propuesta en este trabajo, el sistema de detección de intrusos es fundamental e imprescindible para la evaluación de herramientas de análisis dinámico de vulnerabilidades, pues es con la información que este nodo recopila y procesa, que se puede verificar el adecuado funcionamiento del escáner evaluado. Asimismo, se han utilizado un conjunto de herramientas que facilitan el procesamiento de los datos emitidos por el IDS, de esta forma el reporte entregado por este nodo es más sencillo de analizar y compararlo con el reporte del escáner de vulnerabilidades.

De las reglas que utilice el sistema de detección de intrusos dependerá la precisión en que determina el intento de ataque o no y por lo tanto incide en su reporte y posterior análisis. Se recomienda el uso de la mayor cantidad de reglas posibles y más actuales que analicen trafico HTTP y HTTPS para cubrir cualquier posibilidad de ataque y que el IDS logre identificarlos y reportarlos.

Al finalizar y para iniciar un nuevo análisis de vulnerabilidades, con este modelo, es necesario respaldar en otra ubicación y luego eliminar todos los reportes y bases de datos generadas por el IDS, para que en cada prueba que se realice los resultados no puedan ser modificados por la existencia de datos de pruebas anteriores.

El nodo IDS del modelo capturara todos los paquetes de datos que se transmitan desde el escáner de vulnerabilidades hacia la aplicación web y viceversa

5.2.1.2. Nodo Servidor Web

En este nodo se encuentran las aplicaciones vulnerables que serán objeto de ataque por parte de las herramientas de análisis dinámico seleccionadas. Es recomendable el uso de aplicaciones con vulnerabilidades documentadas para que se pueda comprobar la verdadera existencia o no de las vulnerabilidades halladas por cada escáner. Así se puede verificar falsos positivos y se identifican ataques realizados a vulnerabilidades existentes, pero que finalmente no fueron reportados.

En cada prueba este nodo debe ser reiniciado para hospedar únicamente la aplicación objeto de prueba y sus requerimientos específicos. Así, se hace posible trabajar con configuraciones inalteradas que permiten analizar las aplicaciones tal como fueron desarrolladas y no obtener resultados equivocados.

5.2.1.3. Nodo Atacante

En este nodo se sitúan cada una de las herramientas de análisis dinámico que se utilizaran y evaluaran. Su función es la de ejecutar análisis completos sobre las aplicaciones vulnerables presentes en el Nodo Servidor Web y emitir el reporte correspondiente de cada una de ellas.

La configuración de la herramienta dependerá de lo que se pretenda buscar y de la experticia en el uso del escáner por parte del usuario. Se recomienda el uso de la configuración por defecto, que traen la mayoría de las herramientas de este tipo, ya que es la más sencilla de ejecutar y abarca, al menos, a todas las vulnerabilidades presentes en el listado de OWASP Top 10 2013. El uso de parámetros de entrada específicos es también recomendado, para que de esta forma la herramienta logre acceder a la mayor cantidad de ficheros pertenecientes a la aplicación web y no se encuentre con problemas de autenticación o acceso. Si el escáner utilizado permite rastreo proxy, la obtención de parámetros de entrada para el análisis será más sencillo.

5.3. Configuración y Componentes de Experimento con Modelo Propuesto

Con el fin de comprobar la funcionalidad del modelo que se propone, se realizaron una serie de experimentos utilizando cuatro herramientas de análisis dinámico de vulnerabilidades, dos de ellas gratuitas y las restantes dos, herramientas comerciales.

A continuación se especifican, cada uno de los componentes de software y hardware, así como las configuraciones utilizadas en el desarrollo de estos experimentos.

5.3.1. Componentes

El sistema se ejecutó en un total de dos ordenadores físicos. El primero de ellos realizó únicamente el rol de nodo atacante. En el segundo ordenador se ejecutaron dos máquinas virtuales sobre las cuales se ejecutaban los roles de nodo servidor web y nodo aplicación vulnerable. En la Figura 5.5 se puede observar los distintos componentes físicos y virtuales utilizados en los experimentos y la interacción de cada uno de ellos en el modelo.

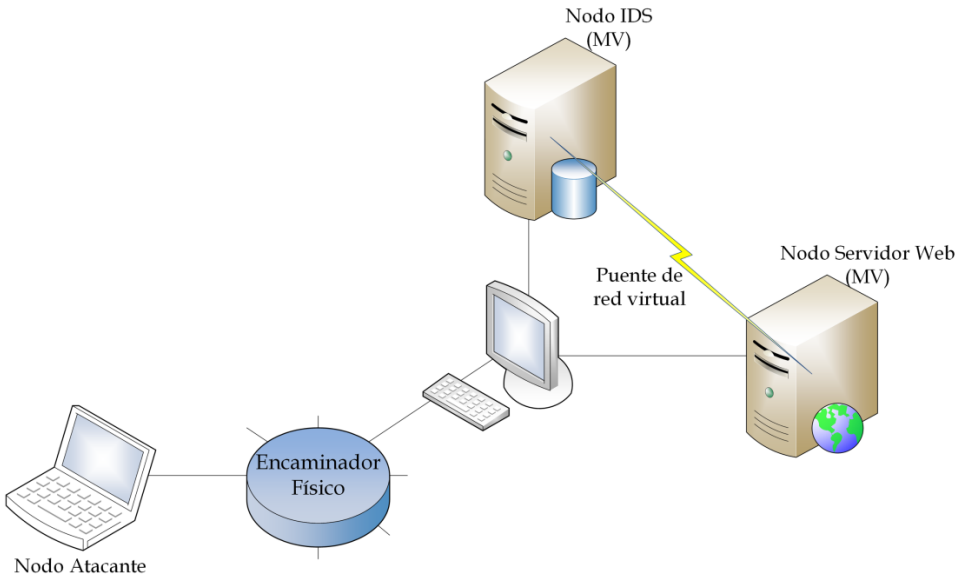

Fig. 5.5 Diagrama de Componentes Físicos de Prueba

Los dos ordenadores físicos se comunican mediante una red por cable y un encaminador de red en medio. El ordenador encargado de ejecutar dos nodos del modelo propuesto de forma virtual, realizó la interconexión de los nodos virtuales con la red física a través de su interfaz de red y el uso de un puente virtual de red.

5.3.2. Herramientas de Análisis Dinámico de Vulnerabilidades

En este trabajo se utilizaron cuatro herramientas de análisis dinámico. Cada una de las cuales fue considerada por los resultados obtenidos en trabajos anteriores, así como la popularidad de su uso dentro del mercado especializado.

Las aplicaciones seleccionadas fueron:

OWASP ZAP: Para este trabajo se utilizó la versión v2.4.3.

Arachni: La versión utilizada en el desarrollo de este trabajo fue la versión v1.4.

Acunetix WVS: En este trabajo se utilizó la versión de prueba gratuita v10.5. de 14 días.

HP WebInspect: La versión utilizada en este trabajo fue la v10.5 de prueba gratuita. Esta versión permite utilizar durante 15 días todas las capacidades de la herramienta.

Cada escáner se ejecutó utilizando su configuración por defecto, que de acuerdo a lo explicado en el manual de usuario de cada una de ellas, realiza una búsqueda exhaustiva de la estructura de la aplicación para posteriormente sobre ello realizar un análisis completo en búsqueda de al menos todas las categorías de vulnerabilidades del OWASP Top 10. Así mismo, todas estas herramientas cuentan con la posibilidad de realizar captura de datos mediante el uso de la funcionalidad de rastreo proxy. Esto permite que los parámetros de entrada para cada formulario y principalmente las credenciales de acceso a la

aplicación web se generen mediante una navegación manual previa.

Detalles sobre las características particulares de cada una de estas herramientas, se encuentran expuestas en el Capítulo 3 del presente trabajo.

5.3.3. Aplicaciones Vulnerables

Las aplicaciones vulnerables utilizadas fueron seleccionadas por sus características particulares, el amplio uso que han tenido en la literatura y la documentación que posee cada una.

A continuación se describen cada una de las características de las aplicaciones web vulnerables seleccionadas para este trabajo.

- **DVWA**

Es una aplicación de código abierto desarrollada por la compañía RandomStorm. Esta desarrollada en PHP y como gestor de base de datos utiliza MySQL. Se utiliza principalmente para el aprendizaje y entrenamiento de desarrolladores y usuarios que deseen mejorar habilidades o aprender sobre la forma en la que actúa cada una de las vulnerabilidades que contiene la aplicación [41]. En la Tabla 5.1 se detallan el tipo y la cantidad de vulnerabilidades con las que cuenta DVWA [5].

Vulnerabilidad	Cantidad
RXSS(XSS Reflejado)	1
PXSS(XSS Persistente)	1
Inyección de SQL	2
Inyección de SQL a Ciegas	1
CSRF Falsificación de Petición en Sitios Cruzados	1
LFI Inclusión de Fichero Local	1
CMDExec Inyección de Comandos de Sistema	1

Tabla 5.1 Vulnerabilidades presentes en DVWA

- **WackoPicko**

Es una aplicación vulnerable desarrollada por Adam Doupé para las pruebas de detección de vulnerabilidades en [38]. Su diseño fue pensado para simular el comportamiento realista de una aplicación web normal. Su función es la de una aplicación que comparte y vende imágenes. Contiene una página de autenticación para el ingreso al sitio. Permite comentar cada imagen disponible y subir fotografías. También cuenta con una página de comentarios y un área de administración con la única funcionalidad de crear nuevos usuarios en el sitio [42]. En la Tabla 5.2 se detallan las vulnerabilidades con las que cuenta WackoPicko.

Vulnerabilidad	Cantidad
XSS Reflejado	3
XSS Almacenado	2
Inyección de SQL Almacenada	1
Inyección de SQL Reflejada	1
Salto de Directorio	1
Inyección de Comandos de Sistema	1
Inclusión de Fichero Local	1
Modificación de Parámetros	1
Error Lógico	1
Usuario/Contraseña Débil	1
Navegación Forzada	1
Fallo en la ID de Sesión	1

Tabla 5.2 Vulnerabilidades presentes en WackoPicko

Las aplicaciones utilizadas en este trabajo, al finalizar de cada análisis, fueron restauradas a su estado original. Reutilizando una copia sin alterar de la máquina virtual con la aplicación instalada. De esta forma las respuestas de la aplicación, a cada ataque realizado por el escáner, no se verían de ninguna forma alteradas, asegurando integridad en los resultados.

5.3.4. Sistema de Detección de Intrusos

Debido a la importancia de este componente dentro del modelo propuesto, es necesario contar con un sistema de detección de intrusos reconocido y que sea confiable, que cuente con gran número de reglas de detección que se puedan utilizar y además que sea gratuito. Estas condiciones las cumple Snort[43], que es el sistema encargado de analizar todo el tráfico de entrada y salida del Nodo Servidor Web. Snort se configuró con un total de 70.165 reglas distintas para el análisis de trafico HTTP/HTTPS. Tras la ejecución de un análisis, la base de datos de alertas se respaldó y se purgó eliminando los registros de navegación y alerta generados. De esta forma se aseguró que en ningún momento se mezclen datos provenientes de análisis anteriores que puedan alterar los resultados. En el Anexo B se detallan las características particulares de Snort.

Las reglas utilizadas en Snort no permitían el reconocimiento de ataques, específicos, de tipo persistente o reflejado, para el caso de las vulnerabilidades de tipo Secuencia de Comandos en Sitios Cruzados XSS (en inglés *Cross Site Scripting)* e inyección de SQL. Esto hizo que las tablas comparativas obtenidas de los resultados se ajustaran de acuerdo a la información que cada herramienta aportaba en sus reportes. El IDS Snort sí cuenta con reglas que permiten identificar los ataques de tipo inyección a Ciegas de SQL, por lo que este tipo de vulnerabilidad tuvo una clasificación distinta en las tablas comparativas en el Capítulo 6.

5.3.5. Software y Hardware Adicional

Además de las herramientas de análisis dinámico, el sistema detector de intrusos y las aplicaciones web vulnerables, fue necesario un conjunto de aplicaciones y características físicas para la correcta ejecución de cada uno de los componentes que conforman el modelo de análisis utilizado en este trabajo.

A continuación se detallan cada una de estas características, señaladas en cada nodo componente del modelo.

5.3.5.1. Nodo Atacante

Las características del ordenador utilizado como nodo atacante son las siguientes: *Macbook Pro* con procesador *Intel(R) Core(TM)* i5 2.7GHz, 8 GB de memoria *RAM* y sistema operativo *Mac OS X v10.11.3 "El Capitan"*.

Para la ejecución de herramientas que solo pueden ejecutarse sobre sistemas operativos *Microsoft Windows*, se utilizó una partición con *bootcamp* sobre la cual se instaló *Microsoft Windows 10*, conservando todas las características físicas de procesador y memoria *RAM.*

5.3.5.2. Nodo Servidor Web

Este nodo fue virtualizado en un ordenador con las siguientes características: *Mac Mini* con procesador *Intel(R) Core(TM)* i7 2.6GHz, 16 GB de memoria *RAM* y sistema operativo *Mac OS X v10.11.3 "El Capitan"*.

La máquina virtual que actúa como Nodo Servidor Web tiene las siguientes características: sistema operativo Linux con distribución Ubuntu Server 14.04 LTS x86, un procesador virtual y 1GB de memoria RAM asignada. En este nodo se instalaron las aplicaciones web *DVWA* y *WackoPicko*, que para su correcto funcionamiento fue necesario además instalar las siguientes aplicaciones:

- Apache como servidor web en su versión estable v2.2.
- MySQL Community Server en la versión estable v5.5 como gestor de base de datos.
- Darkstat se utilizó para monitorizar la red durante los análisis de cada aplicación. Es una herramienta gratuita desarrollada por Emil Mikulic, presenta y realiza en tiempo real estadísticas de uso de red determinando *host* y puertos de origen y destino en cada caso, genera gráficos de tráfico y reportes individuales por *host*. Tiene soporte para IPv6 y el acceso a sus reportes es a través del navegador web[44].

5.3.5.3. Nodo IDS

Este nodo fue virtualizado en el mismo ordenador que el Nodo Servidor Web. Asimismo, las características de esta máquina virtual son: sistema operativo Linux con distribución Ubuntu Server 14.04 LTS x86, un procesador virtual y 1GB de memoria RAM asignada. En esta máquina se instalaron el sistema detector de intrusos Snort y las siguientes herramientas para la gestión y almacenamiento del registro de actividad y alertas de Snort:

- Apache como servidor web en su versión estable v2.2.
- MySQL Community Server en la versión estable v5.5 como gestor de base de datos.
- Barnyard2 como modulo procesador de ficheros binarios generados por Snort. Software gratuito y de código abierto.
- Snorby como aplicación web para monitorizar y gestionar las alertas generadas por Snort. Aplicación gratuita y de código abierto, desarrollada en *Ruby on Rails*.

En la Tabla 5.3 se presenta un resumen de las características de cada nodo del modelo de pruebas utilizado.

Macbook Pro		**Mac Mini**	
Nodo Atacante		**Nodo Servidor IDS**	**Nodo Servidor Web**
SO	Windows 7.1 Profesional	Ubuntu Server 14.04 LTS x86	Ubuntu Server 14.04 LTS x86
Hardware	8GB Ram, Intel Core i5 2.7GHz	1GB Ram, 1 CPU, (Virtual)	1GB Ram, 1 CPU, (Virtual)
Software	Acunetix WVS, OWASP ZAP, HP WebInspect, Arachni	Snort IDS, MySQL, Barnyard2, Apache 2, Snorby	Apache 2, MySQL, DVWA, Wackopicko, Darkstat

Tabla 5.3 Características físicas del entorno de pruebas

6. ANÁLISIS RESULTADOS

En este capítulo se evalúan los resultados obtenidos por las herramientas de análisis dinámico de vulnerabilidades, utilizando el modelo de pruebas especificado en el Capítulo 5. En estos experimentos se utilizaron 4 herramientas de análisis dinámico de vulnerabilidades para evaluar 2 aplicaciones web vulnerables. Posteriormente, se compararon los resultados obtenidos en los reportes de cada herramienta y se contrastaron con los reportes de Snort. Con estos resultados se hizo las tablas comparativas de vulnerabilidades halladas, falsos positivos y los gráficos de uso de recursos de red, tiempo de ejecución y número de peticiones hechas durante todas las pruebas.

6.1. Tiempo y Recursos de Red

El tráfico generado por cada una de las herramientas utilizadas fue comparado como se muestra la Figura 6.1. En ella se puede apreciar el tráfico variable generado por cada herramienta. Se destaca principalmente el tráfico generado por Acunetix WVS en el análisis de la aplicación.

Durante la ejecución de cada herramienta el tráfico generado no fue motivo de sobrecarga o lentitud de la misma ni tampoco incidió en la disponibilidad del sitio atacado. Por tanto no es un factor determinante, en este caso particular, para el rendimiento estas herramientas.

Lo contrario sucede con el tiempo empleado por cada herramienta en cada análisis. La mayor cantidad de tiempo invertido por las herramientas fue durante la etapa de reconocimiento o *crawling* del sitio y esto afecto al tiempo total del análisis, incluido el tiempo empleado en la etapa activa del análisis. Este fue el caso de la herramienta *HP WebInspect*, que tuvo un tiempo de ejecución considerablemente elevado. Esto se puede observar en la Figura 6.2.

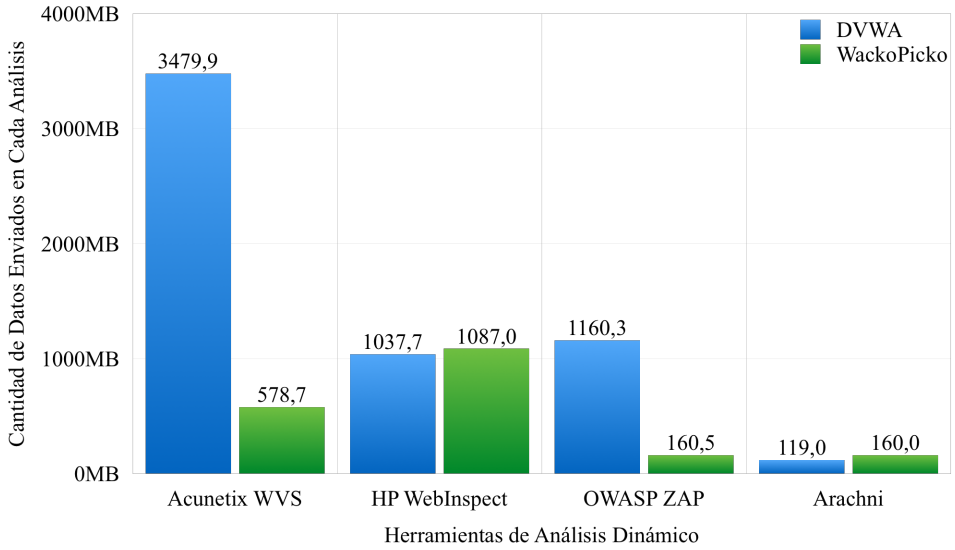

Fig. 6.1 Uso de recursos de red en el análisis de cada herramienta.

El tiempo de ejecución de una herramienta de análisis es un factor importante a ser evaluado. La configuración utilizada para realizar el análisis influye mucho en la duración de la ejecución y el reconocimiento de la aplicación web. La configuración y datos utilizados en estas pruebas con cada herramienta, fueron los mismo con el fin de que los resultados puedan ser comparados. Cada una de las herramientas utilizadas en estas pruebas puede configurarse para acotar la profundidad de búsqueda en la aplicación. Asimismo, las herramientas permiten utilizar vectores de entrada para cada uno de los formularios que se puedan presentar durante su ejecución.

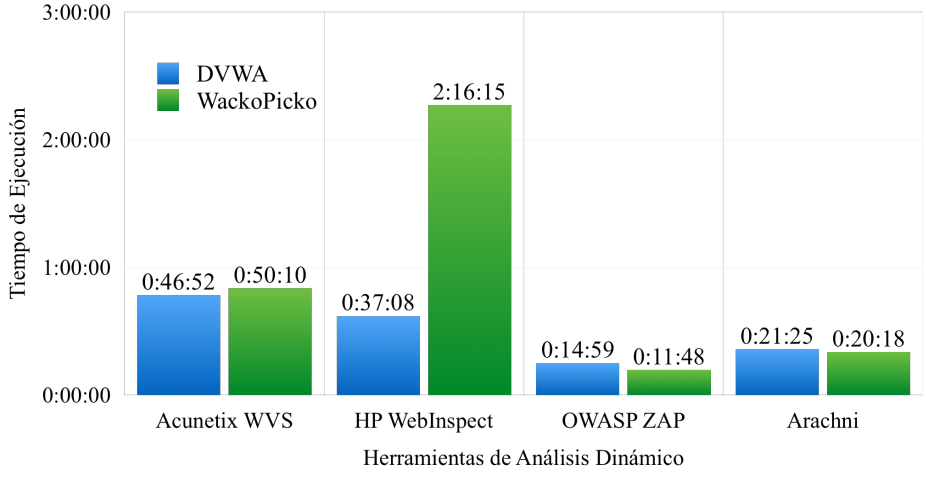

Fig. 6.2 Tiempo de ejecución en el análisis por cada herramienta

6.2. Peticiones Generadas

El número total de peticiones que efectúa cada escáner al momento de realizar su análisis fue calculado sumando las alertas que Snort lanzó. Esto permitió conocer el número de veces que se realizaron intentos de ataque exclusivamente. En la Figura 6.3 se observa que en ninguno de los casos, el número de peticiones es proporcional al tiempo o al flujo de red que se generó. Un claro ejemplo de esto es el presentado por el escáner Arachni, que a pesar de su corto tiempo de ejecución, presenta el mayor número de peticiones de todas las herramientas en ambas aplicaciones.

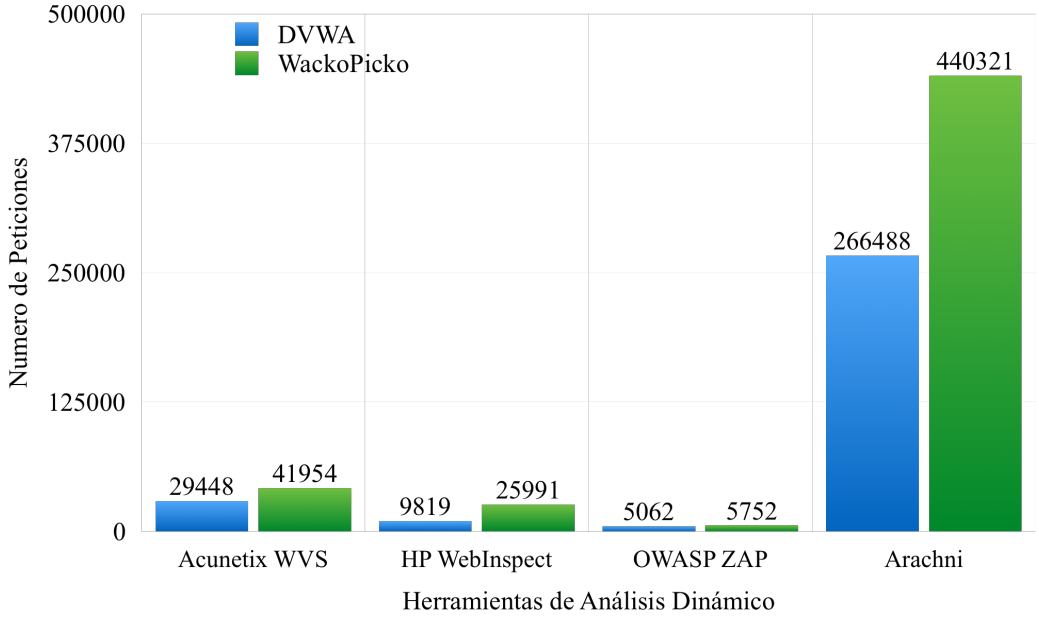

Fig. 6.3 Sesiones generadas por cada herramienta en el análisis

6.3. Resultados con DVWA

En los resultados del análisis de la aplicación DVWA realizado por Acunetix WVS, se detecta la ausencia de un reporte sobre la vulnerabilidad Inclusión de Fichero Local LFI (en inglés *Local File Inclusion)*. Sin embargo, esta vulnerabilidad sí está presente en DVWA que según las alertas de Snort fue explotada durante el análisis, pero Acunetix WVS no la incluyó en su reporte final. Se observa además, que la vulnerabilidad de Falsificación de Petición en

Sitios Cruzados CSRF (en inglés *Cross Site Request Forgery*) no fue verificada su existencia, a pesar de estar presente en la aplicación web DVWA y pertenecer a la clasificación OWASP Top 10 2013. Los resultados obtenidos del análisis de DVWA con todas las herramientas utilizadas en este trabajo se muestran en la Tabla 6.3

Los resultados obtenidos con la herramienta OWASP ZAP no distan de los obtenidos en trabajos anteriores [5]. Sin embargo, en el reporte de Snort se observan ataques sobre vulnerabilidades que no se encuentran presentes en la aplicación. Esto puede considerarse por un lado positivo ya que prueba el comportamiento proactivo de búsqueda por parte de la herramienta y también puede considerarse negativo debido a que esto aumenta el número de falsos positivos alertados por el escáner. El reporte de Snort indica que en este análisis no se verificaron las vulnerabilidades Inclusión de Fichero Local LFI (en inglés *Local File Inclusion)* y Falsificación de Petición en Sitios Cruzados CSRF (del inglés *Cross Site Request Forgery*), herramientas presentes en el OWASP Top 10 2013 [21]. Esta falta de alertas puede reflejar una deficiencia en la búsqueda de vulnerabilidades por parte de la herramienta al momento de realizar su análisis.

El reporte final del análisis de DVWA con HP WebInspect, a pesar de ser el análisis que más tardó en terminar, obtuvo muy pocos aciertos en comparación a los obtenidos por el resto de herramientas. Al igual que con OWASP ZAP, la herramienta no realizó un análisis sobre al menos todas las vulnerabilidades presentes en el OWASP Top 10 2013 [21], como es el caso de las vulnerabilidades Inclusión de Fichero Local LFI (en inglés *Local File Inclusion)* y Falsificación de Petición en Sitios Cruzados CSRF (en inglés *Cross Site Request Forgery*). Esta herramienta fue la única de las cuatro que no pudo hallar ningún caso de Secuencia de Comandos en Sitios Cruzados XSS (en inglés *Cross Site Scripting)* de tipo Persistente.

En el análisis realizado con Arachni sobre DVWA, la herramienta identificó 5 de todas sus vulnerabilidades. Snort identificó ataques de tipo inyección de

SQL pero el reporte de Arachni no lo muestra. Esto es similar al caso de Inyección de Comandos de Sistema CmdExec, en donde Snort identificó un ataque pero Arachni no lo reporto. No logra realizar pruebas para verificar la existencia de la vulnerabilidad Falsificación de Petición en Sitios Cruzados CSRF (en inglés *Cross Site Request Forgery*) igual que ha sucedido con el resto de escáneres.

6.3.1. Falsos Positivos

De los reportes entregados por las herramientas OWASP ZAP, HP WebInspect y Arachni utilizadas en este trabajo, la cantidad y sitios de vulnerabilidades halladas y reportadas no siempre suelen ser ciertas, a esta información se lo denomina falsos positivo. De cada una de estas 3 herramientas se hizo comprobación de la veracidad de su reporte y en la Tabla 6.1 se presentan los resultados.

Vulnerabilidad	OWASP ZAP	HP WebInspect	Arachni
RXSS(XSS Reflejado)	1	4	-
PXSS(XSS Persistente)	-	-	1
Inyección de SQL	-	-	-
Inyección de SQL a Ciegas	-	-	-
CSRF Falsificación de Petición en Sitios Cruzados	-	-	-
LFI Inclusión de Fichero Local	-	-	-
CMDExec Inyección de Comandos de Sistema	-	-	-
Salto de Directorio	1	11	-
DoS Denegación de Servicio	-	-	-
RFI Inclusión de Fichero Remoto	1	-	-
Total de Falsos Positivos	3	15	1

Tabla 6.1 Falsos positivos del análisis de DVWA

Se puede observar que de las tres herramientas, todas presentan falsos positivos. El escáner que mayor cantidad de falsos positivos presento fue HP WebInspect y el escáner que menor cantidad de falsos positivos obtuvo fue

Arachni. Este tipo de comportamiento incide en el coste de reparación de la aplicación web analizada. A mayor cantidad de falsos positivos mayor será el tiempo invertido por parte del desarrollador en verificar si es cierto o no el fallo encontrado en la aplicación web.

6.4. Resultados con WackoPicko

En los reportes de todos los escáneres utilizados no se especifica el subtipo Almacenado de la vulnerabilidad de inyección de SQL. Por tal motivo, en la tabla comparativa se unifican las filas con fines ilustrativos y en la etapa de verificación de falsos positivos se determinan a qué subtipo de vulnerabilidad se hace referencia en su reporte. Esto ocurrió de igual manera con el reporte de Snort, pues sus reglas no lograron determinar subtipos de las vulnerabilidades inyección de SQL y Secuencia de Comandos en Sitios Cruzados XSS, por tal motivo y con fines ilustrativos se unificaron las filas en la tabla comparativa final.

El análisis efectuado por la herramienta Acunetix WVS, es el más completo en cuanto al número de vulnerabilidades halladas, presenta 6 de las 12 vulnerabilidades presentes en WackoPicko. Fue el único en hallar la vulnerabilidad de Contraseña débil. De acuerdo al reporte de Snort en este análisis Acunetix WVS realiza un ataque a la vulnerabilidad de Salto de Directorio y a pesar de estar presente en la aplicación web, este escáner no lo reporta. También se puede observar que no se comprobaron ciertas vulnerabilidades como Navegación Forzada o Modificación de parámetros que son vulnerabilidades presentes en el listado de OWASP Top 10 2013.

OWASP ZAP en su reporte logro identificar 4 del total de vulnerabilidades de la aplicación web. Al comparar el reporte de OWASP ZAP con el de Snort se puede verificar que hubo ataques hacia la vulnerabilidad de Salto de Directorio, que OWASP ZAP no reporto a pesar ser una vulnerabilidad presente en WackoPicko, también de este análisis se puede verificar que el escáner no

realiza una comprobación de vulnerabilidades como Inyección de Comandos de Sistema, Navegación Forzada o Modificación de parámetros que son vulnerabilidades presentes en el listado de OWASP Top 10 2013.

En el reporte del escáner HP WebInspect figuran 4 vulnerabilidades de las 12 presentes en WackoPicko. Al comparar el reporte de Snort, se determina que HP WebInspect realiza ataques a Inyección de Comandos de Sistema, pero no lo reporta a pesar de estar presente en la aplicación web. Este escáner no reporta la presencia de ninguna vulnerabilidad de Secuencia de Comandos en Sitios Cruzados de tipo Almacenado y únicamente señalo la existencia de Secuencia de Comandos en Sitios Cruzados XSS Reflejado. HP WebInspect a diferencia de Acunetix WVS y OWASP ZAP si comprueba y halla el fallo de Inclusión de Fichero Local, sin embargo no realiza una comprobación de vulnerabilidades como Navegación Forzada o Modificación de parámetros que son vulnerabilidades presentes en el listado de OWASP Top 10 2013.

Arachni es el escáner que menos vulnerabilidades reporta, un total de 3 vulnerabilidades fueron reportadas de todas las presentes en WackoPicko. Todas las vulnerabilidades halladas fueron atacadas y se lo puede verificar gracias al reporte de Snort. No hubo alertas de ataque hacia otras vulnerabilidades de la aplicación web que no fueran reportadas por el escáner. No realizo una verificación de las vulnerabilidades de Inclusión de Fichero Local, Navegación Forzada, Modificación de parámetros o Salto de Directorio que son vulnerabilidades presentes en el listado de OWASP Top 10 2013. No reporto vulnerabilidades del subtipo Almacenado de la vulnerabilidad Secuencia de Comandos en Sitios Cruzados XSS y señalo la existencia de Secuencia de Comandos en Sitios Cruzados XSS de forma general, es por eso que se combinaron las celdas en la tabla comparativa de esta herramienta.

6.4.1.1. Falsos Positivos

Los falsos positivos que los reportes de OWASP ZAP, HP WebInspect y Arachni se presentan en la Tabla 6.2.

Vulnerabilidad	OWASP ZAP	HP WebInspect	Arachni
XSS Reflejado	-	-	1
XSS Almacenado	2	-	
Inyección de SQL Almacenada	1	1	1
Inyección de SQL Reflejada	-	-	-
Salto de Directorio	-	1	-
Inyección de Comandos de Sistema	-	-	-
Inclusión de Fichero Local	-	-	-
Modificación de Parámetros	-	-	-
Error Lógico	-	-	-
Usuario/Contraseña Débil	-	-	-
Navegación Forzada	-	-	-
Fallo en la ID de Sesión	-	-	-
Total de Falsos Positivos	3	2	2

Tabla 6.2 Falsos positivos del análisis de WackoPicko.

En el reporte de Arachni se verifico que en las vulnerabilidades halladas de tipo Secuencia de Comandos en Sitios Cruzados XSS no se encontró ninguna del subtipo Almacenado, de igual forma ocurrió con las vulnerabilidades de tipo Inyección de SQL, el subtipo Almacenada no fue hallada. Sin embargo, presenta bajos falsos positivos, que es algo beneficioso, pero contrasta con la poca cantidad de vulnerabilidades halladas del total presentes en las aplicaciones web analizadas.

De estas tres herramientas, todas han fallado en el intento por determinar la vulnerabilidad de subtipo Almacenado de Secuencia de Comandos en Sitios Cruzados XSS y de Inyección de SQL, sin embargo HP WebInspect y OWASP ZAP mostraron mucha precisión en sus hallazgos debido a la relación de poca cantidad de falsos positivos con respecto al número total de vulnerabilidades halladas.

Vulnerabilidad	Aplicación	IDS	Acunetix WVS	IDS	OWASP ZAP	IDS	HP WebInspect	IDS	Arachni
RXSS(XSS Reflejado)	1	X	1	X	2	X	5	X	1
PXSS(XSS Persistente)	1		4		1		-		2
Inyección de SQL	2	X	4	X	1	X	3	X	-
Inyección de SQL a Ciegas	1	X	3	-	-	X	-	-	-
CSRF Falsificación de Petición en Sitios Cruzados	1	-	-	-	-	-	-	-	-
LFI Inclusión de Fichero Local	1	X	-	-	-	-	-	-	1
CMDExec Inyección de Comandos de Sistema	1	X	2	X	1	X	1	X	-
Salto de Directorio	-	X	1	X	1	X	11	X	1
DoS Denegación de Servicio	-	X	1	X	-	X	-	-	-
RFI Inclusión de Fichero Remoto	-	-	-	X	1	-	-	X	1

Tabla 6.3 Resultados de analizar la aplicación web DVWA

Vulnerabilidad	Aplicación	IDS	Acunetix WVS	IDS	OWASP ZAP	IDS	HP WebInspect	IDS	Arachni
XSS Reflejado	3	X	5	X	6	X	5	X	7
XSS Almacenado	2		1		2		-		
Inyección de SQL Almacenada	1	X	1	X	2	X	1	X	1
Inyección de SQL Reflejada	1								
Salto de Directorio	1	X	-	X	-	X	1	-	-
Inyección de Comandos de Sistema	1	X	2	-	-	X	-	X	1
Inclusión de Fichero Local	1	-	2	X	1	X	1	-	-
Modificación de Parámetros	1	-	-	-	-	-	-	-	-
Error Lógico	1	-	-	-	-	-	-	-	-
Usuario/Contraseña Débil	1	-	1	-	-	-	-	-	-
Navegación Forzada	1	-	-	-	-	-	-	-	-
Fallo en la ID de Sesión	1	-	-	-	-	-	-	-	-

Tabla 6.4 Resultados de analizar aplicación web WackoPicko

7. CONCLUSIONES Y TRABAJO FUTURO

7.1. Conclusiones

El modelo propuesto en este trabajo permite obtener detalles sobre el comportamiento no deseado de las herramientas de análisis de vulnerabilidades que no se tuvieron en cuenta en trabajos anteriores. Se detectaron deficiencias en las herramientas de análisis dinámico, utilizadas en este trabajo, para identificar vulnerabilidades en las aplicaciones web vulnerables WackoPicko y DVWA.

En la mayoría de las herramientas de análisis de vulnerabilidades utilizadas en este trabajo se ejecutaron ataques que no fueron reportados como tales en sus informes finales, a pesar de su existencia en la aplicación web. Sin embargo, estas vulnerabilidades fueron confirmadas por el IDS. Estos fallos deben tenerse en cuenta con el objeto de corregirlos para disminuir las incidencias de estos casos.

Asimismo, en los reportes del IDS se observó que hubo herramientas que no realizaron ataques en busca de algunas vulnerabilidades descritas en OWASP TOP 10 2013 y que este tipo de herramientas de análisis dicen cubrir [29][30][31][32]. Esto puede ser considerado como una desventaja, dado que no identifica completamente las vulnerabilidades más importantes y documentadas, aumentando de esta manera el riesgo de que la aplicación analizada sea atacada en algún momento.

A pesar que Snort cuenta con reglas para un amplio espectro de posibles ataques en su reporte generaliza el tipo de ataque. Este reporte no es tan detallado como el generado por las herramientas de análisis de vulnerabilidades utilizadas, que sí las clasifica. Es importante desarrollar nuevas reglas en Snort que permitan determinar específicamente el tipo de ataque para no generalizar resultados.

Si se configuran correctamente los parámetros de entrada y las características particulares de la aplicación en el escáner antes del análisis, se puede mejorar la precisión de los resultados finales del análisis.

Puede reducirse la cantidad de vulnerabilidades explotadas, pero no reportadas y al mismo tiempo el número de falsos positivos. Los reportes de cada herramienta contienen información básica y ayuda al desarrollador a identificar las fallas en la aplicación lo cual permite mejorar la seguridad de la aplicación en el ciclo de desarrollo de la misma y esto puede reducir tiempo y costos de mantenimiento posterior a la puesta en producción de la aplicación. El tiempo de ejecución de cualquier herramienta de análisis de tipo caja negra es mucho menor al tiempo que tomaría analizar la aplicación de forma manual.

7.2. Trabajos Futuros

Entre las líneas de trabajo futuro que se desprenden, están las siguientes:

- Incluir reglas en Snort que permitan determinar específicamente el tipo de ataque y que no generalice en sus resultados.

- Debido a los resultados obtenidos, se considera importante realizar más análisis sobre distintas herramientas de análisis dinámico utilizando aplicaciones vulnerables que contengan características más cercanas a las de una aplicación web real, como el caso de WackoPicko.

- Realizar pruebas sobre la capacidad que poseen las herramientas para localizar vulnerabilidades exclusivamente de tipo Persistente, ya sea de inyecciones de SQL o de *XSS*, debido a que el número de aciertos o incluso falsos positivos de este tipo de vulnerabilidades, por parte de los escáneres es muy bajo, casi inexistente.

RESUMEN EN INGLÉS

8. INTRODUCTION

The software is present in almost all of our daily activities, even without notice that are we using it. The web applications also are presents in these activities, for example in the management of a bank account through the web site of the bank or paying the invoices of basic services and even in the way that we interact with other people through social networks. In general, these applications stored and manage many kind of data, personal data even confidential and sensitive data and that is why those applications must be secure to protected all this data. In the 2015 has been reported 5334 attacks to web applications, of this cases 908 incidents been confirmed the disclosure of the data managed by these attacked applications. The use of tools and good practice of security to secure in all the stages of the development and production of the application reduces the probability of success of an attack. [2][3].

In the market there are many tools that streamline the review and correction of faults and vulnerabilities of an application. Choosing the right tool is a very important task, since the capacity of correct any vulnerability in an application depends on the tool effectiveness. These tools, according to their capabilities and features, can be white box ("*White-box Testing Tools*") or black box ("*Black-box Testing Tools*").

White box analysis [3][4] consists in analyze the source code and structure of the application to find fault that could open security breaches in the application at the production stage, becoming into a vulnerable application. Code analysis can be done by using specific tools or also manually [5]. The main disadvantage that exists on this type of testing is the time it takes to perform the analysis the source code of the application.

In the black-box type analysis [3][4][6], the security of an application is analyzed in a functional way without analyzing the source code, just using

automated tools who have the mission to find potential vulnerabilities in the application, with the realization of penetration attacks to the application.

The use of dynamic analysis tools for web applications reduces time and effort in the development cycle and allows to focus greater efforts on more complex safety tasks [7].

Such tools are not easy to set up for those who are not familiar with them [5]. One of its biggest weaknesses is the presence of false positive results [7], so it is important to have knowledge of the strengths and weaknesses that these tools present.

8.1. Motivation

Currently in the market there are different kind of vulnerabilities scanners for web applications. However, none of these tools is one hundred percent effective. Many of these tools have a high percentage of false positive results.

Existing studies in the literature have focused on mainly analyze the accuracy of scanners and maintaining the same basic method for evaluation. None has gone further, seeking to observe the behavior analysis tools have vulnerabilities with applications that analyze and look around failures that result in the inaccuracy of their reports. It means that is not analyzed the what of the tool to get the report presented, but just compare the report results and then obtain a criteria on base of this.

This paper considers important to know details about the behavior of the scanning tool during the analysis of a vulnerable web application, the conducted tests, the failures and the successes of the scanner, among other things, to make a full assessment of the capabilities and shortcomings of these tools.

8.2. Purpose of Investigation

In this paper are use a different approach that the used in previous work, to obtain a clear landscape on how the dynamic analysis tools work and its precision. It is introduced, in the typical structure of study of the literature, an Intrusion Detection System (IDS), who allow to obtain all the requests of attack launched by each tool on its analysis and compare it with the corresponding report. In this way obtain valuable information to define the efficiency of these tools.

The specific objectives of this work are:

- Propose a functional model to get more information on the results of vulnerability scans performed on vulnerable web applications with dynamic analysis tools.
- Perform an automatic analysis of traffic generated by the web vulnerabilities scanner to speed up the evaluation process and reduce the risk of failure, detected in previous studies by performing a process of manual analysis.
- Find the possible tests carried out and also those that omits a scanner of web vulnerabilities during its execution.

8.3. Structure of this Paper

The rest of the paper is organized in 6 more chapters with the following structure:

Chapter 2 shows general basic concepts and the importance of security in application development. Also, the detailed classification of the most important vulnerabilities.

Chapter 3 discusses the two main categories in which analysis tools of vulnerabilities are classified, and details the stages of execution of these tools. Also is disclosed the main features of the tools used during the experiments conducted in this paper.

In the Chapter 4 present a review of the most important works in the literature who are related with this paper.

Chapter 5 details the methodology proposed in this paper, describing each of the components used in the experiments and settings.

Chapter 6 describes the results obtained from the experiments to test the operation of the proposed methodology, configuration tools and web applications.

Finally, in Chapter 7 the findings of this study are explained and possible lines of future work leading to enrich the mentioned research.

9. CONCLUSIONS AND FUTURE WORKS

9.1. Conclusions

The approach proposed in this work allowed obtain details about the erratic behavior in the analysis of vulnerabilities performed by these tools and that are not considered in previous work. This approach showed the deficiencies presented in the dynamic analysis tools, used in this paper, to identify vulnerabilities in WackoPicko and DVWA.

Most scanners used in this work showed that attacks were made and these were confirmed by Snort, but in the final report of the tool was not considered as an vulnerability, despite its existence in the web application. These failures should be taken into account by developers of these tools and try to correct it or at least lessen the impact of these. Additionally, Snort reports showed tools that were not carried out attacks in search at least of some vulnerabilities described in OWASP TOP 10 2013 and that this tools claim perform [29][30][31][32]. This can be considered as a disadvantage in this type of software, mainly because their aim of covering at least the most important and documented vulnerabilities were not performed completely, thereby increasing the risk that the application analyzed could be attacked at some point.

Although Snort had thousands of rules for a wide range of possible attacks in their report it failed to identify the subtype of attack. This report contrasts with that generated by the used scanners that are able to classify the subtype, in order to refine the results and be more accurate in comparison. It is important to develop new rules to allow Snort to specifically determine the type of attack and not generalize results.

The configuration previous to analysis can determine the accuracy and tool results if all input parameters and the particular characteristics of the application are correctly entered. You can reduce the amount of exploited

vulnerabilities but not reported by the scanner and also the number of false positives. The reports of each tool contain basic information and help the developer to identify failures in the application which helps improve application security within the development cycle and that reduces time and maintenance costs after the start up in the production environment. The runtime of any analysis tool black-box is much less than the time it would take to analyze the application manually.

9.2. Future Works

The lines of future work that emerges from this work are the following

- Include Snort rules that can specifically determine the type of attack and did not generalize their results.
- Because of the results obtained, it is considered important to conduct more analysis of this kind on different tools by using vulnerable applications that contain features closest to the presents in a real web application, as the case of WackoPicko.
- Make more tests over the ability of the scanners to find exclusively vulnerabilities of Persistent type, either SQL injection or XSS, because the number of hits or even false positives of such vulnerabilities by scanners is very low, almost nonexistent.

REFERENCIAS

[1] Verizon, "2016 Data Breach Investigations Report," Report, Verizon, July 2016.

[2] I. Sommerville, Ingeniería de software. Madrid: Pearson Educación de México, 2011.

[3] A. Sagala and E. Manurung, "Testing and Comparing Result Scanning Using Web Vulnerability Scanner," *Advanced Science Letters*, vol. 21, pp. 3458–3462, November 2015.

[4] S. Nidhra and J. Dondeti, "Blackbox and Whitebox Testing Techniquesa Literature Review," *International Journal of Embedded Systems and Applications*, vol. 2, pp. 29–50, June 2012.

[5] Y. Makino and V. Klyuev, "Evaluation of Web Vulnerability Scanners," in *Proceedings of the IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, vol. 1, (Warsaw, PL), pp. 399–402, XXXXX 2015.

[6] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, "State of the Art: Automated Blackbox Web Application Vulnerability Testing," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, (Washington, DC, USA), pp. 332–345, IEEE Computer Society, 2010.

[7] P. Baral, "Web Application Scanners: A Review of Related Articles," *IEEE Potentials*, vol. 30, pp. 10–14, March 2011.

[8] "Chapter 3: Application Software." http://uwf.edu/clemley/cgs1570w/notes/concepts-3.htm, August 2005.

[9] P. Milano, "Seguridad en el Ciclo de Vida del Desarrollo de Software," Release, CYBSEC Security Systems S.A., September 2007.

[10] SANS Institute, "2015 State of Application Security: Closing the Gap," Release, SANS Institute, May 2015.

[11] "Home Web Application Security Consortium." http://www.webappsec.org, April 2016.

[12] J. Dooley, Software Development and Professional Practice. Books for Professionals by Professionals, Apress, 2011.

[13] "MVC Architecture." https://developer.mozilla.org/en-US/Apps/Fundamentals/Modern_web_app_architecture/MVC_architecture, April 2016.

[14] P. D. Gallagher, "Guide for Conducting Risk Assessments," National Institute of Standards and Technology, Acunetix, September 2012.

[15] International Organization for Standardization ISO, "ISO/IEC 27001:2013," Release, International Organization for Standardization ISO, 2013.

[16] A. L. H. J. Mañas S. L., "PILAR Análisis y Gestión de Riesgos, Glosario de Términos." http://www.ar-tools.com/es/glossary/index.html, March 2011.

[17] F. Román Muñoz and L. J. García Villalba, "Algoritmo para el Mapeo de Clasificaciones de Vulnerabilidades Web," in *Libro de Actas del Congreso VIII Iberoamericano de Seguridad Informática*, (Quito, Ecuador), 2015.

[18] Web Application Security Consortium, "WASC Threat Classification," Release, Web Application Security Consortium, January 2010.

[19] F. Román Muñoz and L. J. García Villalba, "Búsqueda de Relaciones entre Vulnerabilidades de Aplicaciones Web," in *Libro de Actas del Congreso VIII Iberoamericano de Seguridad Informática*, (Quito, Ecuador), 2015.

[20] "About the Open Web Application Security Project." https://www.owasp.org/index.php/OWASP:About, May 2016.

[21] The Open Web Application Security Project OWASP, "OWASP Top 10 2013 The Ten Most Critical Web Application Security Risks," Release, The Open Web Application Security Project OWASP, June 2013.

[22] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. Le Traon, "Comparing Whitebox and Blackbox Test Prioritization," in *Proceedings of the 38th International Conference on Software Engineering*, (Austin, Texas, USA), pp. 523–534, 2016.

[23] E. Fong and V. Okun, "Web Application Scanners: Definitions and Functions," in *Proceedings of 40th Annual Hawaii International Conference on System Sciences, 2007.*, pp. 280b–280b, IEEE, 2007.

[24] M. E. Khan, "Different Forms of Software Testing Techniques for Finding Errors," *International Journal of Computer Science Issues*, vol. 7, no. 3, pp. 11–16, 2010.

[25] A. Austin and L. Williams, "One Technique is Not Enough: A comparison of Vulnerability Discovery Techniques," in *Proceedings of the International Symposium on Empirical Software Engineering and Measurement* 2011, pp. 97–106, Sept 2011.

[25] M. Nouman, U. Pervez, O. Hasan, and K. Saghar, "Software Testing: A Survey and Tutorial on White and Blackbox Testing of C/C++ Programs," in *Proceedings of the IEEE Region 10 Symposium*, 2016, pp. 225–230, May 2016.

[26] F. R. Muñoz and L. G. Villalba, "Preproceso de Formularios para el Análisis de Seguridad de las Aplicaciones Web," in *Proceedings of the XII Reunión Española sobre Criptología y Seguridad de la Información*, 2012.

[27] M. Parvez, P. Zavarsky, and N. Khoury, "Analysis of Effectiveness of Blackbox Web Application Scanners in Detection of Stored SQL Injection and Stored XSS Vulnerabilities," in *Proceedings of the 10th International Conference for Internet Technology and Secured Transactions* 2015, pp. 186–191, Dec 2015.

[28] "OWASP Zed Attack Proxy Project," April 2016.

[29] "ARACHNI Web Application Security Scanner Framework." http://www.arachni-scanner.com, March 2016.

[30] Acunetix, "Web Vulnerability Scanner v10 Product Manual," Product Manual, Acunetix, June 2015.

[31] HP, "HP WebInsPect," Product Manual, HP, March 2015.

[32] F. A. Saeed, "Using WASSEC to Evaluate Commercial Web Application Security Scanners," *International Journal of Soft Computing and Engineering*, vol. 4, pp. 177–181, March 2014.

[33] F. A. Saeed, "Using WASSEC to Analysis and Evaluate Open Source Web Application Security Scanners," *International Journal of Computer Science and Network*, vol. 3, pp. 43–49, April 2014.

[34] N. Khoury, P. Zavarsky, D. Lindskog, and R. Ruhl, "Testing and Assessing Web Vulnerability Scanners for Persistent SQL Injection Attacks," in *Proceedings of the First International Workshop on Security and Privacy Preserving in eSocieties*, New York, NY, USA, pp. 12–18, ACM, 2011.

[35] N. Suteva, D. Zlatkovski, and A. Mileva, "Evaluation and Testing of Several Free/Open Source Web Vulnerability Scanners," in *Proceedings of the 10th Conference for Informatics and Information Technology*, (Bitola, MK), pp. 221–224, April 2013.

[36] N. I. Daud, K. A. A. Bakar, and M. S. M. Hasan, "A Case Study on Web Application Vulnerability Scanning Tools," in *Proceedings of the Conference of Science and Information*, pp. 595–600, IEEE, 2014.

[37] A. Doupe, M. Cova, and G. Vigna, "Detection of Intrusions and Malware, and Vulnerability Assessment," in *Proceedings of the 7th International Conference*, Bonn, Germany, pp. 111–131, July 2010.

[38] H. Alnabulsi, M. R. Islam, and Q. Mamun, "Detecting SQL Injection Attacks Using Snort IDS," in *Proceedings of the 2014 AsiaPacific World Congress on Computer Science and Engineering*, pp. 1–7, IEEE, Nov 2014.

[39] M. Dabbour, I. Alsmadi, and E. Alsukhni, "Efficient Assessment and Evaluation for Websites Vulnerabilities Using Snort," *International Journal of Security and its Applications*, vol. 7, no. 1, 2013.

[40] "Damn Vulnerable Web Application (DVWA)." http://www.dvwa.co.uk, March 2016.

[41] "WackoPicko Vulnerable Website." https://github.com/adamdoupe/WackoPicko, March 2016.

[42] "Snort Network Intrusion Detection & Prevention System." https://www.snort.org, July 2016.

[43] "Darkstat." https://unix4lyfe.org/darkstat/, May 2015.

[44] "Remote Code Vulnerability in Spring framework for Java." http://www.infosecurity-magazine.com/news/remote-code-vulnerability-in-spring-framework-for, January 2013.

[45] M. Roesch, "Snort Lightweight Intrusion Detection for Networks," in Proceedings of the 13th USENIX Conference on System Administration, LISA '99, (Berkeley, CA, USA), pp. 229–238, USENIX Association, 1999.

ANEXO A: EJEMPLO DE VULNERABILIDADES DE OWASP TOP 10 2013

Inyección.

- **Dificultad de Explotación**: Baja.
- **Dificultad de Detección**: Media.
- **Impacto**: Alto.

Ejemplo de Inyección de SQL [21]: En el caso de una aplicación que realiza una consulta SQL que no verifica los datos que utiliza.

```
String query = "SELECT * FROM accounts WHERE custID='"
        + request.getParameter("id") + "'";
```

El atacante puede modificar el valor del parámetro *id* en la URL de su navegador y realizar una petición similar a:

```
http://vulnerable.com/app/accountView?id=' or '1'
```

Esta sencilla petición permite al atacante obtener los datos que se encuentran en la tabla *account*. El atacante no solo podría mirar los datos sino también modificarlos o ejecutar procedimientos almacenados.

Fallo en autenticación y gestión de sesiones.

- **Dificultad de Explotación**: Media.
- **Dificultad de Detección**: Media.
- **Impacto**: Alto.

Ejemplo [21]**:** En una aplicación que soporta la reescritura de la URL, un usuario copia la dirección URL, desde su navegador, de una oferta y la envía

por email a un amigo suyo. El enlace tiene esta estructura:

```
http://vulnerable.com/promocion/ofertaArt;jsessionid=2P0O
      C2JSNDLPSKHCJUN2JV?parametro=cualquiercosa
```

Sin saberlo, el usuario está enviando en el enlace su identificador único de sesión y la persona que acceda a la aplicación a través de ese enlace lo hará utilizando la sesión del remitente original, pudiendo tener acceso a todos los datos y configuraciones de su cuenta.

Secuencia de Comandos en Sitios Cruzados.

- **Dificultad de Explotación**: Media.
- **Dificultad de Detección**: Baja.
- **Impacto**: Medio.

Ejemplo [21]: Una aplicación utiliza en su código HTML datos sin validar:

```
      (String) page += "<input name='tarjetacredito'
type='TEXT' value='" + request.getParameter("TC") + "'>";
```

Un atacante puede modificar desde su navegador el parámetro 'TC' de esta forma:

```
'><script>document.location='http://www.malicioso.com/cgi
    -bin/cookie.cgi?foo='+document.cookie</script>'
```

Este *script* permite que el identificador de la sesión de la víctima sea enviada a la página web del atacante y este a su vez puede hacerse de ella.

Referencia Insegura Directa de Objetos.

- **Dificultad de Explotación**: Baja.
- **Dificultad de Detección**: Baja.

- **Impacto**: Medio.

Ejemplo [21]: Una aplicación que utiliza el siguiente código para acceder hacia los datos almacenados de una determinada cuenta:

```
String sqlquery = "SELECT * FROM cuentas WHERE cuenta = ?";
PreparedStatement st = connection.prepareStatement(sqlquery
                                , ...);
     st.setString( 1, request.getParameter("cuenta"));

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

El atacante puede realizar una petición desde su navegador modificando el valor del parámetro *'cuentat'*:

```
http://vulnerable.com/ventas/infoCuenta?cuenta=cualquiera
```

Si el parámetro *'cuenta'* no es verificado, el atacante puede acceder a los datos de cualquier cuenta de usuario que desee.

Configuraciones de Seguridad mal implementadas.

- **Dificultad de Explotación**: Baja.
- **Dificultad de Detección**: Baja.
- **Impacto**: Medio.

Ejemplo [21]: Cuentas de usuarios por defecto que no son desactivadas, consolas de administración instaladas por defecto y de acceso público, datos de ejemplo no borrados, listado de directorio del servidor no deshabilitado.

Exposición de datos sensibles.

- **Dificultad de Explotación**: Alta.
- **Dificultad de Detección**: Media.
- **Impacto**: Alto.

Ejemplo [21]: Las aplicaciones que hacen uso de certificados SSL para las páginas de autenticación tienen el riesgo de que un atacante pueda monitorear el tráfico de red y hacerse de las *cookie's* de usuarios para suplantar su identidad. El cifrado de claves y datos sensibles de forma automática en la base de datos implica que esta información se descifre automáticamente por la aplicación web al recibirse en el navegador para mostrarse y si un atacante utiliza una inyección de SQL puede acceder a los datos en texto plano.

Ausencia de control de acceso a funcionalidades.

- **Dificultad de Explotación**: Baja.
- **Dificultad de Detección**: Media.
- **Impacto**: Medio.

Ejemplo [21]: En el caso de que un atacante pueda acceder a páginas de administración de la aplicación simplemente forzando el acceso desde la dirección del navegador. Si un usuario autenticado pero sin los respectivos privilegios para realizar una acción dentro de la aplicación y puede acceder a la pagina que permite realizar esta acción simplemente forzando la dirección en el navegador:

```
http://vulnerable.com/admin/administracion.php
```

Falsificación de Petición en Sitios Cruzados CSRF.

- **Dificultad de Explotación**: Media.
- **Dificultad de Detección**: Baja.
- **Impacto**: Medio.

Ejemplo [21]: Una aplicación web envía una solicitud dentro de la cual no se incluye ningún parámetro de validación y esta solicitud ejecuta un cambio de estado en el sistema:

```
http://vulnerable.com/admin/enviarDinero?cantidad=1234&cu
entaDestino=4321
```

Un atacante puede entonces modificar la consulta y realizar cambios de estado a conveniencia, embebiendo la consulta dentro de cualquier objeto de una página web maliciosa. Si la victima visita el sitio malicioso mientras se encuentra autenticado en una aplicación web vulnerable, el atacante puede ejecutar la petición utilizando las credenciales de la víctima sin problemas:

```
<img
src="http://vulnerable.com/admin/enviarDinero?cantidad=1234
&cuentaDestino=#Atacante‟ width="0" height="0" />
```

Uso de componentes vulnerables.

- **Dificultad de Explotación**: Media.
- **Dificultad de Detección**: Alta.
- **Impacto**: Medio.

Ejemplo [21]: El uso de componentes que contienen vulnerabilidades y que son utilizados en el desarrollo o despliegue de una aplicación web es común y debido a que estos componentes se ejecutan dentro de la aplicación web con casi todos los privilegios de acceso internos, sus vulnerabilidades afectan a la aplicación web que las usa de forma directa.

- Ejecución remota de código, en 2011 la infraestructura digital Spring para Java contenía una vulnerabilidad que permitía a un atacante inyectar y ejecutar código malicioso a aplicaciones que utilizaban esta versión de Spring. Esta vulnerabilidad fue hallada y subsanada por parte del fabricante, pero para el tiempo en que fue descubierto la versión vulnerable de Spring había sido ya descargada más de 1.3 millones de ocasiones y utilizada por más de 22 mil organizaciones en

todo el mundo.[45]

Redirección y reenvío sin validación.

- **Dificultad de Explotación**: Media.
- **Dificultad de Detección**: Baja.
- **Impacto**: Medio.

Ejemplo [21]: En una aplicación vulnerable se realiza la redirección mediante el uso de un parámetro '*url*' que no es verificado antes de realizarse. A través de esto un atacante puede modificar el parámetro de redirección utilizado por la aplicación web y redirigir a la víctima hacia un sitio malicioso.

```
http://vulnerable.com/redirect.php?url=malicioso.com
```

ANEXO B: SNORT

Snort es un sistema de detección de intrusos de red basado en reglas que fue desarrollado en 1998 por Martin Roesch [46]. Snort es gratuito y de código abierto, desde su creación ha ido mejorando y perfeccionándose.

La arquitectura de funcionamiento de Snort está formada por cinco módulos principales que se muestran a continuación:

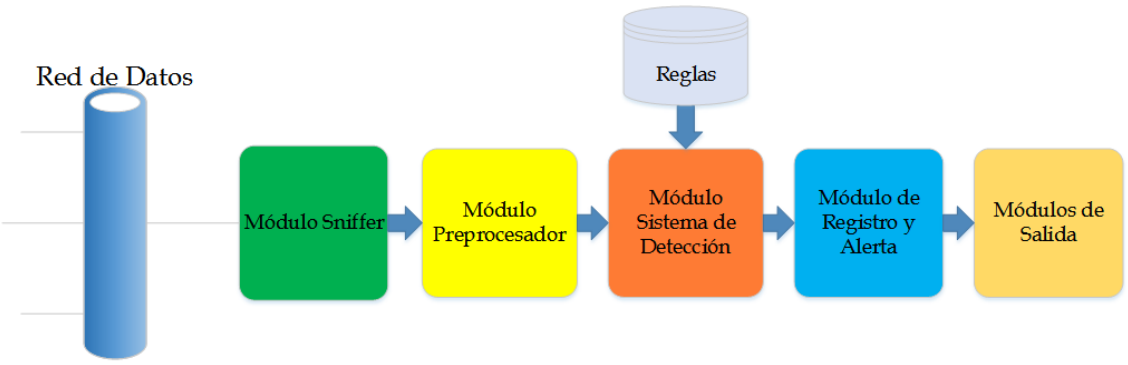

Fig. 0.1 Esquema de Arquitectura de Snort

Módulo Sniffer: Captura los paquetes de red y se encarga de prepararlos para ser preprocesador

Modulo Preprocesador: captura la carga útil del paquete y lo compara para detectar anomalías y de acuerdo a ello enviarlo al sistema de detección y ser procesado.

Modulo Sistema de detección: Toma los paquetes provenientes del preprocesador y extrae los datos para compararlos con el grupo de reglas establecidas en la configuración de Snort.

Modulo Registro y Alertas: De acuerdo a qué ha detectado el sistema de detección, el módulo de alertas y registro se encarga de generar un registro de actividad y alertas en donde todos los paquetes capturados que han coincidido con alguna regla de Snort son guardados. El fichero de registro se guarda en

texto simple

Módulos de salida: A través de este módulo se puede configurar a Snort para que permita guardar el registro de actividad y alertas en distintos formatos y en función del módulo utilizado, por ejemplo:

- Envío de *traps* SNMP
- Envío de mensajes a un SYSLOG
- Almacenamiento del registro y alertas en base de datos
- Generación de fichero XML con el registro de actividad y alertas