Universitat Oberta de Catalunya

uoc.edu

Los sistemas de seguridad perimetral y principales vectores de ataque web (II)

Roberto Carlos Pérez González

Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones
TFM-Ad hoc

Richard Rivera Guevara

4 de Junio de 2018

LICENCIAMIENTO:

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

DEDICATORIA Y AGRADECIMIENTOS:

Quiero comenzar agradeciendo el proyecto a Richard Rivera Guevara, cómo tutor del proyecto me ha ido ayudando a corregir y llevar a buen puerto gran parte del proyecto.

También, quiero agradecer a mi mujer (Beatriz Carlos Sanchez) todo el apoyo y paciencia que ha tenido conmigo mientras iba avanzando con el master durante todos los fines de semanas. Ya ni siquiera recuerdo la cantidad de noches que dejé a mi mujer durmiendo sola mientras programaba o redactaba memorias noche tras noche. O incluso, cuantos fines de semanas nos quedamos en casa para poder avanzar este proyecto.

En último lugar agradeceré y dedicaré este proyecto a mis padres, amigos y mis compañeros de trabajo. Los primeros siempre me han apoyado durante toda mi vida. Los segundos, en concreto Adrián Exposito Barcoj ha tenido que aguantar gran cantidad de mis miserias y una persona especial, Miguel Panizo Laiz, que me ayudó con todas las matemáticas y físicas de la carrera lo que hizo que este master fuese posible en segunda instancia. Y los últimos, (mis compañeros de trabajo), que siempre me apoyaron y ayudaron cada vez que una PAC se quedaba atascada.

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Los sistemas de seguridad perimetral y principales vectores de ataque web*
Nombre del autor:	*Roberto Carlos Pérez González*
Nombre del consultor/a:	*Richard Rivera Guevara*
Nombre del PRA:	*Richard Rivera Guevara*
Fecha de entrega (mm/aaaa):	06/2018
Titulación:	*Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones*
Área del Trabajo Final:	*M1.830 - TFM-Ad hoc*
Idioma del trabajo:	*Castellano*
Palabras clave	*Detección de ataques web, Seguridad perimetral, monitorización, informática*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

El presente proyecto tiene como objetivo el diseño de una herramienta de análisis de trazas de registros de diferentes servidores webs que se podrá utilizar con fines forenses. Para ello se ha generado un entorno virtual y se ha codificado la herramienta sobre lenguaje python.

En cuanto al entorno virtual se ha gerado con dos máquinas, la primera de ellas es un equipo vulnerable montado sobre un entorno "LAMP" (tecnologías Linux, apache, MySQL y PHP). La segunda de ellas es un correlador de eventos de seguridad basado en tecnología OSSIM / AlienVautl que se utilizará principalmente cómo visualizador de eventos.

Adicionalmente se ha generado el motor de detección codificado en lenguaje "Python", las firmas de dicho motor se codifican de forma estructura con XML. Estas se basan en expresiones regulares en Python que detectan los vectores de intrusión web más comunes.

Finalmente, se ha lanzado una batería de test de intrusión para poner a prueba la herramienta diseñada, evaluando de este modo la calidad de esta y presentando unas conclusiones finales sobre su funcionamiento.

Abstract (in English, 250 words or less):

The main objective of this project is to design an analysis tool that we can use to apply forensic techniques over different web servers. With this objective, a virtual environment has been generated and the tool has been coded into the Python language.

The virtual environment has been generated with two machines. The first one is a web server on "LAMP" technology (Linux, apache, MySQL and PHP), and the second one is a security information event management tool based on technology AlienVault / Ossim.

Additionally, the detection engine has been encoded within the "Python" language. The signatures of this engine are in the structured language "XML". The signature set contains the most common intrusion patterns programmed with regular Python expressions.

Finally, an intrusion test battery has been done with the objective of evaluating the detection engine. It has introduced the results of the entire job and has presented some final conclusions about the way it functions.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

El contexto en el que se encarma este proyecto es el mismo en que se ha trabajado en proyectos anteriores [9]. El desbordanete crecimiento de la red de redes (Internet) está consolidando un claro hecho: cada día se utilizan más los servicios que ofrece esta red para las cosas más cotidianas. Este tipo de servicios se soportan sobre los entornos webs que ofrecen las empresas. Dichas webs contienen gran cantidad de datos de usuarios y suponen un punto de entrada muy importante de la facturación de muchas empresas. Este hecho hace que cada vez sea más importante llevar a cabo una securización de los sistemas de tecnologías de la información y comunicación de las diferentes empresas que ofertan estos servicios.

La evolución de los entornos web, dónde el intercambio de datos y bienes a través de medios telemáticos es totalmente habitual, se ha convertido en un medio para intentar conseguir recursos de manera fraudulenta. De este modo han aparecido diferentes vectores de intrusión realizados sobre estos medios virtuales. Entre ellos cabe destacar los siguientes:

- Ataques Web, aprovechando fallos de programación o vulnerabilidades en servidores web.
- Intentos de intrusión aprovechando vulnerabilidades conocidas en los softwares base para realizar accesos no autorizados.
- Campañas de Phising.
- Campañas de Malware.
- Ingeniería social.
- Fugas de información.

Con estas herramientas, los diferentes entes maliciosos intentan aprovecharse de los recursos de las empresas para beneficio propio o de mafias. Debido a ello, nace el campo de la seguridad informática. Uno de los ámbitos más habituales de la seguridad informática es la seguridad perimetral, en este ámbito de la seguridad se introducen diferentes mecanismos con el fin de limitar y controlar los accesos e invocaciones remotas. Por otro lado, nace la monitorización de seguridad (otro ámbito de seguridad), dónde se implementan diferentes medios para detectar y bloquear los principales intentos de vulneración e intrusión de sistemas.

Concluyendo, este proyecto se realiza como una continuación del trabajo realizado anteriormente por el autor [9], en el cual se analizó la efectividad de la seguridad perimetral que implementan las diferentes empresas y organismos públicos para proteger sus entornos webs. En esta continuación del trabajo se pretende automatizar la detección de patrones que son capaces de atravesar las medidas de seguridad perimetral. Para ello se ha generado un pequeño proceso que es capaz de detectar diferentes intentos de intrusión web en los servidores finales.

1.2 Objetivos del Trabajo

El presente proyecto tiene como objetivo principal el realizar una pequeña aplicación que ayude a la detección de diferentes tipos de ataques web. Esto se realizará a través del análisis de las trazas que dejan estos en los equipos finales. Con este objetivo se creará un entorno de prueba virtualizado, en este se aplicarán distintos vectores de ataques mediante tests de intrusión y se medirá la tasa de detección de la solución desarrollada.

Con el fin de realizar las pruebas se ha decidido implementar un entorno web clásico. Esto se hará mediante el despliegue de distintos aplicativos de software libre en un entorno virtualizado y controlado de maqueta sobre sistema operativo Linux. Entre las tecnologías que se implementarán en nuestro sistema "LAMP" encontramos:

- Sistema operativo Linux (debian 8).
- Un servidor web (apache).
- Una base de datos (mariaDB).
- PHP (cómo lenguaje de programación web).

Por otro lado, el desarrollo de la aplicación de detección usará principalmente los siguientes lenguajes de programación:

- Python (versión 2.7).
- XML.

Además utilizará distintas librerías para aplicar filtros de expresiones regulares sobre las trazas que dejan los diferentes aplicativos. Las expresiones regulares que se utilizarán para detectar los ataques son tanto de desarrollo propio cómo reutilizadas de algunos productos de software libre.

Así mismo, el obejtivo final del proyecto es obtener una aplicación que sea capaz de ayudar a los equipos de BlueTeam tanto a detectar intrusiones (más o menos en tiempo real), cómo a facilitar la realización de labores de análisis forenses posteriores a las intrusiones. Por tanto, con la entrega de este proyecto se hace una contribución a la comunidad. El código se ha liberado y puesto a disposición de esta en los enlaces de GitHub que pueden encontrarse en el anexo del proyecto "10.1 Código fuente "Morbot – Necromancer"."

1.3 Enfoque y método seguido

El proyecto se enfocará principalmente en tres fases. En la primera de ellas se implementará el entorno vulnerable en PHP sobre el sistema Linux con Apache y MariaDB. En el segundo, se desarrollará la aplicación de análisis y detección desarrollada en el lenguaje de programación de "Python". En la última fase se pondrá a prueba el sistema establecido y se documentará todas las pruebas realizadas con los resultados obtenidos.

Para la primera fase existen ya máquinas virtuales que podrían realizar más o menos la función necesaria. Todas estas versiones han sido descartadas ya

que al contrario que los sistemas reales, estas apenas guardan trazabilidad de las peticiones que reciben. Por tanto, al tratarse de un ejercicio práctico dónde el entorno debe guardar gran trazabilidad, se ha decidido utilizar un enfoque dónde toda la plataforma sea de desarrollo propio.

En cuanto a la segunda fase, se ha decidido seguir un enfoque de desarrollo dónde se divide la aplicación principalmente en dos secciones:

1. Sección de código: se trata del aplicativo base o motor diseñado en Python que aplicará las diferentes expresiones regulares sobre las trazas de log.
2. Sección de patrones: se basa en varios ficheros de firmas estructurados en XML que contienen los nombres de las firmas, así cómo las expresiones regulares de los principales vectores de ataques.

En último lugar, para la fase de pruebas se utilizará un enfoque de pruebas basado en metodología OWASP [3]. De este modo iremos probando todos los tipos de vectores de intrusión de forma ordenada y metódica para poder evaluar la fiabilidad de detección de nuestra herramienta. Los intentos de intrusión se realizarán principalmente de dos formas:

1. Realización de pentesting manual mediante inyección controlada de diferentes payload y fragmentos de código.
2. Realización de pentesting con herramientas automáticas como son “nmap”, “openvas”, “sqlmap”, etc.

Por tanto, el enfoque general será el establecer un sistema vulnerable monitorizado por un sistema de desarrollo propio el cual se pondrá a prueba mediante inyecciones controladas y barridos con herramientas.

1.4 Planificación del Trabajo

La planificación del trabajo se ha realizado principalmente en tres fases durante las cuales se desarrollan distintas actividades secuencialmente. A continuación, se detalla el contenido de cada fase:

Fase I: Desarrollo del entorno de pruebas (PEC 2).

- **Inicio**: 13/03/2018, fin: 09/04/2018.
- **Detalle**: se instalará una máquina virtual basada en tecnología “LAMP” sobre un entorno de vMware Workstation. Una vez instalado se procederá al desarrollo del aplicativo web vulnerable con su correspondiente base de datos. Una vez finalizado el conjunto se procederá a testear el correcto funcionamiento.
- **Horas estimadas:** 100
- **Actividades:**
  I. Instalación del sistema operativo LINUX
  II. Instalación de los aplicativos web y BBDD
  III. Desarrollo del entorno web en PHP
  IV. Pruebas de funcionamiento

Fase II: Diseño y desarrollo de la aplicación de detección y conjunto de firmas (PEC 3).

- **Inicio**: 10/04/2018, fin: 07/05/2018.
- **Detalle**: se procederá a diseñar y desarrollar la aplicación de detección así cómo el conjunto de firmas que se utilizarán para detectar posteriormente los diferentes vectores de intrusión.
- **Horas estimadas**: 100
- **Actividades:**
  I. Realización del diseño del aplicativo software
  II. Implementación del software en Python
  III. Pruebas de funcionamiento del software
  IV. Diseño e implementación de firmas de ataques
  V. Pruebas de funcionamiento del aplicativo general

Fase III: Realización y documentación de pruebas sobre el entorno y realización de memoria (PEC 4).

- **Inicio**: 08/05/2018, fin: 04/06/2018.
- **Detalle**: se realizará la ejecución de los diferentes intentos de intrusión, tanto con escaneres cómo con ejecuciones manuales. Posteriormente se documentará todo y se realizará la memoria final del trabajo fin de master.
- **Horas estimadas**: 100
- **Actividades**:
  I. Ejecución de vectores de intrusión
  II. Documentación de los resultados
  III. Realización de la memoria
  IV. Realización y preparación de la presentación

A continuación, se adjunta la planificación detallada de las actividades de cada fase y diagrama de tiempo correspondiente.

Detalle de las diferentes fases:

Name	Begin date	End date
Fase I: Desarrollo del entorno de pruebas (PEC 2).	3/13/18	4/9/18
Fase I.I: Instalación del sistema operativo LINUX	3/13/18	3/15/18
Fase I.II: Instalación de los aplicativos web y bbdd	3/16/18	3/19/18
Fase I.III: Desarrollo del entorno web en PHP	3/20/18	4/6/18
Fase I.IV: Pruebas de funcionamiento	4/9/18	4/9/18
Fase II: Diseño y desarrollo de la aplicación de detección y conjunto de firmas (PEC 3).	4/10/18	5/7/18
Fase II.I: Realización del diseño del aplicativo software	4/10/18	4/11/18
Fase II.II: Implementación del software en Python	4/12/18	4/27/18
Fase II.III: Pruebas de funcionamiento del software	4/30/18	4/30/18
Fase II.IV: Diseño e implementación de firmas de ataques.	5/1/18	5/4/18
Fase II.V: Pruebas de funcionamiento del aplicativo general	5/7/18	5/7/18
Fase III: Realización y documentación de pruebas sobre el entorno y realización de memoria (PEC 4).	5/8/18	6/4/18
Fase III.I: Ejecución de vectores de intrusión.	5/8/18	5/10/18
Fase III.II: Documentación de los resultados.	5/11/18	5/14/18
Fase III.III: Realización de la memoria.	5/15/18	6/4/18

Figura 1: detalle de las fases del proyecto.

Planificación temporal del proyecto:

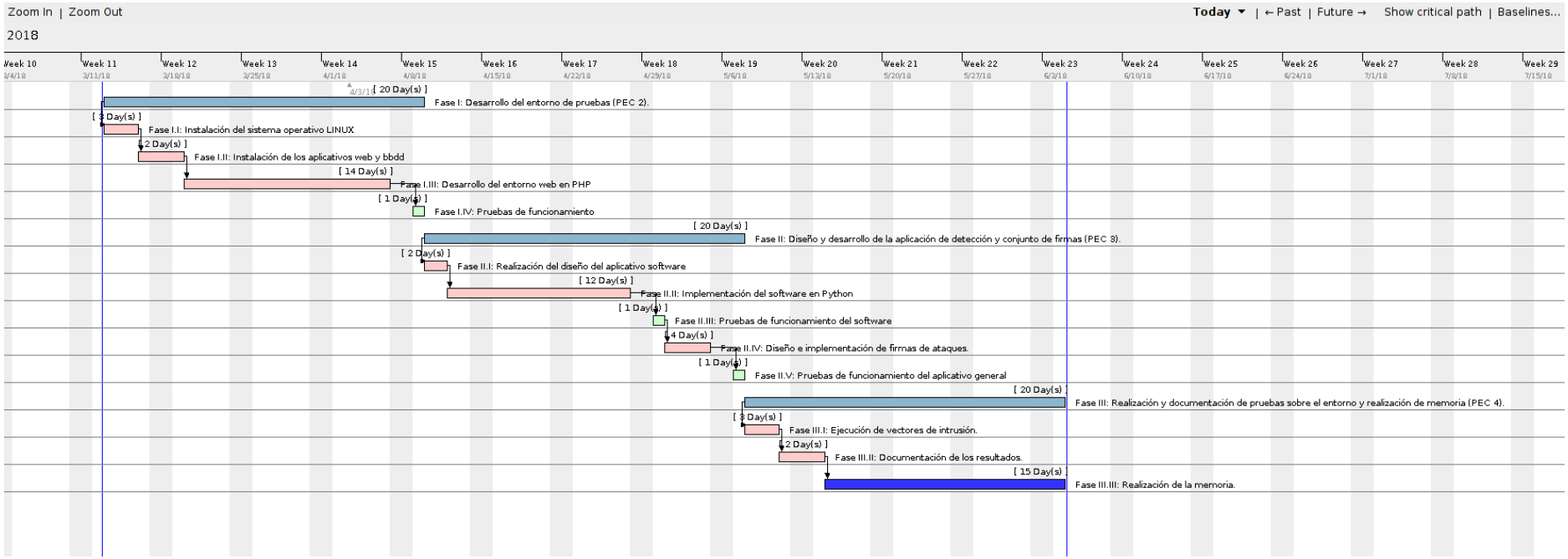


Figura 2: planificación temporal del proyecto.

1.5 Breve sumario de productos obtenidos

El producto que se ha obtenido de este proyecto se trata de una herramienta que da soporte a los equipos de monitorización y respuesta de incidentes. La herramienta es capaz de realizarlo a través del análisis exhaustivo de los registros de aplicativos web, aplicando sobre estos un motor de detección basado en expresiones regulares. Esta utilidad ha alcanzado tasas de detección de entre el 40% al 50% en las pruebas realizadas, tasa que demuestra que el aplicativo es capaz de dar dicho soporte a los equipos arriba mencionados.

1.6 Breve descripción de los otros capítulos de la memoria

La memoria se ha estructurado en un total de 7 capítulos. Esta estructura contiene el proyecto completo aportando en primera instancia los conocimientos teóricos básicos que sustentarán los capítulos 3 a 7. En último lugar tendremos los capítulos 8, 9 y 10 que contienen el glosario, bibliografía y anexos respectivamente. A continuación se detallan brevemente los capítulos 2 a 7:

Capítulo 2: "Conceptos teóricos básicos". En este segundo capítulo se explicarán los conocimientos teóricos básicos necesarios para entender tanto el entorno de trabajo, la herramienta diseñda, las pruebas y conclusiones que se realizarán.

Capítulo 3: "El escenario de trabajo". En este segunda fase se implementará un correlador de eventos de tipo "OSSIM" y el entorno web vulnerable que se usará a posteriori para realizar las pruebas de intrusión.

Capítulo 4: "Herramienta de detección: Morbot - Necromancer". En este capítulo se explica el funcionamiento del motor de detección diseñado. Incluídos los ficheros que contienen los juegos de firmas y configuración.

Capítulo 5: "Pruebas ejecutadas sobre el entorno de trabajo". En el capítulo 5 se realizarán las diferentes pruebas de intrusión siguiendo la metodología de OWASP [3] reducida creada para este proyecto.

Capítulo 6: "Resultados obtenidos". En este capítulo se revisará de forma metódica sí se han encontrado detecciones de intrusiones relativas a los ataques realizados en el capítulo anterior.

Capítulo 7: "Conclusiones". En este último capítulo de la memoria principal se sacarán conclusiones sobre la efectividad de la herramienta implementada. También se revisarán los siguientes pasos que se podrían realizar, dando continuidad a este y el anterior proyecto del autor [9].

2. Conceptos teóricos básicos

2.1 Herramientas de protección

Las herramientas de protección son un conjunto de utilidades informáticas que se han ido creando a lo largo del tiempo con el fin de proteger o defender los sistemas informáticos. Este proyecto se encuentra ubicado entre aquellas herramientas que se utilizan para detectar intrusiones en los sistemas de información. A continuación se explican los tipos de detectores de intrusos que existen.

2.1.1 Dispositivos de tipo detector de intrusos

Llamaremos intrusión a un conjunto de acciones que intentan comprometer la integridad, confidencialidad o disponibilidad de un recurso. Debido a esto denominaremos sistemas de detección de intrusos a los dispositivos encargados de detectar este tipo de comportamientos. Existen principalmente dos tipos en función de la ubicación que ocupa el dispositivo en la red y del comportamiento que tengan tras detectar una alerta:

- **IDS** (sistema de detección de intrusos): estos dispositivos se encuentran offline y a priori sólo detectan eventos sin realizar remediaciones ni mitigaciones sobre el tráfico.
- **IPS** (sistema de prevención de intrusos): estos dispositivos se encuentran online en la red y cada vez que detectan un comportamiento anómalo proceden a mitigar el ataque.

Por otro lado, según su funcionamiento podemos diferenciar cuatro tipos de motores de detección de intrusos:

1. **Network-Based:** se trata de detectores de intrusos que analizan los diferentes tráficos que atraviesan por la red corporativa o DMZ de la empresa.
2. **Wireless:** son un tipo de detectores que monitorizan las redes wireless disponibles así como la actividad sospechosa detectada relacionada con los protocolos inalámbricos.
3. **Network Behavior Analysis (NBA):** estos dispositivos son los encargados de analizar los tráficos de red de manera más analítica y estadística con el fin de detectar comportamientos inusuales de los dispositivos o de determinados flujos de red.
4. **Host-Based:** es un tipo de software que se instala en un equipo terminal para que realice una protección de este. Suele monitorizar la actividad sospechosa trazando la actividad de los usuarios en un sistema

En nuestro caso, el software desarrollado será de tipo “Host-Based”.

2.2 Vulnerabilidades

La RAE (Real Academia Española) define vulnerabilidad al hecho de poder ser herido, o bien recibir una lesión física o moralmente. En nuestro caso, al hablar de sistemas de la información definimos vulnerabilidad como una debilidad de cualquier tipo que se puede utilizar con el fin de comprometer la seguridad de un sistema informático.

Principalmente se pueden desglosar las vulnerabilidades en tres grandes familias:

a) **Vulnerabilidades de diseño:** en esta familia entran principalmente todas las vulnerabilidades relacionadas con diseños erróneos de protocolos de red. También entrarían las vulnerabilidades por las malas políticas de seguridad o arquitecturas de red.

b) **Vulnerabilidades de implantación:** aquí encontramos todas las vulnerabilidades relacionadas con errores de programación de aplicativos o fallos en las implementaciones que realizan los fabricantes de aplicativos, protocolos, etc.

c) **Vulnerabilidades de "uso":** en este grupo definiremos "uso" como la explotación o puesta en marcha de un sistema informático. Es bastante habitual configurar erróneamente las aplicaciones dejando expuesto al exterior más datos de los necesarios.

A continuación se detallan las principales vulnerabilidades que se explotarán a lo largo del proyecto.

SQL injection:

Los ataques inyección SQL explotan una vulnerabilidad muy específica generada por errores de programación web. Estos errores consisten en la no validación de las entradas introducidas por los usuarios en las páginas web. Estas entradas se realizan por ejemplo en los campos de búsqueda. De este modo, un atacante puede aprovechar dicha vulnerabilidad para inyectar código SQL adicional con el fin de alterar el funcionamiento normal de la página web. El nuevo código insertado puede conseguir acceder a datos que normalmente no estarían disponibles, e incluso depende del tipo de inyección SQL que se realice, se puede generar más o menos daño en el entorno, llegando al punto de descargarse la base de datos completa o llegando a realizar un DROP de todas las tablas.

ShellShock:

ShellShock es una vulnerabilidad relativamente nueva que se ha adicionado a este entorno por la facilidad de explotación y de detección que tiene. Simplificando mucho el detalle, esta vulnerabilidad afecta a una gran cantidad de equipos Linux y Unix. Esta permite mediante la definición de una variable la

ejecución de un comando posterior con elevación de privilegios. ShellShock es especialmente peligrosa sí se dispone de servidores que utilizan la “Shell” de tipo BASH en servidores web que interpreten CGI de manera automática.

File Inclusion:

La Inclusión de ficheros se trata de vulnerabilidades que permiten subir un fichero a un directorio web con el fin de posteriormente ejecutarlo. Habitualmente se da en servidores web que alojan imágenes y que no tienen correctamente validada la carga de ficheros. De este modo, un atacante aprovecha esto para subir pequeños “scripts” o software con el fin de posteriormente ejecutarlo de manera remota.

Cross Site Scripting:

Se trata de ataques basados en realizar inyecciones de código con el fin de que el servidor web las interprete y las ejecute realizando tareas para las que no estaba diseñado. Dichas tareas suelen estar orientadas al robo de credenciales o datos comprometidos del usuario. Las inyecciones de código habitualmente se realizan en lenguajes que el servidor web es capaz de interpretar como son lenguajes de scripting tipo “Java Script” o directamente lenguaje HTML.

Directory Transversal:

Este tipo de vulnerabilidades permiten a un atacante acceder a ficheros o directorios superiores (directorios padre) del sistema operativo que aloja un servidor web. Esto ocurre cuando se ejecuta un servidor web con un usuario con demasiados privilegios o incluso con un súper usuario. Explicado esto, un hipotético atacante podría usar esta vulnerabilidad para acceder al fichero “/etc/passwd” (fichero de Linux que contiene listado de usuarios) al poner lo siguiente en la URL de un servidor web Apache montado sobre un sistema Unix / Linux:

http://web.de.ejemplo.com/../../../../../../etc/passwd

Inyección de comandos:

Esta vulnerabilidad consiste en un fallo de programación por el cual un servidor web interpreta el texto que un atacante puede llegar a insertar en un campo de texto libre o campo de búsqueda. El atacante aprovecha la vulnerabilidad insertando comandos que el servidor ejecuta, forzando a este a realizar tareas para las que inicialmente no estaba diseñado.

Inyección XML:

La inyección XML realmente es un derivado de las inyecciones SQL, el principal matiz es que están orientadas a bases de datos XML en lugar de SQL. Al igual que en SQL, se trata de una vulnerabilidad producida por la no validación de la entrada de datos procedentes de los usuarios. Todos estos datos se insertan a través de la página web, y al igual que en casos anteriores, un atacante utilizará este error con el fin de alterar las consultas que se realizan al XML con el fin de extraer la mayor cantidad de información posible o generar daños persistentes.

Fuerza bruta:

Los ataques de fuerza bruta consisten en realizar comprobaciones masivas de autenticación con una gran cantidad de usuarios y contraseñas con el fin de obtener acceso al sistema. Estos ataques intentan explotar fallos de configuración en los servidores. Estos fallos consisten en que las aplicaciones no suelen tener bien configuradas algunos de los siguientes puntos:

- No se limita el número de conexiones e intentos de autenticación que puede realizar un usuario.
- No tienen medidas de seguridad la cual genere bloqueos temporales de usuarios o IPs.
- No limitan la cantidad de autenticaciones durante un periodo de tiempo.
- No tienen políticas de contraseñas adecuadas para los usuarios (las contraseñas son demasiado débiles).

Esto acaba produciendo que los atacantes utilizando herramientas específicas y diccionarios con las contraseñas y usuarios más habituales puedan ganar acceso al sistema sin demasiada dificultad.

2.3 Metodologías de pruebas de intrusión

Cuando se van a realizar pruebas de intrusión se suele utilizar una metodología concreta. Está metodología es básicamente un compendio de pasos o tareas a revisar con el fin de encontrar posibles fallas de seguridad. Uno de los más conocidos para desarrollo web es la metodología conocida como OWASP.

2.3.1 OWASP

OWASP es el acrónimo en inglés que se traduce como "Proyecto abierto de seguridad de aplicaciones web". Es decir, es un proyecto código abierto cuyos objetivos son determinar y combatir principalmente:

- Los vectores de ataque web más utilizados.
- Los fallos que comenten los programadores a la hora de desarrollar aplicativos.
- Los fallos que comenten los administradores de servidores web.

Como tal, OWASP es un organismo sin ánimo de lucro que se encarga de generar documentación, crear herramientas y generar metodologías de revisión

de servidor y aplicativos. El fin de esta labor es la corrección de los fallos antes mencionados.

2.3.2 Metodología utilizada para las pruebas de intrusión

Para las pruebas que se realizarán en este proyecto se seguirá una metodología basada en OWASP [3] pero mucho más reducida. Por tanto, se ha diseñado una metodología adaptada a las necesidades del proyecto y las pruebas que hay que realizar. Esta se ejecutará principalmente en tres fases o grandes bloques:

- Fase I: (recopilación de información) durante la primera fase se realizarán las pruebas menos intrusivas en los sistemas. Básicamente se lanzarán escaneos con el fin de descubrir servicios y versiones asociadas a estos.

- Fase II: (pruebas de gestión de la configuración) en esta segunda fase se intentarán testear los fallos de configuración más habituales en los servidores web como es los accesos a los directorios padre del sistema operativo.

- Fase III: (pruebas de validación de datos) en esta última fase se procederá a realizar todas las pruebas de inyección de código malicioso con el fin de vulnerar los sistemas de información.

3. El escenario de trabajo

3.1 Entorno virtual

Para la realización de las pruebas se ha generado un entorno virtual muy sencillo con dos máquinas virtuales y dos redes diferenciadas. Estas dos máquinas son:

- "TFM – Equipo de pruebas": contiene el servidor web "LAMP" dónde alojaremos nuestra página web vulnerable.
- "TFM – AlienVault": contiene un sistema de correlación de eventos de seguridad basado en tecnología de software libre "OSSIM" dónde volcaremos los eventos facilitando así la visualización de estos.

A continuación se adjunta el direccionamiento de red:

Red	Direccionamiento	IP host	TFM – Equipo de pruebas	TFM - AlienVault
Interconexión con el host	192.168.58.0/24	192.168.58.1	192.168.58.138	192.168.58.140
Interconexión interna	192.168.128.0/24	N/A	192.168.128.200	192.168.128.100

El diagrama de red de los elementos implicados sería el siguiente:

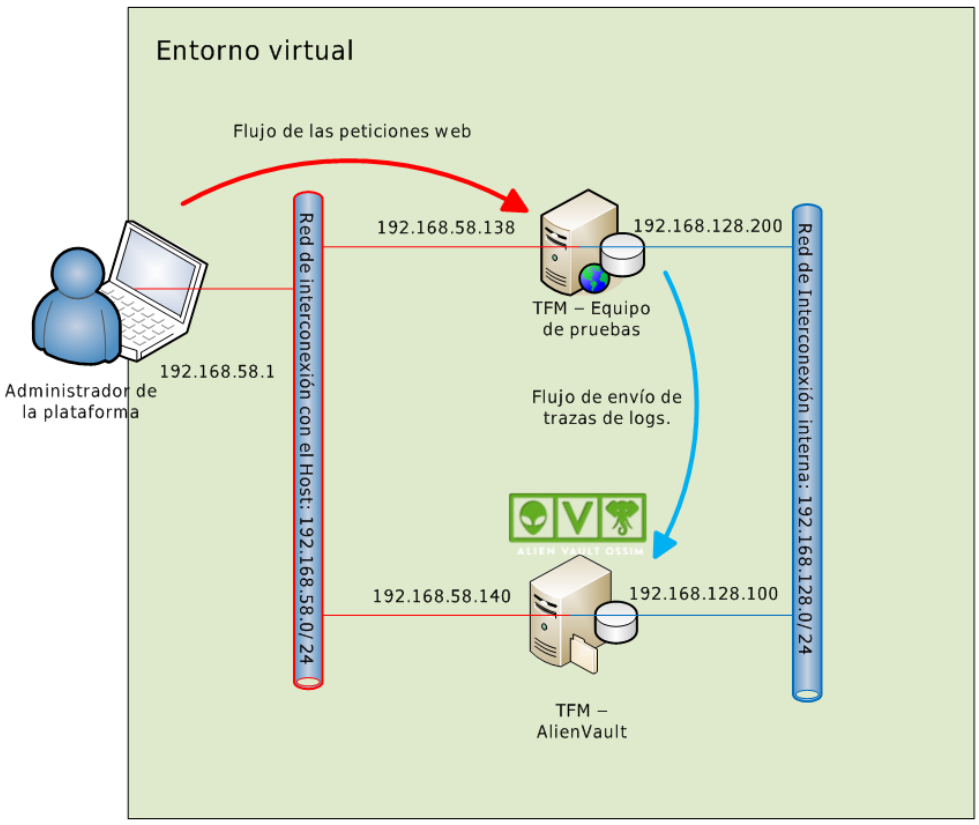


Figura 3: entorno virtual para realizar las pruebas.

3.1.1 Carácterísticas hardware virtual del entorno

Los requisitos hardware a nivel de máquinas virtuales del entorno son los que se muestran a continuación:

Máquina: "TFM – Equipo de pruebas":

Esta máquina es la que contiene el servidor web sobre un entorno de tecnología LAMP. Cómo el entorno es de pruebas y no va a recibir gran carga se ha generado una máquina virtual con las siguientes características:

- Memoria: 4GB.
- Procesdores: 2
- Disco duro: 20GB
- Interfaz de red 1: tipo NAT
- Interfaz de red 2: tipo LAN Segment.

Dónde la primera interfaz está en modo NAT para dar conexión al equipo con el "Host" y poder administrarlo cómodamente y la segunda interfaz está en "LAN Segment" para que se comunique con la otra máquina virtual del entorno.

Máquina: "TFM – AlienVault":

Esta máquina es la que contiene el servidor de OSSIM y se encarga del procesado, presentación y visualización de los datos. Cómo es un entorno que requiere más rendimiento y espacio en disco se ha generado la siguiente máquina virtual:

- Memoria: 4GB.
- Procesdores: 2
- Disco duro: 50GB
- Interfaz de red 1: tipo NAT
- Interfaz de red 2: tipo LAN Segment.

Dónde la primera interfaz está en modo NAT para dar conexión al equipo con el "Host" y poder administrarlo cómodamente y la segunda interfaz está en "LAN Segment" para que se comunique con la otra máquina virtual del entorno.

3.2 Servidor WEB vulnerable

El servidor web vulnerable está compuesto por un conjunto de páginas webs muy sencillas escritas en PHP. Al ser un servidor de pruebas no contienen contenido real, todo lo que tiene son noticias falsas rellenadas con fragmetos aleatorios de texto. Las páginas webs que se han diseñado son:

3.2.1 Página de conexión a base de datos

Descripción: Se trata del segmento de código que contiene los datos de conexión a la base de datos dónde se almacenan las noticias.

Nombre del fichero: “bbdd.php”.

Código fuente de la página:

```
<?php
 try {
        $username = "root";
        $password = "Roro2009";
        $hostname = "localhost";
        $db    =    new    PDO('mysql:host='.$hostname.';dbname=SQLiWeb;charset=utf8mb4',
$username, $password, array(PDO::ATTR_EMULATE_PREPARES => false,
                PDO::ATTR_ERRMODE => PDO::ERRMODE_SILENT));

 } catch(Exception  $ex) {
    //echo "Ha ocurrido algún problema";
 }
 ?>
```

Vista web de la página: (No aplica, no está visible)

3.2.2 Página principal (index)

Descripción: página principal dónde redirige el servidor web. Tiene la opción de hacer login que da paso a la sección de noticias.

Nombre del fichero: “index.php”.

Código fuente de la página:

```
<html>
<head><title>Web de prueba</title></head>
<body>
<h1>Pagina principal</h1>

<?php
if (isset($_COOKIE['name']) && isset($_COOKIE['password'])) {

 echo "Bienvenido, ".$_COOKIE['name'];
 echo '<form action="login.php" method="post" name="Logout">
       <input type="submit" class="linkButton" name="Logout" value="Logout">
        </form>';
 echo '';
} else {
echo '<a href="login.php">Login</a>';
echo '<br>';
echo '<br>';
}
?>

<h1>Noticias</h1>

<!-- RCP seccion de noticias -->

<?php
require_once dirname(__FILE__) . '/bbdd.php';


if (isset($_COOKIE['name']) && isset($_COOKIE['password'])) {

  if (isset($_GET['id']))
  {
   try {
        global $db;
        $id = $_GET['id'];
```

```
        $query = 'SELECT title, body, datetime FROM SQLiWeb.NEWS WHERE id = ' . $id .'
order by title';

        $stmt = $db->prepare($query);
        if (!$stmt){
                echo '<h3>No existe la página consultada</h3>';

      } else {
            $stmt->execute();
            $row = $stmt->fetch();
                if (!$row){
                        echo '<h3>No existe la página consultada</h3>';
                }else{
                        echo  "<h3>".  $row['title']  ."</h3><h4>"  .  $row['datetime']
."</h4><p>" . $row['body'] . "</p><br>";
                }
      }
   } catch(Exception  $ex) {

   }
   } else {

   try {
        $query = 'SELECT id, title FROM SQLiWeb.NEWS order by id';
        $stmt = $db->prepare($query);
        if (!$stmt){
                echo '<br><h3>No existe la página consultada</h3><br>';
        } else {
                $stmt->execute();
                foreach($stmt as $row) {
                        echo        '<br><a        href="news.php?id='.$row['id'].'">'.
$row['title'].'</a><br>';
                }
                echo '<br><br>';
    }
   } catch(Exception  $ex) {

   }
}
} else {
  echo '<h3>Las noticias solo son visibles para usuarios autenticados.</h3>';
}
?>


</body>
</html>
```

Vista web de la página:

Figura 4: página principal entorno web vulnerable.

3.2.3 Página de autenticación

Descripción: página dónde el usuario tiene que introducir sus credenciales para entrar a la sección reservada de la web, en este caso la de noticias.

Nombre del fichero: “login.php”.

Código fuente de la página:

```
<?php
require_once dirname(__FILE__) . '/bbdd.php';


function validarUsuarioContrasena($user, $password) {
  try{
        global $db;

        $query = "SELECT accountId FROM SQLiWeb.USERS WHERE name = '" . $user . "' AND
password = '". $password ."';";

        $stmt = $db->prepare($query);
        if (!$stmt){
                return FALSE;
    } else {
                $stmt->execute();
             $row = $stmt->fetch();
                if (!$row){
                        return FALSE;
                }else{
                        return TRUE;
                }
        }
  } catch(Exception  $ex) {

 }

}


if (isset($_POST['username']) && isset($_POST['password'])) {
        $_COOKIE['name'] = $_POST['username'];
        $_COOKIE['password'] = $_POST['password'];
}


if (isset($_POST['Logout'])) {
        echo 'hola';
        # Delete cookies
        setcookie('name', FALSE);
        setcookie('password', FALSE);

        unset($_COOKIE['name']);
        unset($_COOKIE['password']);

        header("Location: index.php");
}


if (isset($_COOKIE['name']) && isset($_COOKIE['password'])) {
        if (validarUsuarioContrasena($_COOKIE['name'], $_COOKIE['password'])) {
                $login_ok = TRUE;
                $error = "";
        } else {
                $login_ok = FALSE;
                $error = "Usuario o contraseña inválidos<br>";
        }
} else {
        $error = "";
        $login_ok = FALSE;
```

```
}

if ($login_ok == FALSE) {

?>
<html>
<head><title>Login</title></head>
<body>


<h1>Página de login</h1>
<?= $error ?>
<h2>Login</h2>
<form action="#" method="post">
User: <input type="text" name="username"><br>
Password: <input type="password" name="password"><br>
<input type="submit" value="Login">
</form>

</body></html><?php

exit (0);

} else {
  //Si no redirigimos a la página principal
  header("Location: index.php");
}

setcookie('name', $_COOKIE['name']);
setcookie('password', $_COOKIE['password']);


?>
```

Vista web de la página:

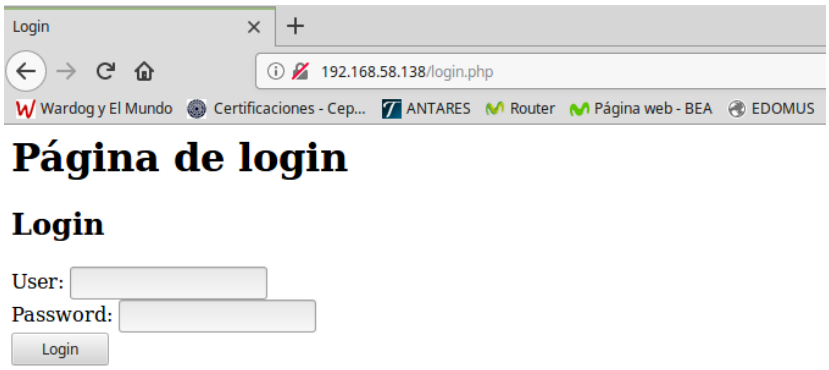


Figura 5: página de login entorno web vulnerable.

3.2.4 Página con la zona reservada de noticias

Descripción: página que contiene las noticias “inventadas” de la web.

Nombre del fichero: “news.php”.

Código fuente de la página:

```
<html><head><title>Noticias</title></head>
<body>
<h1>Noticias</h1>
```

```
<?php
require_once dirname(__FILE__) . '/bbdd.php';

if (isset($_GET['id']))
{
 try {
        global $db;
        $id = $_GET['id'];

        $query = 'SELECT title, body, datetime FROM SQLiWeb.NEWS WHERE id = ' . $id .'
order by title';

        $stmt = $db->prepare($query);
        if (!$stmt){
                echo '<h3>No existe la página consultada</h3>';

    } else {
             $stmt->execute();
             $row = $stmt->fetch();
                if (!$row){
                        echo '<h3>No existe la página consultada</h3>';
                }else{
                        echo  "<h3>".  $row['title']  ."</h3><h4>"  .  $row['datetime']
."</h4><p>" . $row['body'] . "</p><br>";
                }
    }
 } catch(Exception  $ex) {

 }
} else {

 try {
        $query = 'SELECT id, title FROM SQLiWeb.NEWS order by id';
        $stmt = $db->prepare($query);
        if (!$stmt){
                echo '<br><h3>No existe la página consultada</h3><br>';
        } else {
                $stmt->execute();
                foreach($stmt as $row) {
                        echo         '<br><a         href="news.php?id='.$row['id'].'">'.
$row['title'].'</a><br>';
                }
                echo '<br><br>';
    }
 } catch(Exception  $ex) {

 }
}
?>


<br><a href="index.php">Back</a>
<br>
</body></html>
```

Vista web de la página:

Figura 6: página post-autenticación entorno web vulnerable.

3.3 Prueba inicial de concepto

Cómo prueba inicial del entorno lo que haremos será realizar una inyección de código, posteriormente revisaremos que esta se registra con el nivel de debug suficiente cómo para que nuestro motor de detección fuese capaz de detectar los patrónes anómalos. Para ello realizamos la siguiente inyección de código:

> http://192.168.58.138/news.php?id=1%20OR%20UNION%20SELECT%20*%20FROM

Esto puede verse en la siguiente captura:

Figura 7: prueba inicial de concepto, inyección de código.

Acto seguido podemos ver cómo en los logs del aplicativo apache se ha registrado este tipo de comportamiento:

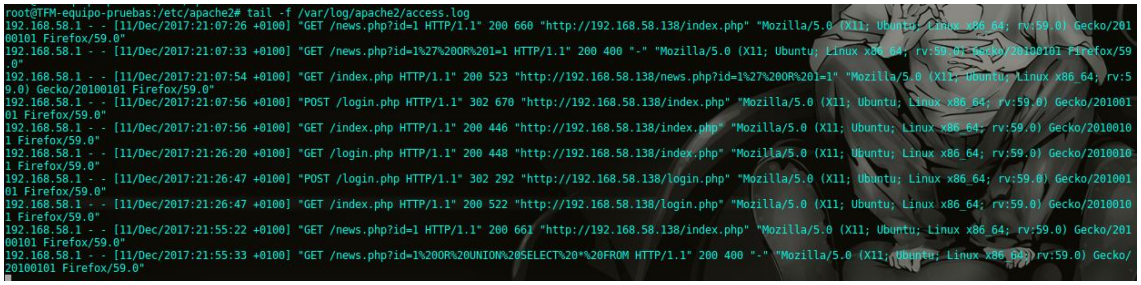


Figura 8: prueba inicial del entorno, detección de la inyección.

En concreto en las siguientes trazas subrayadas en verdes:

…
192.168.58.1 - - [11/Dec/2017:21:26:47 +0100] "GET /index.php HTTP/1.1" 200 522 "http://192.168.58.138/login.php" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0"
192.168.58.1 - - [11/Dec/2017:21:55:22 +0100] "GET /news.php?id=1 HTTP/1.1" 200 661 "http://192.168.58.138/index.php" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0"
192.168.58.1 - - [11/Dec/2017:21:55:33 +0100] "GET /news.php?id=1%20OR%20UNION%20SELECT%20*%20FROM HTTP/1.1" 200 400 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0"
…

4. Herramienta de detección: Morbot - Necromancer

4.1 Descripción general de la herramienta

La herramienta desarrollada en este proyecto a grandes rasgos se encarga de detectar los patrónes anómalos que se quedan registrados en los logs de un aplicativo web. Esto lo realiza a través de un motor de inferencia dónde se aplican un conjunto de reglas diseñadas con expresiones regulares sobre las trazas "en crudo" que dejan los servicios webs. Una vez que se detecta un patrón, el motor genera una salida de datos y pasa a analizar la siguiente línea del fichero de logs. Este proceso se realiza cíclicamente mientras queden líneas que leer, de tal modo que va leyendo un fichero de trazas y generando trazas propias con los patrones detectados.
A lo largo de los siguientes puntos se irá detallando la herramienta en profundidad.

4.2 Componentes de la herramienta

La herramienta que se ha diseñado tiene varias partes claramente diferenciadas. Por un lado tiene el código fuente, dónde se ha diseñado todo el motor de detección o inferencia. En segundo lugar tiene un conjunto de reglas que se aplican sobre las trazas del servidor web apache que se recolectan. Estas reglas son las que se encarga de utilizar el motor de detección.En tercer lugar dispone de un fichero de configuración, en este se puede configurar parámetros relativos a los “inputs” y “outputs” del programa así cómo exceptuar firmas y algunos parámetros de rendimiento.
En los siguientes subpuntos se irá detallando que contiene cada uno de ellos.

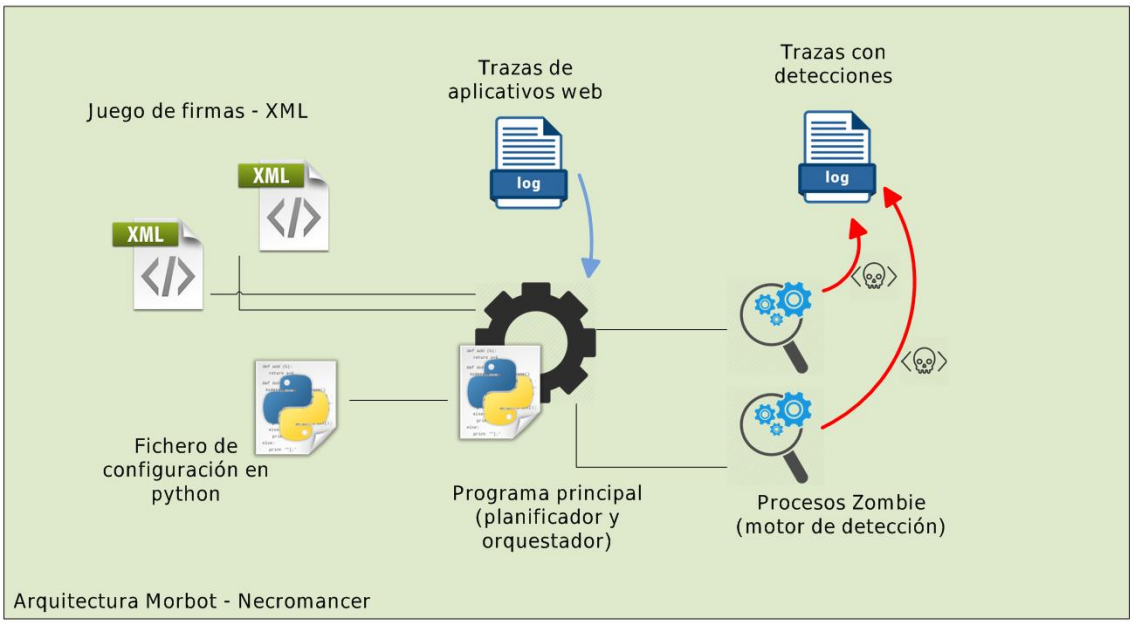


Figura 9: arquitectura software de detección.

4.2.1 Motor de detección

El motor de detección está compuesto principalmente por un programa escrito en python (nec_analyzer.py) y una librería que se utiliza para el procesado de eventos (zombie_analyzer_class.py).
El programa principal se utililiza principalmente para las siguientes labores:

- Generación del juego de firmas: el programa principal se encarga de la compilación y generación del juego de firmas que se pasa luego a los procesos hijos de detección. Para ello se compilan las expresiones regulares y se generan las estructuras de datos necesarios con las siguientes funciones:

- def compile_all_patterns(list_of_patterns, organized_list_of_patterns, organized_list, list_of_compile_patterns):
- def compile_rule(rule):
- def load_parser_rules(format_rules, select_tecnology, rule_id):
- def load_xml_patterns(pattern_file, list_of_patterns, exclude_ids, list_of_exceptions):

Adicionalmente, existe un pequeño módulo de aprendizaje que se encarga de organizar los patrones de detección por orden de ocurrencia. Esto se realiza con el fin de que el motor sea lo más óptimo posible ya que una vez que se encuentra un patrón dejan de analizarse el resto de firmas. La función que se encarga de organizar los patrones de firmas por orden de "matchs" es:

- def learning_module(list_of_patterns, organized_list_of_patterns, organized_list):

- Carga de configuraciones: cuando se inicializa el programa en Python este va a cargar todas las configuraciones generales. Para ello importa del fichero "necromancer_config.py" una serie de variables globales que son las que definen parámetros generales de funcionamiento. Entre estos parámetros encontramos configuraciones cómo son cuantos procesos hijos se pueden crear, dónde están los patrones de detección o cuales son los ficheros de entrada y salida de datos del programa.

- Control de los procesos de detección: la sección principal del programa también se encarga de controlar el número total de procesos hijos que están revisando los diferentes logs que llegan de los aplicativos webs. Esto se realiza con la sección central de código que existe en el "Main" del aplicativo junto con la siguiente función:

- def control_number_of_zombies(number_of_zombies):

Este conjunto se encarga de que haya siempre el menor número de procesos de detección, (para consumir la menor cantidad de recursos de CPU), habiendo cómo máximo el parámetro global del fichero de configuración

"MAX_ZOMBIES". Por tanto, el código del “Main” se encarga de orquestar los procesos de detección y repartir el trabajo de procesado que va llegando.

En último lugar, tenemos la librería de procesado (zombie_analyzer_class.py). Esta librería contiene la "class" Zombie, cuya definición puede verse a continuación:

```
class Zombie:

""" Clase principal de motor de deteccion de patrones en logs """

def __init__ (self, list_of_compile_patterns, list_of_compile_useragent_patterns,
output, rule_00_compiled, rule_11_compiled,fileInMemory, ua_engine_detection):
```

Cómo se observa, esta clase se invoca pasandole todos los patrones de detección compilados así cómo los "inputs" y "outputs" de nuestro programa. De tal modo que ella únicamente se encarga de analizar traza a traza de log e indicar sí concuerda con alguno de los patrones de ataque codificados en nuestro aplicativo.

4.2.2 Patrones de detección y formatos

Existen principalmente tres tipos formatos o patrones que utiliza el aplicativo. Dos de ellos están orientados a la detección de ataques o anomalías y el tercero está orientado a que el aplicativo sepa “entender” distintos formatos de logs y no se límite únicamente al formato de apache. A continuación se detallan cada uno de estos ficheros.

Fichero XML con firmas de detección de ataques (XML_attack_filter.xml):
Este fichero XML contiene en formato estructurado cada una de las firmas que el motor de inferencia escaneará sobre los logs para detectar intrusiones. El formato en el que se almacenan es el siguiente:

```
<filter>
    <id>1</id>
    <rule><![CDATA[(?:"[^"]*[^-]?>)|(?:[^\w\s]\s*\/>)|(?:>")]]></rule>
    <description>finds html breaking injections including whitespace attacks</description>
    <impact>4</impact>
    <exception></exception>
</filter>
```

Dónde se establecen los siguientes campos:

- Id: identificador único de cada firma.
- Rule: patrón de detección en formato de expresión regular de python.
- Descripción: breve definición del ataque que detecta cada firma.
- Impact: grado de severidad que tiene una firma.
- Exception: configuración de posibles excepciones que puede tener una firma.

Las firmas que se han diseñado para este proyecto son las siguientes:

ID	Descripción	ID	Descripción	ID	Descripción
1	finds html breaking injections including whitespace attacks	16	Detects possible includes and typical script methods	31	Detects common XSS concatenation patterns 2/2
2	finds attribute breaking injections including whitespace attacks	17	Detects JavaScript object properties and methods	32	Detects possible event handlers
3	finds unquoted attribute breaking injections	18	Detects JavaScript array properties and methods	33	Detects obfuscated script tags and XML wrapped HTML
4	Detects url-, name-, JSON, and referrer-contained payload attacks	19	Detects JavaScript string properties and methods	34	Detects attributes in closing tags and conditional compilation tokens
5	Detects hash-contained xss payload attacks, setter usage and property overloading	20	Detects JavaScript language constructs	35	Detects common comment types
6	Detects self contained xss via with(), common loops and regex to string conversion	21	Detects very basic XSS probings	37	Detects base href injections and XML entity injections
7	Detects JavaScript with(), ternary operators and XML predicate attacks	22	Detects advanced XSS probings via Script(), RexExp, constructors and XML namespaces	38	Detects possibly malicious html elements including some attributes
8	Detects self-executing JavaScript functions	23	Detects JavaScript location/document property access and window access obfuscation	39	Detects nullbytes and other dangerous characters
9	Detects the IE octal, hex and unicode entities	24	Detects basic obfuscated JavaScript script injections	40	Detects MySQL comments, conditions and ch(a)r injections
10	Detects basic directory traversal	25	Detects obfuscated JavaScript script injections	41	Detects conditional SQL injection attempts
11	Detects specific directory and path traversal	26	Detects JavaScript cookie stealing and redirection attempts	42	Detects classic SQL injection probings 1/2
12	Detects etc/passwd inclusion attempts	27	Detects data: URL injections, VBS injections and common URI schemes	43	Detects classic SQL injection probings 2/2
13	Detects halfwidth/fullwidth encoded unicode HTML breaking attempts	28	Detects IE firefoxurl injections, cache poisoning attempts and local file inclusion/execution	44	Detects basic SQL authentication bypass attempts 1/3
14	Detects possible includes, VBSCript/JScript encodeed and packed functions	29	Detects bindings and behavior injections	45	Detects basic SQL authentication bypass attempts 2/3
15	Detects JavaScript DOM/miscellaneous properties and methods	30	Detects common XSS concatenation patterns 1/2	46	Detects basic SQL authentication bypass attempts 3/3

ID	Descripción	ID	Descripción	ID	Descripción
47	Detects concatenated basic SQL injection and SQLLFI attempts	62	Detects common function declarations and special JS operators	80	Generic SQL injection attemp
48	Detects chained SQL injection attempts 1/2	63	Detects common mail header injections	81	Generic SQL injection attemp
49	Detects chained SQL injection attempts 2/2	64	Detects perl echo shellcode injection and LDAP vectors	82	Detects /etc/shadow inclusion attempts
50	Detects SQL benchmark and sleep injection attempts including conditional queries	65	Detects basic XSS DoS attempts	83	Detects php remote file include attempt 1
51	Detects MySQL UDF injection and other data/structure manipulation attempts	67	Detects unknown attack vectors based on PHPIDS Centrifuge detection	84	Detects php remote file include attempt 2
52	Detects MySQL charset switch and MSSQL DoS attempts	68	Finds attribute breaking injections including obfuscated attributes	85	Detects php remote file include attempt 3
53	Detects MySQL and PostgreSQL stored procedure/function injections	69	Finds basic VBScript injection attempts	86	Detects php remote file include attempt 4
54	Detects Postgres pg_sleep injection, waitfor delay attacks and database shutdown attempts	70	Finds basic MongoDB SQL injection attempts	87	Detects php remote file include attempt 5
55	Detects MSSQL code execution and information gathering attempts	71	finds malicious attribute injection attempts and MHTML attacks	88	Detects php remote file include attempt 5
56	Detects MATCH AGAINST,	72	Detects blind sqli tests using	89	SQL Password Change

	MERGE, EXECUTE IMMEDIATE and HAVING injections		sleep() or benchmark().		attempt
57	Detects MySQL comment-/space-obfuscated injections and backtick termination	73	An attacker is trying to locate a file to read or write.	90	SQL Password Change attempt
58	Detects code injection attempts 1/3	74	Detects remote code exectuion tests. Will match "ping -n 3 localhost" and "ping localhost -n 3"		
59	Detects code injection attempts 2/3	75	Looking for a format string attack		
60	Detects code injection attempts 3/3	76	Looking for basic sql injection. Common attack string for mysql, oracle and others.		
61	Detects url injections and RFE attempts	77	Looking for intiger overflow attacks, these are taken from skipfish, except 2.2250738585072007e-308 is the "magic number" crash		

Fichero XML con "UserAgents" anómalos (XML_useragent_filter.xml):

Al igual que el fichero de ataques, el fichero XML con patrones de "UserAgents" contiene en formato estructurado cada una de las firmas que el motor de inferencia escaneará sobre los logs para detectar navegadores web anómalos. El formato en el que se almacenan es el siguiente:

```
<filter>
    <id>100000</id>
    <rule><![CDATA[((GET|HEAD|POST)\s+\S+"Wget\/\d+(\.)?\d+\s+)]]></rule>
    <description>Wget User agent detected</description>
    <impact>1</impact>
    <exception></exception>
</filter>
```

A diferencia del fichero XML de ataques estos empiezan en la ID "100000" para que no haya solapamiento de identificadores, ya que los identificadores tienen que ser únicos. Al igual que en el anterior fichero, se establecen los siguientes campos:

- Id: identificador único de cada firma.
- Rule: patrón de detección en formato de expresión regular de python.
- Descripción: breve definición del ataque que detecta cada firma.
- Impact: grado de severidad que tiene una firma.
- Exception: configuración de posibles excepciones que puede tener una firma.

El conjunto de "UserAgents" anómalos que se han recopilado son los siguientes:

ID	Descripción	ID	Descripción	ID	Descripción
100000	Wget User agent detected	100012	Nikto scanner attack detected	100024	DTS Agent User-Agent detected
100001	Havij User agent detected	100013	Nmap scanner attack detected	100025	FOCA User-Agent detected
100002	Nikto User agent detected	100014	Nsauditor scanner attack detected	100026	DirBuster User-Agent detected
100003	Acunetix scanner attack detected	100015	Paros scanner attack detected	100027	HTTrack User-Agent detected
100004	Acunetix scanner attack detected	100016	SQLmap scanner attack detected	100028	Morfeus Fucking Scanner User-Agent

					detected
100005	Acunetix scanner attack detected	100017	Scanner attack detected	100029	Nmap scanner attack detected
100006	Dfind scanner attack detected	100018	Scanner attack detected	100030	Paros scanner attack detected
100007	Dfind scanner attack detected	100019	Scanner attack detected	100031	SiteSucker scanner attack detected
100008	Havij scanner attack detected	100020	XSpider scanner attack detected	100032	Sputnik scanner attack detected
100009	Morfeus scanner attack detected	100021	XSpider scanner attack detected	100033	SWebsecurifyScanner scanner attack detected
100010	Nessus scanner attack detected	100022	DigExt scanner attack detected	100034	ZmEu scanner attack detected
100011	Netsparker scanner attack detected	100023	DigExt User-Agent detected		

Fichero XML con formatos soportados (XML_uri_format.xml):

El fichero "XML_uri_format.xml" contiene todos los formatos de logs que soporta el aplicativo. Básicamente cada una de las tecnologías tiene dos formatos o reglas distintas:

- rule_00: este formato contiene todos los eventos que se consideran normales y que no se quieren escanear. Se activa principalmente cuando por necesidades de rendimiento no se quieren escanear las peticiones a imágenes, css, o similares.
- rule_01: esta segunda regla se aplica tras la "rule 00" y contiene el patrón de logs que queremos que pasen al motor de detección.

Existe un formato especial que es "process-all", este formato contiene un conjunto de reglas para que se analicen todas las trazas que reciba el fichero de logs.

El contenido de este fichero es el que se muestra a continuación:

```
<?xml version="1.0"?>
<!-- Rule00: Descarte de eventos considerados normales -->
<!-- Rule11: Eventos a procesar -->

<tecnologies>
    <tecnology>
        <name>Apache</name>
        <rule_00><![CDATA[(.*(GET|POST|HEAD)     (\/|\-|\_|\.|[a-zA-Z0-9]|\)|\(|\?|\%|)*
HTTP/1.(0|1)\" ([0-9]+).*)]]></rule_00>
        <rule_11><![CDATA[(.*(GET|POST|HEAD) \S+ \S+\" ([1-5][0-9][0-9]).*)]]></rule_11>
    </tecnology>
    <tecnology>
        <name>F5</name>
        <rule_00><![CDATA[(.*(GET|POST|HEAD)                    (\/|\-|\_|\.|[a-zA-Z0-
9]|\)|\(|\?|\%|)*\s+\S+\s+([0-9]+|-).*)]]></rule_00>
        <rule_11><![CDATA[(.*(GET|POST|HEAD) \S+ \S+\" (\d+|-).*)]]></rule_11>
    </tecnology>
    <tecnology>
        <name>BlueCoat-px-inverso</name>
        <rule_00><![CDATA[(.*(GET|POST|HEAD)                    (\/|\-|\_|\.|[a-zA-Z0-
9]|\)|\(|\?|\%|)*\s+\S+\s+\S+\s+([0-9]+|-).*)]]></rule_00>
        <rule_11><![CDATA[(.*(GET|POST|HEAD) \S+ \S+ \S+ (\d+|-).*)]]></rule_11>
    </tecnology>
    <tecnology>
        <name>process-all</name>
        <rule_00><![CDATA[(GETPOSTHEADCONNECT)]]></rule_00>
        <rule_11><![CDATA[(.*)]]></rule_11>
```

```
    </tecnology>
</tecnologies>
```

4.2.3 Fichero de configuración

El fichero de configuración se denomina “necromancer_config.py” y contiene todos los parámetros configurables a nivel general del aplicativo. A continuación se detalla los segmentos que tiene:

Sección 1: Ficheros de logs.
Se trata por un lado del fichero dónde el aplicativo va leuyendo así cómo de todos los ficheros dónde el aplicativo va dejando las trazas, (tanto de detección de patrones cómo del propio aplicativo). Al no dar tiempo a adaptar el software para que utilice un paso de mensajes entre procesos, se ha dejado que cada procesador utilice un fichero de salida de trazas distinto. El formato de configuración es el siguiente:

```
# Log files:
LOG = "morbot.log"
input_LOGFILE = "access_in.log"
output_LOGFILE_01 = "morbot01.log"
output_LOGFILE_02 = "morbot02.log"
output_LOGFILE_03 = "morbot03.log"
output_LOGFILE_04 = "morbot04.log"
output_LOGFILE_05 = "morbot05.log"
output_LOGFILE_06 = "morbot06.log"
output_LOGFILE_07 = "morbot07.log"
output_LOGFILE_08 = "morbot08.log"
```

Sección 2: Reglas que aplica la herramienta.
En esta sección se configura el conjunto de patrones que queremos que detecte la herramienta. En un principio se cargarán los patrones de ataques más comunes y los patrones de detección de cabeceras “UserAgent”. Igualmente en esta sección se puede indicar sí queremos que se excluya alguna regla de detección. Esto está orientado especialmente a la detección de “Cross Site Scripting” que suele generar muchos falsos positivos en ocasiones.
El segmento de configuración es el siguiente:

```
# Attack rules:
attack_PATTERNS = "XML_attack_filter.xml"
useragent_PATTERNS = "XML_useragent_filter.xml"

# Exclude attacks by ID:
exclude_ids = [7,8]
```

Sección 3: Formatos de logs
En esta sección se define el tipo de formato de log que estamos analizando. Está orientada a que a través de expresiones regulares el aplicativo sea capaz de procesar trazas del aplicativo “apache”, otros aplicativos web o incluso balanceadores que puedan dejar trazas analizables. La sección de confiuguración es la siguiente:

```
# Parser rules:
format_rules = "XML_uri_format.xml"

# Tecnology - Apache, F5, ...
tecnology = "Apache"
```

Sección 4: Configuración general

En este segmento de la configuración se pueden configurar los parámetros generales del aplicativo. Entre los parámetros generales podemos limitar el número de procesadores que utiliza nuestro aplicativo. Otra opción a configurar es cada cuanto queremos que revise sí han entrado logs nuevos al "input file". Así mismo, podemos habilitar o deshabilitar el motor de detección de "UserAgents", esto es debido a que no todos los logs dejan registrada esta cabecera. El segmento de configuración de este apartado es el siguiente:

```
# Set General configs

# 5 by default, number of cores that process go to use
# Don't support more of 8 cores
number_of_cores = 4

# Number of second to wait after check the file again
loop_interval = 1

# load first priority signatures like SQLi and others 0 - NO, 1 - YES
priority_signatures = 1

# Activate userAgent Engine detection 0 - disable, 1 - enable
useragent_engine_detection = 1
```

4.3 Flujo del funcionamiento

El flujo del funcionamiento de la herramienta al completo es el que se puede ver a continuación:

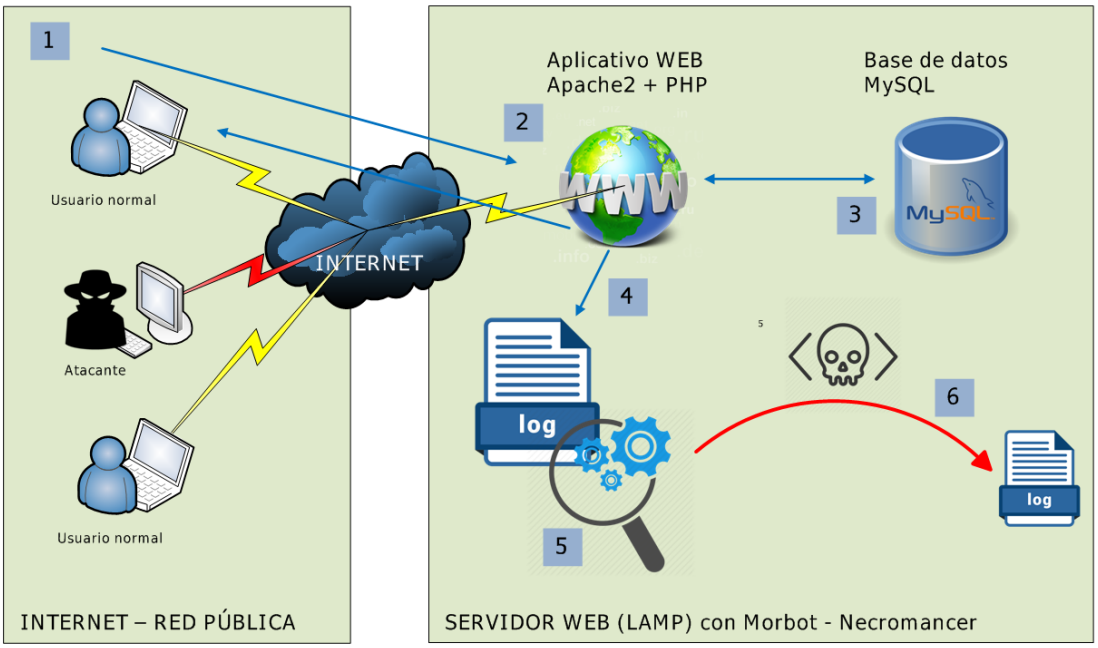


Figura 10: flujo de funcionamiento general de la herramienta y servidor web vulnerable.

La secuencia de acontecimientos sería la siguiente:

1. Realización de peticiones: inicialmente el servidor web está recibiendo peticiones. Entre todas las que recibe se haya una petición anómala que contiene un intento de vulnerar el sistema web.

2. Procesamiento de peticiones: el servidor web recibe las diferentes peticiones de usuarios y las va respondiendo en función del contenido solicitado. Cuando es necesario consultar contenido almacenado pregunta a la base de datos.

3. Consultas a la base de datos: la base de datos recibe y responde las consultas que redirige el frontend o servidor web. Una vez responde a las consultas, el servidor web es el encargado de presentar los datos de forma legible a los usuarios finales.

4. Registro y respuesta de la petición web: el servidor web tras responder a todas las peticiones que ha recibido, las registra en el fichero “access.log”. En dicho fichero registra todas las peticiones, tanto las lícitas cómo las que contienen el contenido malicioso.

5. Detección de nueva traza y búsqueda de patrones maliciosos: una vez entra un registro en el fichero “access.log” el aplicativo “Morbot – Necromancer” se encarga de analizar dicha petición. Para ello lee el fichero y se lo pasa a uno de los motores de procesamiento. En caso de que el registro fuese lícito se ignoraría y se pasaría a la siguiente línea a procesar. En caso contrario, se pasaría al punto 6.

6. Detección de patrón malicioso y registro de evidencia: en caso de que el motor de inferencia detectase un patrón malicioso, este se encargaría de marcarlo añadiendole la siguiente cabecera:

```
Pattern found ID: [Número de la firma]; RAW line: [Línea de log original]
```

Posteriormente lo escribiría en uno de los ficheros de salida de detecciones configurados.

4.4 Prueba inicial de funcionamiento

Para realizar una comprobación de funcionamiento básico de la herramienta vamos a proceder a inyectar el payload siguiente:

```
http://192.168.58.138/news.php?id=1%20OR%20union%20select%20from
```

Con este payload se pretende hacer saltar la firma del motor de detección número 80 (Generic SQL injection attemp) que se muestra a continuación:

```
<filter>
   <id>80</id>
    <rule><![CDATA[((GET|HEAD|POST)\s+\S+(union(.*)select)\S+\s+\S+\"\s+\d+)]]></rule>
    <description>Generic SQL injection attemp</description>
    <impact>3</impact>
    <exception>select\w+</exception>
</filter>
```

Para ello, en primer lugar arrancamos el motor de detección con el siguiente comando:

```
echo "INCIANDO MORBOT - $(date)" ; echo ; python nec_analyzer.py
```

En las siguientes capturas puede verse que se inicializa correctamente (esto se revisar en el fichero “morbot.log” del path dónde se ejecuta el aplicativo). Además se puede observar cómo el motor detecta que hay una nueva línea de log y genera un proceso “Zombie” para procesar estos nuevos datos:

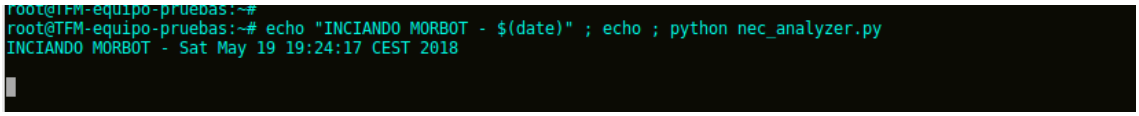


Figura 11: ejemplo de inicialización del software de detección [I].

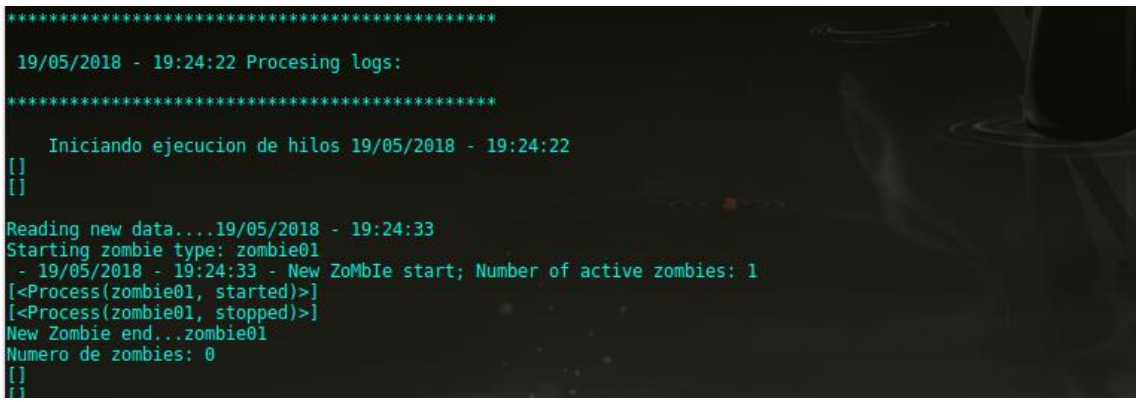


Figura 12: ejemplo de inicialización del software de detección [II].

Tras esto, procedemos a abrir las trazas de logs e inyectar el payload anteriormente mencionado:

Figura 13: prueba de funcionamiento. Ejemplo de inyección.

A continuación procedemos a revisar que el evento de detección se haya generado correctamente en el log configurado (/var/log/apache2/morbot01.log):

Figura 14: prueba de funcionamiento. Detección de la inyección.

Dónde podemos ver la traza original:

```
==> access.log <==
192.168.58.1 - - [19/May/2018:19:24:32 +0200] "GET
/news.php?id=1%20OR%20union%20select%20from HTTP/1.1" 200 400 "-" "Mozilla/5.0 (X11;
Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0"
```

Y la traza generada con el patrón de detección número 80:

```
==> morbot01.log <==
Pattern found ID: 80; RAW line: 192.168.58.1 - - [19/May/2018:19:24:32 +0200] "GET
/news.php?id=1%20OR%20union%20select%20from HTTP/1.1" 200 400 "-" "Mozilla/5.0 (X11;
Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0"
```

Una vez se ha generado la traza, esta es enviada vía syslog al servidor de AlienVault dónde es procesada por el plugin para ser visualizada con más facilidad. A continuación se presenta el formato final que vemos en el correlador de eventos:

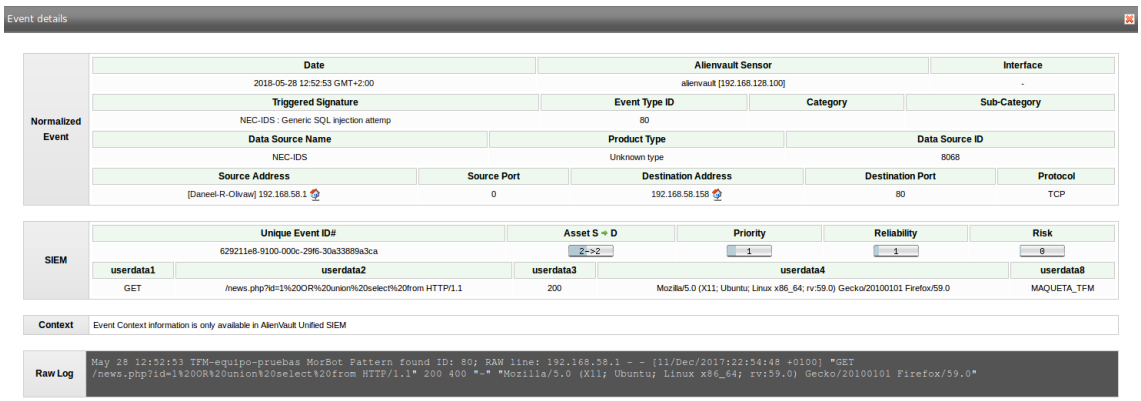


Figura 15: prueba de funcionamiento. Visualización de la inyección en AlienVault.

5. Pruebas ejecutadas sobre el entorno de trabajo

5.1 Fase I: recopilación de información

En este caso al tratarse de pruebas de descubrimiento de servicios y de fingerprinting se van a ignorar. El principal motivo es debido a que la herramienta desarrollada no está diseñada para detectar este tipo de comportamientos y por tanto, pasarán totalmente inadvertidos.

5.2 Fase II: Pruebas de gestión de la configuración

Durante esta fase se irá aumentando el intrusismo de las pruebas. En concreto, se procederá a realizar pruebas de fuerza bruta tanto contra servicios como directorios. En los siguientes puntos desarrollaremos cada una de estas pruebas.

5.2.1 Enumeración de directorios

Para esta primera prueba se ha utilizado una herramienta de software libre conocida como "Wfuzz". Esta herramienta está diseñada con el fin de realizar fuerza bruta de aplicaciones web. Realiza múltiples tipos de fuerza bruta como son de directorios, SQL, XSS o LDAP. En nuestro caso la utilizaremos inicialmente para encontrar recursos no vinculados como son:

- Directorios.
- Servlets.
- Scripts.
- Páginas de administración por defecto.
- Etc.

La ejecución se realiza con los siguientes parámetros:

```
date; python wfuzz-cli.py -w wordlist/general/common.txt --hc 404
http://192.168.58.138/FUZZ
```

El resultado de la prueba es el que puede verse a continuación:

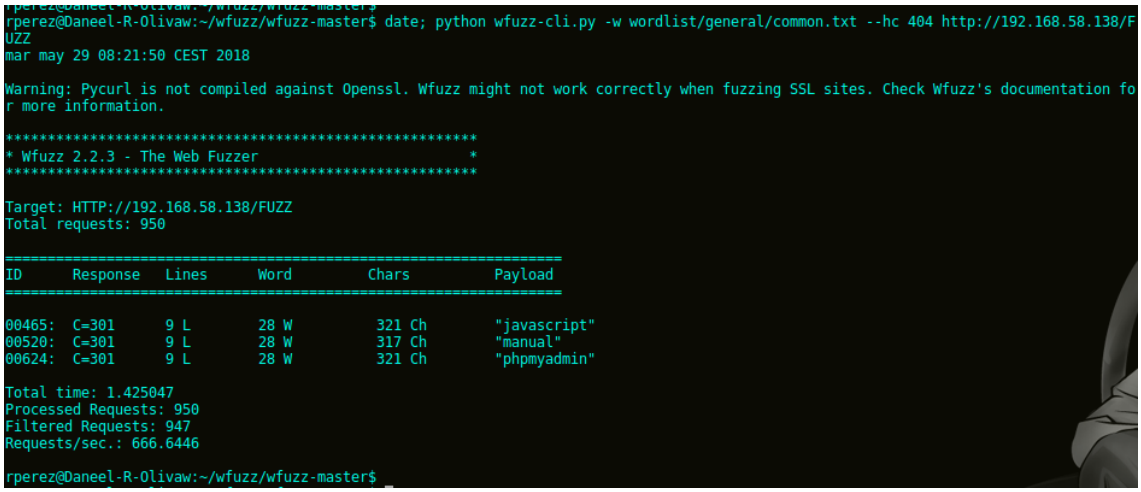


Figura 16: ejecución de fuerza bruta de directorios.

5.2.2 Pruebas de "Directory Transversal"

Para esta segunda prueba inyectaremos el siguiente conjunto de payloads con el fin de extraer varios de los paths más habituales:

- ../../../../../../etc/hosts
- ../../../../../../etc/passwd
- ../../../../../../etc/shadow

Estas pruebas la realizaremos tanto con el comando "wget" cómo con el navegador web. Los comandos inyectados con "wget han sido":

- wget -O - 'http://192.168.58.138/index.php/../../../../../../../etc/passwd'
- wget -O - 'http://192.168.58.138/index.php/../../../../../../../etc/shadow'
- wget -O - 'http://192.168.58.138/index.php/../../../../../../../etc/hosts'

5.2.3 Pruebas de Fuerza bruta

Nuestro aplicativo web no está preparado para registrar los fallos de autenticación. Por tanto, esta prueba no se realiza debido a que el resultado es la "no detección". Igualmente, el aplicativo diseñado no sería capaz de detectar múltiples fallos de autenticación y sería necesario añadir reglas de correlación en el SIEM para detectar este comportamiento.

5.3 Fase III: Pruebas de validación de datos

Esta tercera y última fase ha consistido principalmente en la ejecución de los test más intrusivos. Estos vectores de intrusión son para los que se diseñó esta herramienta de forma específica. Las siguientes tablas reflejan los payloads inyectados en la página web en función del tipo de vector de intrusión.

5.3.1 Inyecciones SQL.

Juego de inyecciones SQL		
ID prueba	**Tipo de Prueba**	**Payload inyectado**
1	Inyecciones SQL	http://192.168.58.138/news.php?id=10000' AND '1'='1
2	Inyecciones SQL	http://192.168.58.138/news.php?id=10000' OR '1'='1
3	Inyecciones SQL	http://192.168.58.138/news.php?id=10000' UNION SELECT FROM
4	Inyecciones SQL	http://192.168.58.138/news.php?id=10000' AND SELECT FROM
5	Inyecciones SQL	http://192.168.58.138/news.php?id=10000' CONCAT SELECT

6	Inyecciones SQL	http://192.168.58.138/news.php?id=10000' UNION SELECT 1,concat(login,':',password),3,4 FROM users;

5.3.2 Inyecciones de código y comandos.

Juego de inyecciones de código y comandos		
ID prueba	**Tipo de Prueba**	**Payload inyectado**
7	Inyecciones de código	http://192.168.58.138/news.php?id=1".system('uname -a');%23
8	Inyecciones de código	http://192.168.58.138/news.php?id=1".system('cat /etc/passwd');%23
9	Inyección de comandos	echo -e "HEAD /cgi-bin/status HTTP/1.1\r\nUser-Agent: () { :;}; echo \$(</etc/hosts)\r\nHost: vulnerable\r\nConnection: close\r\n\r\n" | nc 192.168.58.138 80

5.3.3 Inyecciones de XML y XSS.

Juego de inyecciones XML & XSS		
ID prueba	**Tipo de Prueba**	**Payload inyectado**
10	Inyección XML	http://192.168.58.138/news.php?xml=<!DOCTYPE test [<!ENTITY x SYSTEM "file:///etc/shadow">]><test>%26x;</test>
11	Inyección XML	http://192.168.58.138/news.php?id=1'%20or%201=1] /child::node()%00
12	XSS	http://192.168.58.138/news.php?id=<script>alert(1); </script>
13	XSS	http://192.168.58.138/news.php?id=<<SCRIPT>alert("XSS"); //<</SCRIPT>
14	XSS	http://192.168.58.138/news.php?id=

5.3.4 Escaneos automatizados con herramientas.

Se ha realizado escaneos automatizados con escaneres clásicos de páginas web. En concreto se ha utilizado “Nikto” para ver el comportamiento del motor cuando se lanzan pruebas masivas.

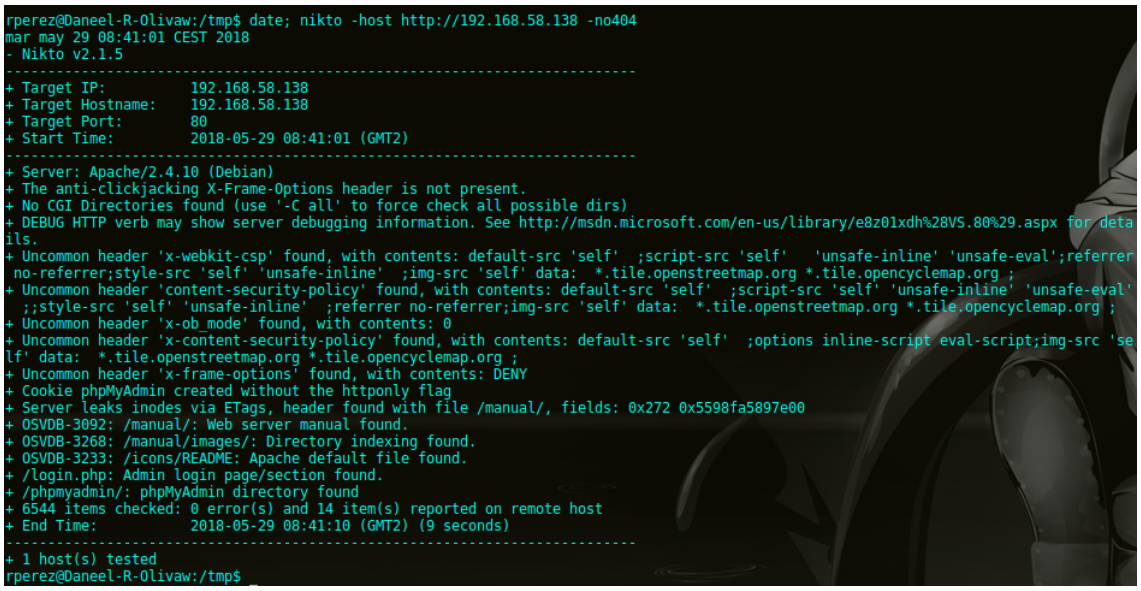


Figura 17: ejecución de escaneo con herramienta Nikto.

6. Resultados obtenidos

En este capítulo se revisarán las pruebas de intrusión realizadas en comunión con los resultados obtenidos en la herramienta desarrollada. De este modo se observará sí cada ataque ha sido detectado o hemos tenido un falso negativo. Cómo resultado final se extraerán los datos cualitativos de cara a analizarlos en el capítulo de conclusiones.

6.1 Resultados fase I: recopilación de información

Cómo ya se indicó en el capítulo "5.1 Fase I: recopilación de información", al tratarse de pruebas de descubrimiento de servicios y de fingerprinting fueron ignoradas. El principal motivo era que la herramienta desarrollada no está diseñada para detectar este tipo de comportamientos, y por tanto, pasarán totalmente inadvertidos. En consecuencia no es necesario realizar las pruebas para saber el resultado de estas.

6.2 Resultados fase II: Pruebas de gestión de la configuración

Durante esta segunda fase de pruebas de intrusión se realizaron ataques de fuerza bruta de directorios y de tipo "Directory trasversal". No se llegaron a ejecutar ataques de fuerza bruta sobre el aplicativo por la misma razón que la fase I. La aplicación de detección no está diseñada para detectar este tipo de ataques y pasarán totalmente desapercibidos. En total se ejecutaron un total de cuatro pruebas de las que fueron detectadas únicamente dos.

Las pruebas realizadas han sido:

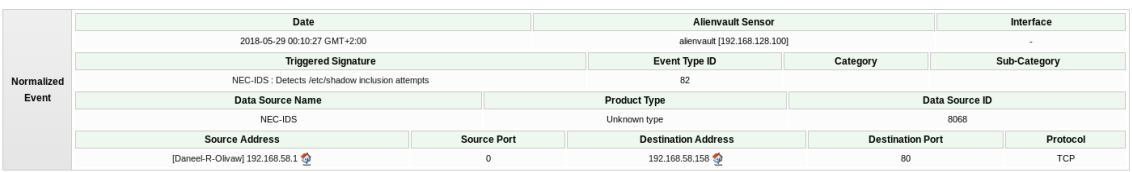
- date; python wfuzz-cli.py -w wordlist/general/common.txt --hc 404 http://192.168.58.138/FUZZ
- wget -O - 'http://192.168.58.138/index.php/../../../../../../../etc/passwd'
- wget -O - 'http://192.168.58.138/index.php/../../../../../../../etc/shadow'
- wget -O - 'http://192.168.58.138/index.php/../../../../../../../etc/hosts'

Cómo se puede ver a continuación, únicamente el acceso a los ficheros “/etc/passwd” y “/etc/shadow” han generado detecciones en el motor. El resto de ataques no se han detectado. A continuación se adjunta las detecciones encontradas relativas a las firmas 12 y 82 que se muestran a continuación:

Firma 12:

```
<rule><![CDATA[(?:etc\/\W*passwd)]]></rule>
<description>Detects etc/passwd inclusion attempts</description>
```

Evento normalizado detectado:



Normalized Event	Date	Alienvault Sensor	Interface
	2018-05-29 00:10:27 GMT+2:00	alienvault [192.168.128.100]	-

Triggered Signature	Event Type ID	Category	Sub-Category
NEC-IDS : Detects /etc/shadow inclusion attempts	82		

Data Source Name	Product Type	Data Source ID
NEC-IDS	Unknown type	8068

Source Address	Source Port	Destination Address	Destination Port	Protocol
[Daneel-R-Olivaw] 192.168.58.1	0	192.168.58.158	80	TCP

Figura 18: Fase II. Visualización OSSIM de inyección /etc/passwd.

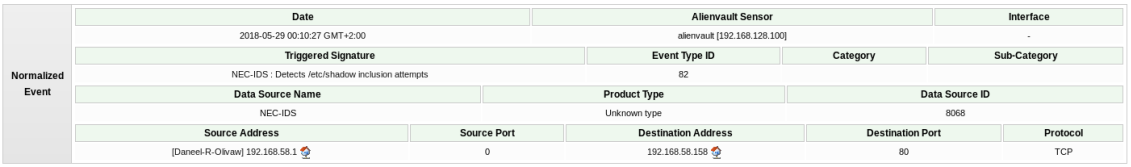
Raw log:

```
May 29 00:10:27 TFM-equipo-pruebas MorBot Pattern found ID: 82; RAW
line: 192.168.58.1 - - [29/May/2018:00:10:20 +0200] "GET /etc/shadow
HTTP/1.1" 404 505 "-" "Wget/1.17.1 (linux-gnu)"
```

Firma 82:

```
<rule><![CDATA[(\/etc\/\W*shadow)]]></rule>
<description>Detects /etc/shadow inclusion attempts</description>
```

Evento normalizado detectado:



Normalized Event	Date	Alienvault Sensor		Interface
	2018-05-29 00:10:27 GMT+2:00	alienvault [192.168.128.100]		-
	Triggered Signature	**Event Type ID**	**Category**	**Sub-Category**
	NEC-IDS : Detects /etc/shadow inclusion attempts	82		
	Data Source Name	**Product Type**		**Data Source ID**
	NEC-IDS	Unknown type		8068
	Source Address	**Source Port**	**Destination Address**	**Destination Port**	**Protocol**
	[Daneel-R-Olivaw] 192.168.58.1	0	192.168.58.158	80	TCP

Figura 19: Fase II. Visualización OSSIM de inyección /etc/shadow.

Raw log:

```
May 29 00:10:27 TFM-equipo-pruebas MorBot Pattern found ID: 82; RAW
line: 192.168.58.1 - - [29/May/2018:00:10:20 +0200] "GET /etc/shadow
HTTP/1.1" 404 505 "-" "Wget/1.17.1 (linux-gnu)"
```

6.3 Resultados fase III: Pruebas de validación de datos

Esta tercera y última fase de pruebas, además de ser la fase en la que más test se han realizado, estos tenían el mayor grado de intrusión posible y los principales vectores de ataque web. Por ello, nuestra herramienta debería ser capaz de detectar gran parte de ellos. Sí analizamos detalladamente los diferentes vectores de intrusión realizados tenemos que:

6.3.1 Inyecciones SQL

Del conjunto de inyecciones manuales SQL sólo se han detectado las inyecciones 5 y 6, que son:

ID prueba	Tipo de Prueba	Payload inyectado
5	Inyecciones SQL	http://192.168.58.138/news.php?id=10000' CONCAT SELECT
6	Inyecciones SQL	http://192.168.58.138/news.php?id=10000' UNION SELECT 1,concat(login,':',password),3,4 FROM users;

Las trazas que han generado estas dos detecciones han sido:

Inyección SQL (prueba 5):
Firma detectada: id – 80.

```
<rule><![CDATA[((GET|HEAD|POST)\s+\S+(union(.*)select)\S+\s+\S+\"\s+\d+)]]></rule>
<description>Generic SQL injection attemp</description>
```

Evento normalizado:

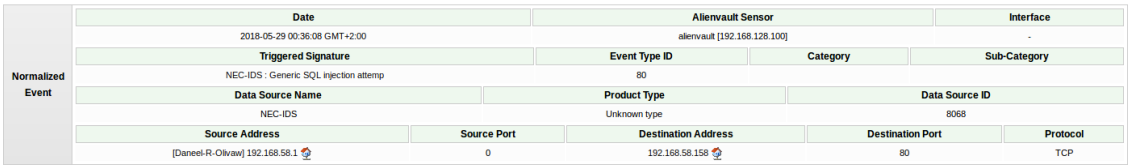


Figura 20: Fase III. Visualización OSSIM de inyección SQL(1).

Raw log:

```
May 29 00:36:08 TFM-equipo-pruebas MorBot Pattern found ID: 80; RAW
line: 192.168.58.1 - - [29/May/2018:00:36:02 +0200] "GET
/news.php?id=10000'%20UNION%20SELECT%20FROM HTTP/1.1" 200 394 "-"
"Wget/1.17.1 (linux-gnu)"
```

Inyección SQL (prueba 6):

Firma detectada: id – 81.

```
<rule><![CDATA[((GET|HEAD|POST)\s+\S+(select(.*)from)\S+\s+\S+\"\s+\d+)]]></rule>
<description>Generic SQL injection attemp</description>
```

Evento normalizado:

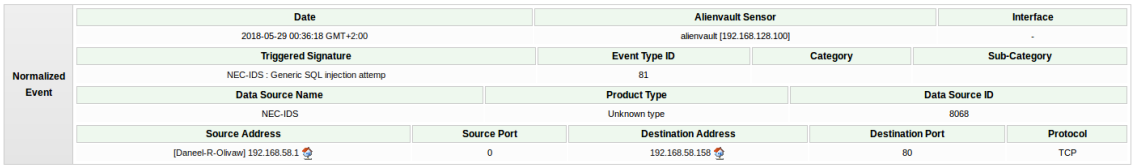


Figura 21: Fase III. Visualización OSSIM de inyección SQL(2).

Raw log:

```
May 29 00:36:18 TFM-equipo-pruebas MorBot Pattern found ID: 81; RAW
line: 192.168.58.1 - - [29/May/2018:00:36:17 +0200] "GET
/news.php?id=10000'%20UNION%20SELECT%201,concat(login,':',password),3,
4%20FROM%20users; HTTP/1.1" 200 394 "-" "Wget/1.17.1 (linux-gnu)"
```

6.3.2 Inyecciones de código y comandos

De las pruebas de inyecciones de comando sólo se han detectado las pruebas 8 y 9. En concreto, la preba 8 se ha detectado de manera indirecta al contener en el comando la visualización del fichero “/etc/passwd”. Por otro lado, la prueba número 9 basada en la vulnerabilidad de shellshock sí ha sido detectada y catalogada cómo tal.

Por tanto, la matriz de detecciones queda de la siguiente manera:

ID prueba	Tipo de Prueba	Payload inyectado
8	Inyecciones de código	http://192.168.58.138/news.php?id=1".system('cat /etc/passwd');%23
9	Inyección de comandos	echo -e "HEAD /cgi-bin/status HTTP/1.1\r\nUser-Agent: () { :;}; echo \$(</etc/hosts)\r\nHost: vulnerable\r\nConnection: close\r\n\r\n" | nc 192.168.58.138 80

El detalle de las detecciones es el siguiente:

Inyección de código (prueba 8):
Firma detectada: id – 12.

```
<rule><![CDATA[(?:etc\/\W*passwd)]]></rule>
<description>Detects etc/passwd inclusion attempts</description>
```

Evento normalizado:

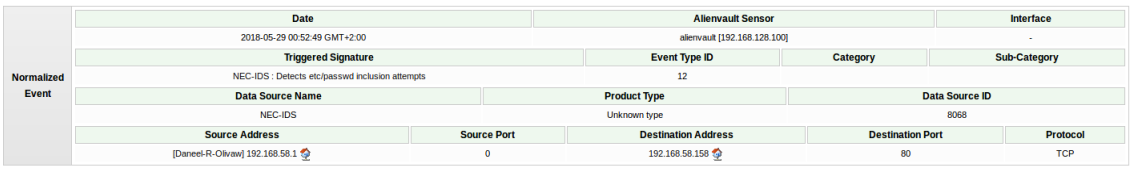


Figura 22: Fase III. Visualización OSSIM de inyección de comando.

Raw log:

```
May 29 00:52:49 TFM-equipo-pruebas MorBot Pattern found ID: 12; RAW
line: 192.168.58.1 - - [29/May/2018:00:52:45 +0200] "GET
/news.php?id=1.system('cat%20/etc/passwd');%23 HTTP/1.1" 200 394 "-"
"Wget/1.17.1 (linux-gnu)"
```

Inyección SQL (prueba 9):

Firma detectada: id – 60.

```
<rule><![CDATA[(?:(?:[;]+|(<[?%](?:php)?)).*[^\w](?:echo|print|print_r|var_dump|[fp]open
))|(?:;\s*rm\s+-\w+\s+)|(?:;.*{.*\$\w+\s*=)|(?:\$\w+\s*\[\]\s*=\s*)]]></rule>
<description>Detects code injection attempts 3/3</description>
```

Evento normalizado:

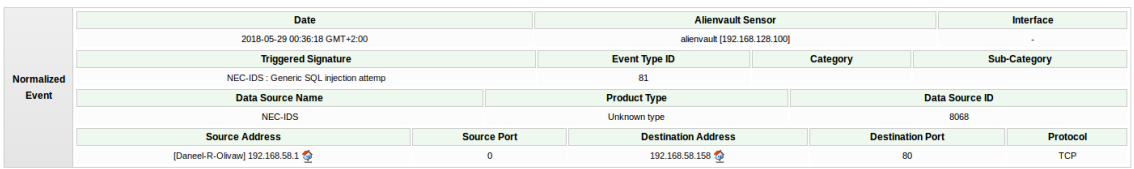


Figura 23: Fase III. Visualización OSSIM de inyección de comando (ShellShock).

Raw log:

```
May 29 00:52:59 TFM-equipo-pruebas MorBot Pattern found ID: 60; RAW line: 192.168.58.1 - - [29/May/2018:00:52:50 +0200] "HEAD /cgi-bin/status HTTP/1.1" 404 159 "-" "() { :;}; echo $(</etc/hosts)"
```

6.3.3 Inyecciones de XML y XSS

De las diferentes pruebas que se han realizado de inyecciones XML y XSS únicamente se ha detectado las pruebas 10 y 14. La primera se ha detectado de manera indirecta al solicitar el fichero “/etc/shadow”, en el segundo caso sí se ha detectado la prueba correctamente cómo un intento de Cross Site Scripting. Por tanto, la matriz de pruebas detectadas queda de la siguiente manera:

ID prueba	Tipo de Prueba	Payload inyectado
10	Inyección XML	http://192.168.58.138/news.php?xml=<!DOCTYPE test [<!ENTITY x SYSTEM "file:///etc/shadow">]><test>%26x;</test>
14	XSS	http://192.168.58.138/news.php?id=

El detalle de las detecciones es el siguiente:

Inyección de código (prueba 10):
Firma detectada: id – 82.

```
<rule><![CDATA[(\/etc\/\W*shadow)]]></rule>
<description>Detects /etc/shadow inclusion attempts</description>
```

Evento normalizado:

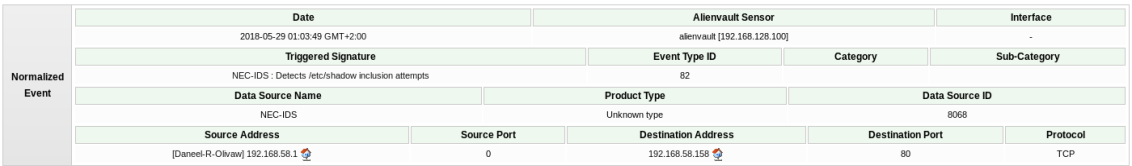


Figura 24: Fase III. Visualización OSSIM de inyección XML.

Raw log:

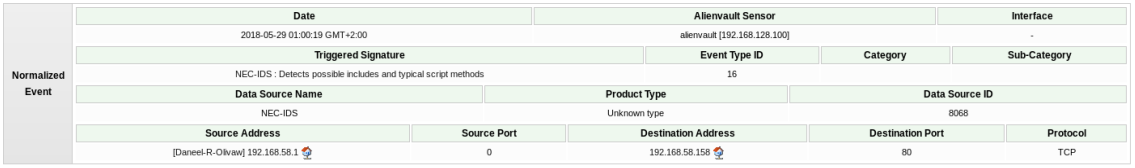
```
May 29 01:03:49 TFM-equipo-pruebas MorBot Pattern found ID: 82; RAW line: 192.168.58.1 - - [29/May/2018:01:03:40 +0200] "GET /news.php?xml=%3C!DOCTYPE%20test%20[%3C!ENTITY%20x%20SYSTEM%20%22file:///etc/shadow%22%3E]%3E%3Ctest%3E%26x;%3C/test%3E HTTP/1.1" 200 405 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0"
```

Inyección SQL (prueba 14):

Firma detectada: id – 16.

```
<rule><![CDATA[([^*\s\w,.\/?+-]\s*)?(?<![a-mo-z]\s)(?<![a-
z\/_@])(\s*return\s*)?(?:alert|inputbox|showmod(?:al|eless)dialog|showhelp|infinity|isna
n|isnull|iterator|msgbox|executeglobal|expression|prompt|write(?:ln)?|confirm|dialog|urn
|(?:un)?eval|exec|execscript|tostring|status|execute|window|unescape|navigate|jquery|get
script|extend|prototype)(?(1)[^\w%"]|(?:\s*[^@\s\w%",.:\/+\-]))]]></rule>
<description>Detects possible includes and typical script methods</description>
```

Evento normalizado:



Normalized Event					
Date		Alienvault Sensor			Interface
2018-05-29 01:00:19 GMT+2:00		alienvault [192.168.128.100]			-
Triggered Signature			Event Type ID	Category	Sub-Category
NEC-IDS : Detects possible includes and typical script methods			16		
Data Source Name		Product Type		Data Source ID	
NEC-IDS		Unknown type		8068	
Source Address	Source Port	Destination Address		Destination Port	Protocol
[Daneel-R-Olivaw] 192.168.58.1	0	192.168.58.158		80	TCP

Figura 25: Fase III. Visualización OSSIM de inyección XSS.

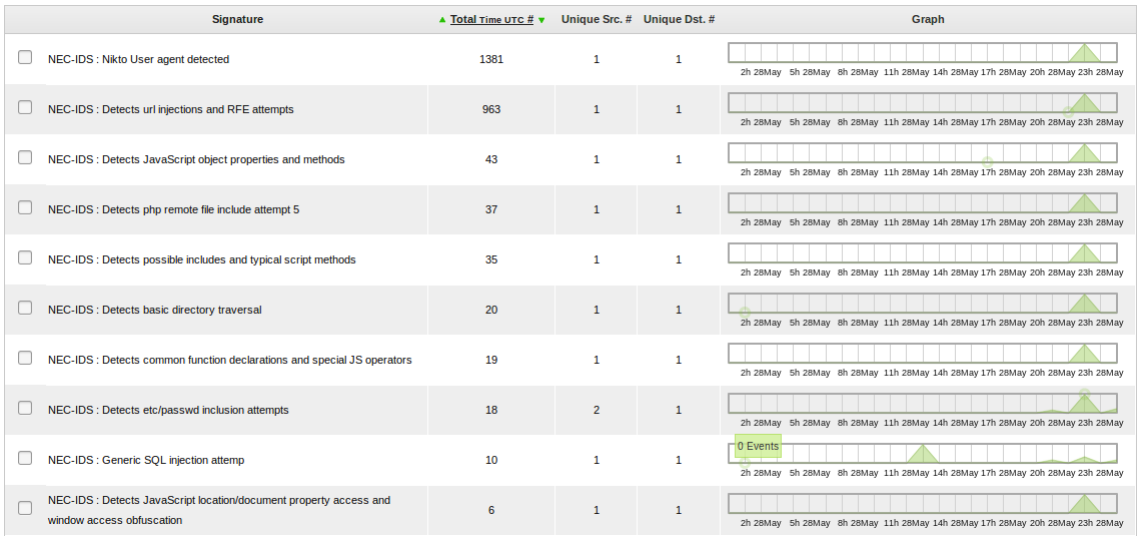
Raw log:

```
May 29 01:00:19 TFM-equipo-pruebas MorBot Pattern found ID: 16; RAW
line: 192.168.58.1 - - [29/May/2018:01:00:17 +0200] "GET
/news.php?id=%3Cimg%20src='zzzz'%20onerror='alert(1)'%20/%3E HTTP/1.1"
200 394 "-" "Wget/1.17.1 (linux-gnu)"
```

6.3.4 Escaneos automatizados con herramientas

Cuando se utilizó “Nikto” para realizar una gran cantidad de pruebas diferentes de forma automatizada la cantidad de las detecciones de la herramienta se disparó. En total se generaron un total de 22 firmas distintas con un cómputo total de 2591 eventos de seguridad.

A continuación se presenta la distribución de eventos detectados durante el tiempo que tardó en lanzarse la herramienta “Nikto”.



Signature	Total Time UTC #	Unique Src. #	Unique Dst. #	Graph
NEC-IDS : Nikto User agent detected	1381	1	1	
NEC-IDS : Detects url injections and RFE attempts	963	1	1	
NEC-IDS : Detects JavaScript object properties and methods	43	1	1	
NEC-IDS : Detects php remote file include attempt 5	37	1	1	
NEC-IDS : Detects possible includes and typical script methods	35	1	1	
NEC-IDS : Detects basic directory traversal	20	1	1	
NEC-IDS : Detects common function declarations and special JS operators	19	1	1	
NEC-IDS : Detects etc/passwd inclusion attempts	18	2	1	
NEC-IDS : Generic SQL injection attemp	10	1	1	
NEC-IDS : Detects JavaScript location/document property access and window access obfuscation	6	1	1	

Graph axis: 2h 28May 5h 28May 8h 28May 11h 28May 14h 28May 17h 28May 20h 28May 23h 28May; 0 Events

Figura 26: Fase III. Visualización OSSIM de ataque con Nikto (1).

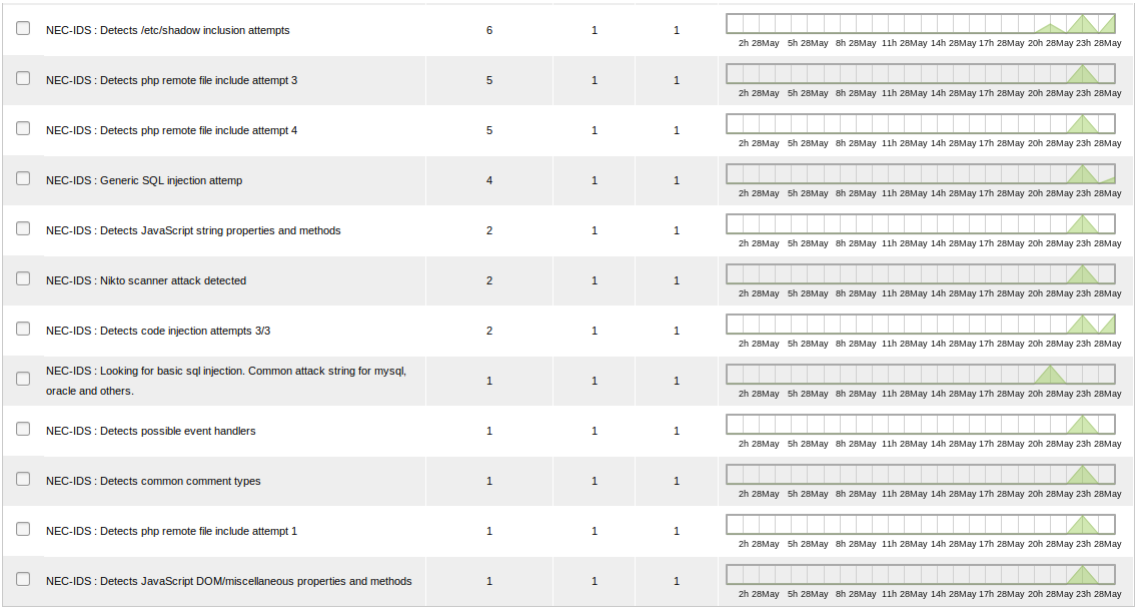


Figura 27: Fase III. Visualización OSSIM de ataque con Nikto (2).

6.4 Tabla resumen: relación payloads ejecutados – detecciones

RESUMEN DE DETECCIONES		
Fase	**Payload inyectado**	**¿Detección?**
I	No aplica	No
II	http://192.168.58.138/index.php/../../../../../../../etc/passwd	Sí
II	http://192.168.58.138/index.php/../../../../../../../etc/shadow	Sí
II	http://192.168.58.138/index.php/../../../../../../../etc/hosts	No
II	date; python wfuzz.py -w wordlist/general/common.txt --hc 404 http://192.168.58.138/FUZZ	No
III	http://192.168.58.138/news.php?id=10000' AND '1'='1	No
III	http://192.168.58.138/news.php?id=10000' OR '1'='1	No
III	http://192.168.58.138/news.php?id=10000' UNION SELECT FROM	No
III	http://192.168.58.138/news.php?id=10000' AND SELECT FROM	No
III	http://192.168.58.138/news.php?id=10000' CONCAT SELECT	Sí
III	http://192.168.58.138/news.php?id=10000' UNION SELECT 1,concat(login,':',password),3,4 FROM users;	Sí
III	http://192.168.58.138/news.php?id=1'.system('uname -a');%23	No
III	http://192.168.58.138/news.php?id=1.system('cat /etc/passwd');%23	Sí
III	HEAD /cgi-bin/status HTTP/1.1\r\nUser-Agent: () { :;}; echo \$(</etc/hosts)\r\nHost: vulnerable\r\nConnection: close\r\n\r\n" | nc 192.168.58.138 80	Sí
III	http://192.168.58.138/news.php?xml=<!DOCTYPE test	Sí

	[<!ENTITY x SYSTEM 'file:///etc/shadow'>]><test>%26x;</test>	
III	http://192.168.58.138/news.php?id=1'%20or%201=1]/child::node()%00	No
III	http://192.168.58.138/news.php?id=<script>alert(1);</script>	No
III	http://192.168.58.138/news.php?id=<<SCRIPT>alert("XSS"); //<</SCRIPT>	No
III	http://192.168.58.138/news.php?id=	Sí
III	Escaneo automatic con “Nikto”	Sí
Total de detecciones		**9/20**

6.5 Análisis de los resultados

Sí analizamos en detalle los resultados obtenidos y los estudiamos de forma agrupada por tipología de intrusión, tenemos que la herramienta es capaz de detectar con mayor eficacia los ataques más intrusivos. Estos son los ataques que contienen payloads inyectados con código javaScript o SQL incrustado dentro de las URIs. En el siguiente gráfico podemos observar este tipo de comportamiento:

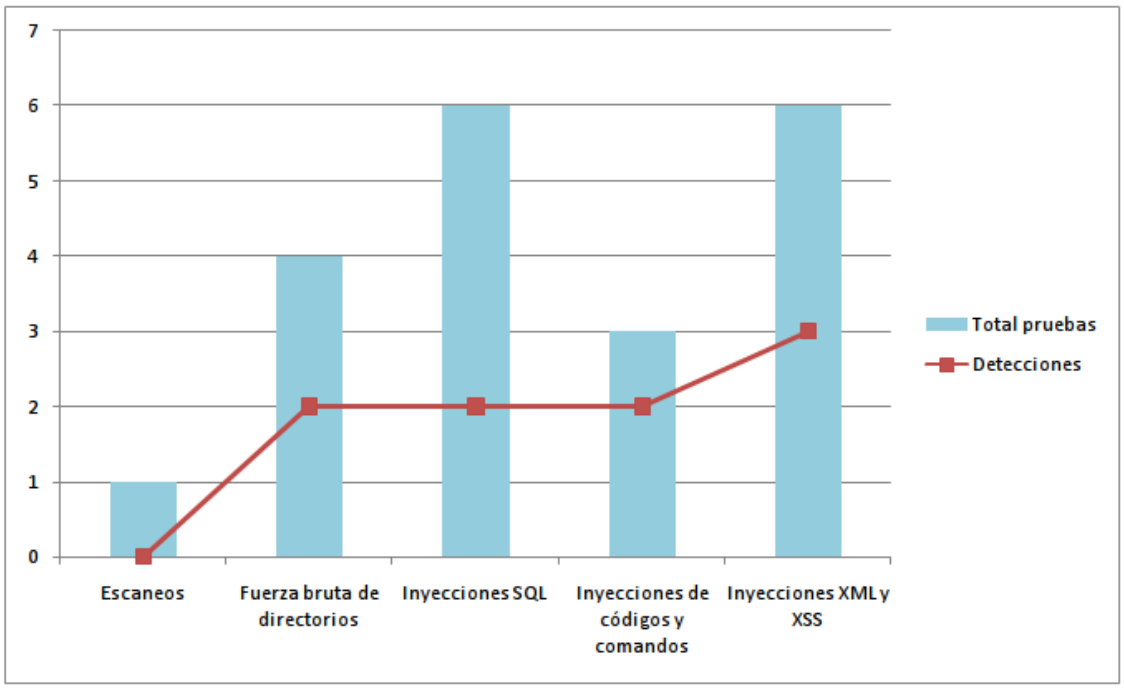


Figura 28: análisis de resultados. Volumen de ataques y detecciones.

Siendo más analítico con los datos se puede indicar que la herramienta ha sido capaz de identificar una cantidad inferior al 50% de los ataques, pasando de

forma transparente gran parte de estos. En concreto la herramienta sólo realiza detecciones a medida que van avanzando las fases de un ataque de intrusión. En los reconocimientos previos (fases I y II) la herramienta apenas es capaz de ver el acceso a determinados ficheros a través de ataques de tipo “Path trasversal”. Por el contrario, en la fase III es dónde la herramienta es más últil y es capaz de detectar un número de ataques razonables.

Igualmente, cómo se puede ver en el punto “6.3.4 Escaneos automatizados con herramientas”, los ataques con herramientas de escaneos de vulnerabilidades se detectan sin problemas tanto a través del “UserAgent”, cómo por la cantidad de eventos que generan en la herramienta.

En último lugar, se adjunta la tabla resumen agrupada por tipologías de ataques:

RESUMEN DE DETECCIONES POR TIPOLOGÍA		
Juego de pruebas realizadas	**Total**	**Detecciones**
Escaneos	1	0
Fuerza bruta de directorios	4	2
Inyecciones SQL	6	2
Inyecciones de códigos y comandos	3	2
Inyecciones XML y XSS	6	3
TOTALES	**20**	**9**

7. Conclusiones

7.1 Premisas

Antes de sacar conclusiones sobre las capacidades de la herramienta hay que tener en cuenta que esta se basa en la detección de patrones en registros de eventos ubicados en ficheros. Para que la herramienta funcione adecuadamente debe estar bien configurado el nivel de "logging" o de "debug" de los diferentes servidores webs. En caso de que esto no fuese así, la herramienta no sería capaz de detectar nada. Por otro lado, con el nivel de "debug" configurado en nuestro entorno vulnerable no somos capaces de ver las peticiones de tipo "POST", en consecuencia sólo podemos intentar detectar patrones en las "URIs" de las peticiones.
Por tanto, más allá de los problemas de diseño y detección de patrones que pueda tener nuestra herramienta, esta depende de las trazas de terceros para tener una funcionalidad completa.

7.2 Conclusiones de los resultados

Analizando en profundidad los resultados podemos sacar una serie de conclusiones relativas a la herramienta diseñada. A continuación se enumeran cada una de estas:

1. Tomando en cuenta la experiencia del autor en proyectos previos [9], se puede concluir que esta tasa de detección es buena comparada con otros productos que existen en el mercado tanto en software libre cómo privativo. En el siguiente gráfico se puede ver los porcentajes de detección que ha tenido la herramienta, estos superan el 40%.

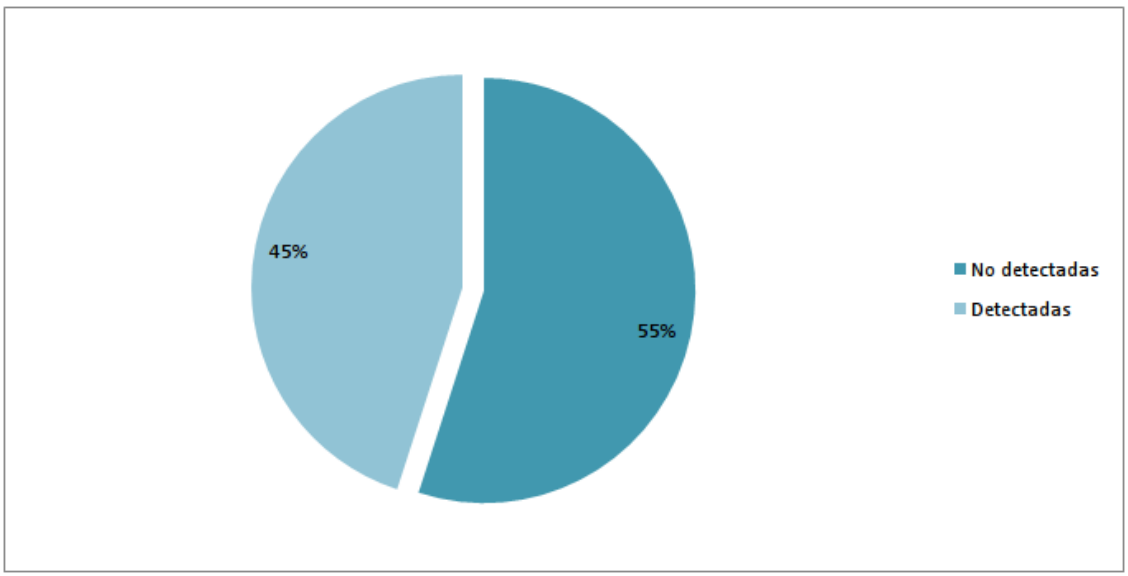


Figura 29: Porcentaje de detecciones respecto el total de pruebas realizadas.

2. La herramienta tiene una limitación clara y la dependencia en gran medida de los registros que se almacenan de las páginas webs. Esto hace que la

herramienta sólo sea buena detectando sí los logs se almacenan adecuadamente y con la mayor cantidad de payloads posibles.

3. La herramienta no es capaz de detectar ataques en el método “POST” del protocolo HTTP.

4. La herramienta es especialmente útil detectando pruebas masivas cómo realizan las herramientas automáticas. Los intentos de intrusión más “finos” son mucho más costosos de encontrar y esta utilidad no tiene todavía todos estos tipos de patrones o firmas diseñadas para detectarlos.

Resumiendo, la herramienta diseñada tiene una buena tasa de detección y es especialmente útil cómo último nivel de detección, es decir, se trata de una herramienta que puede detectar ataques en los servidores webs que han sido capaces de traspasar las defensas perimetrales. Adicionalmente, esta herramienta es buena para utilizarse con fines forenses, ya que podría analizar los logs de un servidor para ver sí ha habido intrusiones pretéritas.

7.3 Análisis del seguimiento de la planificación y metodología

Revisando los tiempos de ejecución del trabajo se puede indicar que la mayoría de los hitos se han cumplido en tiempo y forma. El único hito que se ha realizado con cierto retraso es el que estaba relacionado con las pruebas de intrusión. Este hito no se ha podido finalizar hasta dos semanas antes de la entrega final, esto ha sido debido a que en última instancia se decidió anañdir un correlador de eventos de seguridad para cerrar el círculo y de este modo facilitar la visualización de los payloads.

Esto ha hecho que aunque no se hayan modificado hitos a nivel temporal, sí se ha ido justo de tiempo para la finalización del proyecto. Por otro lado, en cuanto la metodología no ha sido necesario revisarla ni retocarla.

Finalmente, hablando de los productos se ha conseguido lo que se esperaba, una herramienta que puede detectar a nivel de “host” o equipo final ataques de tipo web.

7.4 Evolución del proyecto

La línea más lógico de evolución de este proyecto sería probar la herramienta de detección en un entorno real con defensas perimetrales. De este modo, seríamos capaces de comprobar los siguientes hechos:

- Cuantos ataques se paran en las defensas perimetrales.
- Cuantos ataques traspasan la defensa y se detectan por esta herramienta diseñada.
- Cuantos ataques traspasan y llegan al servidor sin detección alguna.

Por tanto, cómo posible línea de trabajo futura se podría integrar en el entorno que se diseño en un proyecto anterior [9], dónde había servidores vulnerables y un entorno de detección basado en tecnología IPS y firewall. El esquema de ambos trabajos conjuntos quedaría de la siguiente manera:

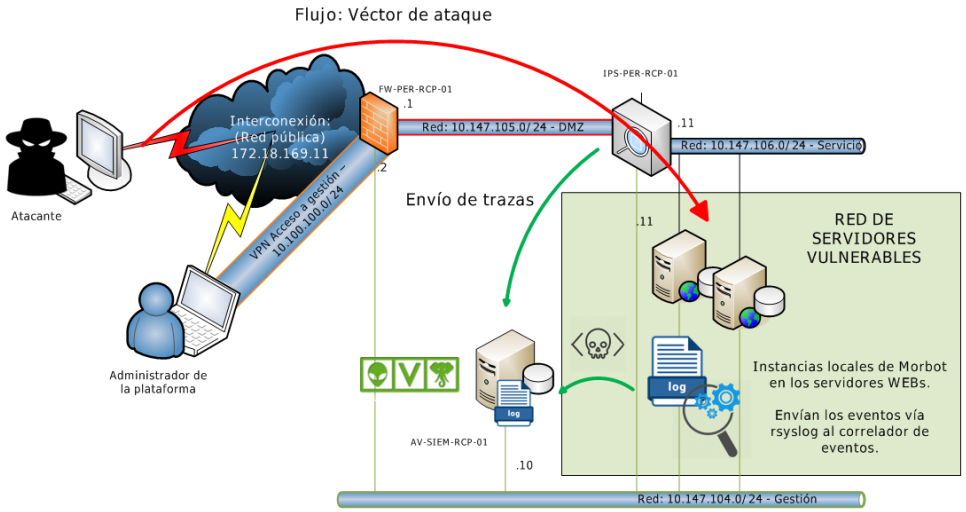


Figura 30: inclusión de la herramienta de detección en un entorno con seguridad perimetral.

8. Glosario

DNS – Domain Name Server

FTP – File Transfer Protocol

HTML – Hypertext Markup Language

IP – Internet Protocol

IDS – Intrusion Detection System

IPS – Intrusion Prevention System

LAMP – Linux Apache MySQL y PHP.

NAT – Network Address Translation

NTP – Network Time Protocol

NG – Next Generation

OSSIM – Open Source SIEM

OWASP – Open Web Application Security Project

QOS – Quality of Service

SIEM – Security information and event management

SQL – Structured Query Language

SSH – Secure Shell

TCP – Transmission Control Protocol

UTM – Unified threat management

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

UDP – User Datagram Protocol

VLAN – Virtual Local Area Network

VPN – Virtual Private Network

WAF – Web application firewall

XML – Extensible Markup Language

XSS – Cross-Site Scripting

9. Bibliografía

9.1 Bibliografía general:

[1] Nicholas Pappas, (2008). Network IDS & IPS Deployment Strategies. SANS Institute, 2008.

[2] Sunil Gupta, (2012). Logging and Monitoring to Detect Network Intrusions and Compliance Violations in the Environment, http://www.sans.org/reading-room/whitepapers/detection/logging-monitoring-detect-network-intrusions-compliance-violations-environment-33985, (July 4, 2012).

[3] GUÍA DE PRUEBAS OWASP, 2008 V3.0. OWASP Foundation, (2008).

[4] Alienvault OSSIM, https://www.alienvault.com/products/ossim.

[5] ModSecurity, https://www.modsecurity.org/

[6] OWASP Best Practices: Use of Web Application Firewalls, https://www.owasp.org/index.php/Category:OWASP_Best_Practices:_Use_of_Web_Application_Firewalls

[7] Venkatesh Sundar, (February 5, 2014), WAF Signatures. https://www.indusface.com/blog/beating-waf-signatures/

[8] Mariusz Stawowski, Bartosz Krynski, (2011), Guidelines for Web applications Application Firewall. http://clico.pl/services/Guidelines_for_Web_application_protection.pdf

[9] Roberto Carlos Pérez González, (July 16 2016), Los sistemas de seguridad perimetral y principales vectores de ataque web. http://openaccess.uoc.edu/webapps/o2/bitstream/10609/52986/6/rperezgonTFG0716mem%C3%B2ria.pdf

10. Anexos

10.1 Código fuente “Morbot – Necromancer”.

10.1.1 Código fuente código principal (“nec_analyzer.py”).

https://github.com/rororperez/Morbot/blob/master/nec_analyzer.py

10.1.2 Código fuente clase de procesado (“zombie_analyzer_class.py”).

https://github.com/rororperez/Morbot/blob/master/zombie_analyzer_class.py

10.1.3 Código fuente patrones de ataques

https://github.com/rororperez/Morbot/blob/master/XML_attack_filter.xml
https://github.com/rororperez/Morbot/blob/master/XML_useragent_filter.xml

10.1.4 Código fuente formato soportados (“XML_uri_format.xml”).

https://github.com/rororperez/Morbot/blob/master/XML_uri_format.xml

10.2 Fichero de configuración “Morbot – Necromancer”.

El fichero de configuración publicado es el siguiente:

https://github.com/rororperez/Morbot/blob/master/necromancer_config.py

En nuestro caso, se ha adaptado con los siguientes parámetros.

```
#############################################################################################################
###
### Fichero de configuracion de Morbot
### Importante: tener todos los ficheros de patrones en el mismo directorio que
###             el fichero de configuracion
###
#############################################################################################################
################################## Morbot Necromancer config ################################################
#############################################################################################################

# Log files:
LOG = "morbot.log"
input_LOGFILE = "access_in.log"
output_LOGFILE_01 = "morbot01.log"
output_LOGFILE_02 = "morbot02.log"
output_LOGFILE_03 = "morbot03.log"
output_LOGFILE_04 = "morbot04.log"
output_LOGFILE_05 = "morbot05.log"
output_LOGFILE_06 = "morbot06.log"
output_LOGFILE_07 = "morbot07.log"
output_LOGFILE_08 = "morbot08.log"

#############################################################################################################
#############################################################################################################
#############################################################################################################
```

```
# XML Pattern with all regexp and regex to exclude

# Attack rules:
attack_PATTERNS = "XML_attack_filter.xml"
useragent_PATTERNS = "XML_useragent_filter.xml"

# Exclude attacks by ID:
exclude_ids = [7,8]

#############################################################################################################
#############################################################################################################
#############################################################################################################

# Parser rules:
format_rules = "XML_uri_format.xml"

# Tecnology - Apache, F5, ...
tecnology = "process-all"

#############################################################################################################
#############################################################################################################
#############################################################################################################

# Set General configs

# 5 by default, number of cores that process go to use
# Don't support more of 8 cores
number_of_cores = 4

# Number of second to wait after check the file again
loop_interval = 1

# load first priority signatures like SQLi and others 0 - NO, 1 - YES
priority_signatures = 1

# Activate userAgent Engine detection 0 - disable, 1 - enable
useragent_engine_detection = 1

#############################################################################################################
```

10.3 Script para crear la base de datos de noticias del entorno web vulnerable (crear_ddbb.sql).

```
create database SQLiWeb;

create table SQLiWeb.USERS(email varchar(100) primary key, name varchar(100), password
varchar(100), accountId integer);
insert into SQLiWeb.USERS values ("admin@rcpg.com", "admin", "password_admin", 0);
insert into SQLiWeb.USERS values ("roberto@rcpg.com", "roberto", "password_roberto", 1);
insert into SQLiWeb.USERS values ("carlos@rcpg.com", "carlos", "password_carlos", 2);

create table SQLiWeb.NEWS(id integer primary key auto_increment, title varchar(100), body text,
datetime datetime);
insert into SQLiWeb.NEWS values (1, "Noticia 1", "Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt
mollit anim id est laborum.", "2017-12-10 00:00:01");
insert into SQLiWeb.NEWS values (2, "Noticia 2", "Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt
mollit anim id est laborum.", "2017-12-10 00:00:02");
insert into SQLiWeb.NEWS values (3, "Noticia 3", "Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
```

```
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt
mollit anim id est laborum.", "2017-12-10 00:00:03");
```

10.4 Plugin tecnología “Morbot-Necromancer” para OSSIM

10.4.1 Plugin de AlienVault / Ossim:

Ubicado en: /etc/ossim/agent/plugins/nec-ids_alienvault_plugin.cfg

```
;;
;; plugin_id: 8068
;;
;; Roberto Carlos Perez Gonzalez
;; 22-05-2018
;; Morbot - Necromancer, AlienVault plugin
;;

[DEFAULT]
plugin_id=8068
dst_port=80

[config]
type=detector
enable=yes

source=log
location=/var/log/remote_morbot.log

# create log file if it does not exists,
# otherwise stop processing this plugin
create_file=false

process=    ; change by apache|httpd|etc.
start=no  ; launch plugin process when agent starts
stop=no     ; shutdown plugin process when agent stops
startup=
shutdown=

# list of sids (comma separated) to be excluded by the detector
exclude_sids=

[translation]

#
# Custom logs formats defined in apache2.conf
# To see variable definition: http://httpd.apache.org/docs/2.2/mod/mod_log_config.html#formats
#

[00 - Morbot - apache-access]
#May 20 12:52:53 TFM-equipo-pruebas MorBot Pattern found ID: 80; RAW line: 192.168.58.1 - -
[11/Dec/2017:22:49:24 +0100] "GET /news.php?id=1%20OR%20union%20select%20from HTTP/1.1" 200 400
"-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0"

event_type=event
regexp=(?P<date>\w+\s+\d{1,2}\s+\d{1,2}:\d{1,2}:\d{1,2})\s+(?P<remote_host>\S+)\s+(?P<applicatio
n>\S+)\s+Pattern             found             ID:\s+(?P<plugin_sid>\d+);\s+RAW             line:
(?P<src_ip>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}).*?\"(?P<method>\w+)\s+(?P<payload>.*?)\"\s+(?P<re
sponse_code>\d+)\s+\S+\s+\S+\s+\"(?P<useragent>.*?)\".*
sensor={resolv($remote_host)}
src_ip={resolv($src_ip)}
dst_ip=192.168.58.158
date={normalize_date($date)}
plugin_sid={$plugin_sid}
userdata1={$method}
userdata2={$payload}
userdata3={$response_code}
userdata4={$useragent}
userdata8=MAQUETA_TFM
```

10.4.2 SQL para cargar eventos en AlienVault / Ossim:

https://github.com/rororperez/Morbot/blob/master/alienvault-integration/nec-ids_plugin_sid.sql